Distance and kernel based learning over composite representations

Woznica, Adam

UNIVERSITÉ DE GENÈVE

FACULTÉ DES SCIENCES

Département d'informatique

Professeur Christian Pellegrini
Docteur Mélanie Hilario
Docteur Alexandros Kalousis

Distance and Kernel Based Learning over Composite Representations

THÈSE

présentée à la Faculté des sciences de l'Université de Genève
pour obtenir le grade de Docteur ès sciences, mention informatique

par

Adam Woźnica

de

Łask (Pologne)

These N^o 4013

GENÈVE
ReproMail
2008

To my family

Nauczyli mnie mnóstwa mądrości,
Logarytmów, wzorów i formułek,
Z kwadracików, trójkącików i kółek
Nauczyli mnie nieskończoności.

...

Wiem o kuli, napełnionej lodem,
O bursztynie, gdy się go pociera...
Wiem, że ciało, pogrążone w wodę
Traci tyle, ile...et cetera.

...

I nic nie wiem, i nic nie rozumiem,
I wciąż wierzę biednymi zmysłami,
Że ci ludzie na drugiej półkuli
Muszą chodzić do góry nogami.

...

Nauka - J. Tuwim

Remerciements

J'ai maintenant le plaisir et la joie de saisir cette occasion pour remercier toutes les personnes qui m'ont accompagné et m'ont soutenu au cours des années de travail qui ont conduit à cette thèse.

Tout d'abord je tiens à remercier Prof. Christian Pellegrini, Dr. Mélanie Hilario et Dr. Alexandros Kalousis pour m'avoir accepté comme doctorant au sein du Groupe d'Intelligence Artificielle de l'Université de Genève. Venir à Genève en 2003 a vraiment élargi mes horizons et préparé le terrain pour l'état actuel des choses. Je suis particulièrement reconnaissant à Alexandros Kalousis pour son soutien, sa bonne volonté, ainsi que pour m'avoir mis la pression nécessaire sur différentes phases de mes études. Il m'a appris à me poser des questions de recherche, à exprimer des idées et m'a montré différentes façons d'aborder un problème de recherche. Il m'a aidé à comprendre le sens de la discipline scientifique.

Je suis profondément reconnaissant au professeur Pavel Brazdil de l'Université de Porto qui, il y a presque huit ans, m'a présenté les domaines fascinants de l'apprentissage automatique et de la fouille de données (data mining). Il m'a fourni les conseils et les ressources dont j'avais besoin, et grâce à son soutien j'ai pu faire mon doctorat ici à Genève.

Un remerciement spécial va à tous les autres membres du Groupe d'Intelligence Artificielle grâce auxquels j'ai pu évolué dans un environnement stimulant, en particulier pour toutes les (longues) discussions que nous avons eu. Je voudrais mentionner ici Frédéric Ehrler, James Henderson, Jee-Hyub Kim, Julien Prados, Ivan Titov et Antonio Jimeno Yepes.

Je tiens également à remercier les membres de mon comité de thèse, Prof. Stefan Kramer et Dr. Tamás Horváth pour leurs conseils et commentaires.

Je remercie également ma famille pour le soutien constant et ses encouragements. En particulier, je suis à jamais redevable à mes parents - je leur dois beaucoup pour ce que je suis devenu. Mes parents m'ont non seulement donné la vie, mais aussi l'ont remplie avec tout l'amour infini que l'on puisse désirer.

Enfin (mais pas des moindres), j'exprime mes sincères remerciements à ma charmante épouse Aldona pour sa compréhension, sa patience et son soutien durant tout ce temps. J'ai rencontré Aldona au tout début de mon séjour à Genève. Et à travers chaque étape de la préparation de ma thèse, elle a partagé les charges de cette étude, en particulier lors des nombreux jours pendant lesquels j'ai passé plus de temps avec mon ordinateur qu'avec elle. Je suis particulièrement recon-

naissant à mon épouse de m'avoir donné le cadeau le plus précieux que je puisse imaginer - Michał mon fils - qui est né seulement deux jours après que j'ai soutenu ma thèse. Donia, je ne peux pas imaginer la vie sans toi!

Contents

Remerciements **v**

List of Tables **xv**

List of Figures **xxii**

List of Algorithms **xxiii**

Résumé **xxv**

Abstract **xxvii**

1 Introduction **1**
1.1 Limitations of the Existing Approaches 3
1.2 Contributions 8
1.3 Reading Guide 10
1.4 Bibliographical Note 12

2 Representation Language **13**
2.1 Introduction of the Basic Concepts 15
2.1.1 Primitive Attributes 16
2.1.2 Tuples 16
2.1.3 Sets 16
2.1.4 Lists 17
2.1.5 General Composite Objects 17
2.1.6 Operators 19
2.2 Relational Representation 19
2.2.1 Relational Algebra 20
2.2.2 Relational Instance 21
2.2.3 An Example 24
2.3 Graphs and Trees 28
2.3.1 Representation of Graphs 30
2.4 Operators over Relational Representations 33
2.5 Related Work 39
2.6 Conclusions 42

3 Distances **45**
3.1 Preliminaries . . . 47
3.2 Distances over Relational Building Blocks . . . 49
3.2.1 Distances on Primitive Attributes . . . 49
3.2.2 Distances on Tuples . . . 50
3.2.3 Distances on Sets . . . 51
3.2.4 Distances on Lists . . . 59
3.2.5 Normalization . . . 61
3.3 Distances on Trees and Tree-like Structures . . . 62
3.4 Properties of the Relational Distance . . . 64
3.5 Experiments . . . 65
3.5.1 K-Nearest Neighbor . . . 65
3.5.2 Experimental Setup . . . 66
3.5.3 Datasets . . . 67
3.5.4 Classification Performance . . . 68
3.5.5 Families of Set Distance Measures . . . 74
3.5.6 Comparison with Other Systems . . . 76
3.5.7 Selecting Complex Distances . . . 78
3.5.8 Parameter Selection for the Tanimoto Distance . . . 80
3.6 Related Work . . . 83
3.7 Conclusions . . . 88

4 Kernels **91**
4.1 Preliminaries . . . 93
4.2 Kernels over Relational Building Blocks . . . 95
4.2.1 Kernels on Primitive Attributes . . . 95
4.2.2 Kernels on Tuples . . . 95
4.2.3 Cross Product Kernel on Sets . . . 99
4.2.4 Kernels on Sets Based on Mappings . . . 101
4.2.5 Kernels on Lists . . . 108
4.3 Kernels on Trees and Tree-like Structures . . . 109
4.4 Experiments . . . 110
4.4.1 Support Vector Machines . . . 111
4.4.2 Experimental Setup . . . 113
4.4.3 Experiments with Cross Product Kernel . . . 114
4.4.4 Experiments with Set Kernels Based on Mappings . . . 121
4.4.5 Experiments with Kernels on Lists . . . 131
4.5 Related Work . . . 133
4.6 Conclusions . . . 139

5 Adaptive Approaches **141**
5.1 Adaptive Methods . . . 143
5.1.1 General Optimization Method . . . 143
5.1.2 Xing's Method . . . 145
5.1.3 MCML . . . 146
5.1.4 NCA . . . 147

5.1.5 Regularization . . . 147
5.1.6 Solving Optimization Problems . . . 148
5.1.7 Computational Complexity . . . 149
5.2 Application Scenarios . . . 149
5.2.1 Graph Kernels . . . 150
5.3 Experiments on Distance Measure Learning . . . 154
5.3.1 Experimental Setup . . . 154
5.3.2 Results and Analysis . . . 156
5.4 Experiments on Representation Learning . . . 161
5.4.1 Experimental Setup . . . 162
5.4.2 Results and Analysis . . . 163
5.5 Related Work . . . 175
5.6 Conclusions . . . 178

6 Overview, Discussion and Future Work 181
6.1 Future Work . . . 185
6.1.1 Extracting Relational Representations . . . 185
6.1.2 Cost Functions . . . 186
6.1.3 Regularization . . . 187
6.1.4 Learning with Side-Information . . . 187
6.1.5 Adaptive Distances on Sets . . . 188
6.1.6 Theoretical Analysis of Mapping-Based Set Kernels . . . 191

Bibliography 210

A Datasets 211
A.1 Sets . . . 212
A.2 Graphs . . . 213
A.3 General relational structures . . . 214

B Detailed Experimental Results 219
B.1 Detailed Experimental Results for Distances . . . 219
B.2 Detailed experimental for Kernels . . . 228
B.2.1 Results for CP kernel using SVMs . . . 228
B.2.2 Results for CP kernel using kNN . . . 235
B.3 Spectras of Gram Matrices . . . 242
B.4 Visualizations of distance combinations . . . 244
B.5 Visualizations of representation combinations . . . 257

C List of Notation 271

List of Tables

2.1 The example database corresponding to the relational schema from Figure 2.4. 26

3.1 Characterization of the different distance measures according to the properties they satisfy. 57
3.2 Characterization of the different distance measures according to their complexities (n is the set cardinality). 59
3.3 Accuracy and rank results (the first parenthesis) on the datasets where instances are represented as sets of vectors. The ranking is based on the scheme described in Section 3.5.2. The sign in the second parenthesis compares the performance of the corresponding distance measures with that of the default classifier (Def. Acc.). 68
3.4 Accuracy and rank results (the first parenthesis) on the graph datasets. For each dataset the performance for two different representation is given. The ranking is based on the scheme described in Section 3.5.2. The sign in the second parenthesis compares the performance of the corresponding distance measures with that of the default classifier (Def. Acc.). The sign in the third parenthesis for version 1 of datasets compares the performance with version 2. 69
3.5 Average ranks of set distances over all the datasets. The ranking is based on the scheme described in Section 3.5.2. 70
3.6 Accuracy and rank (the first parenthesis) results on the protein fingerprint dataset, both with and without weights. The ranking is based on the scheme described in Section 3.5.2. The second parenthesis contains ranks results where d_{edit} is not considered (when computing the ranks from the first parenthesis, d_{edit} was included). The sign in the third parenthesis compares the performance of the corresponding distance measures with that of the default classifier (Def. Acc.). The sign in the fourth parenthesis for weighted dataset compares the weighted with non-weighted versions. 72
3.7 Accuracy results of the best (k_{Best}) and CV (k_{CV}) distances together with performances of other relational systems. The references of the specific systems are given in the text. 79

3.8 Effect of parameter tuning on classification performance of the Tanimoto distance. The sign in the "sig" column indicates whether the performance of the best threshold value was significantly better than that of the default (the + sign) or there was no significant difference (the = sign). 82

4.1 Accuracy and rank results (SVM) for k_{CP} on the datasets where instances are sets of vectors. 115
4.2 Accuracy and rank results (SVM) for k_{CP} on the graph and protein fingerprints datasets. 116
4.3 Accuracy and rank results (kNN) for k_{CP} on the datasets where instances are sets of vectors. 117
4.4 Accuracy and rank results (kNN) for k_{CP} on the graph and protein fingerprints datasets. 118
4.5 Accuracy results of the best kernel from this section (SVM_{best}) together with the best distance (kNN_{Best}) and other related relational kernels ("Kernel-based"). Additionally, we provide the best results obtained using the kernels from Section 4.4.4. The references of the specific systems ("Kernel-based" column) are given in Section 3.5.6. 120
4.6 Accuracy, ranks and significance test results for set kernels in diterpenes and both versions of the musk dataset. The description of the notation is given in the text. 123
4.7 Accuracy, ranks and significance test results for set kernels in duke, muta and FM datasets. The description of the notation is given in the text. 124
4.8 Accuracy, ranks and significance test results for set kernels in FR, MM and MR datasets. The description of the notation is given in the text. 125
4.9 Accuracy, ranks and significance test results on the considered datasets. The first parenthesis corresponds to the comparison of SVM_{P} vs. kNN_{P} and the second compares SVM_{P} vs. kNN. . . . 130
4.10 Accuracy, rank and significance test results (SVM) on the weighted version of the protein fingerprints dataset. 132

5.1 Worst-case complexity of related kernels based on walks and trees. 154
5.2 Accuracy and significance test results of the kNN_{DC} algorithm. The + sign stands for a significant win of the first algorithm in the pair, - for a significant loss and = for no significant difference. The signs in the first parenthesis correspond to the comparison of kNN_{DC} vs. kNN where the best distance is used (the kNN_{Best} column) and the second to the comparison of kNN_{DC} vs. kNN with cross-validation (the kNN_{CV} column). 155

5.3 Accuracy and significance test results of SVM in the graph datasets using $\text{MCML}_{\text{full}}$ (the + sign stands for a significant win of the first algorithm in the pair, - for a significant loss and = for no significant difference). The sign in the first parenthesis corresponds to the comparison of SVM vs. SVM with $k_{PD_{tuple,\boldsymbol{A}}}$ with d_{AL}, while the sign in the second parenthesis compares SVM vs. SVM with k_{CP}. 168
5.4 Accuracy and significance test results of SVM in the graph datasets using NCA_{full} (the + sign stands for a significant win of the first algorithm in the pair, - for a significant loss and = for no significant difference). The sign in the first parenthesis corresponds to the comparison of SVM vs. SVM with $k_{PD_{tuple,\boldsymbol{A}}}$ with d_{AL}, the sign in the second parenthesis compares SVM vs. SVM with k_{CP}, and finally the sing in the last parenthesis compares NCA_{full} with $\text{MCML}_{\text{full}}$ from Table 5.3. 169
5.5 Comparison with other related graph kernels. The best kernels in each datasets are emphasized. The results for the best Cross Product Kernel are taken from Section 4.4.3; the results the best kernel based on mappings are taken form Section 4.4.4. 175

A.1 Datasets used in this work. 211
A.2 Datasets where instances are represented as sets of vectors. Minimum, maximum and median number of sets cardinalities is given by min., max. and median, respectively. 212
A.3 Various statistics of the graph datasets. # pos. and # neg. denote the number of positive and negative examples, respectively. Minimum, maximum and median number of atoms is given by min. $|G(\mathcal{V})|$, max. $|G(\mathcal{V})|$ and med. $|G(\mathcal{V})|$. Minimum, maximum and median number of bonds is given by min. $|G(\mathcal{E})|$, max. $|G(\mathcal{E})|$ and med. $|G(\mathcal{E})|$. 214

B.1 Detailed results of McNemar's test of statistical significance for kNN in diterpenes and duke. 220
B.2 Detailed results of McNemar's test of statistical significance for kNN in musk. 221
B.3 Detailed results of McNemar's test of statistical significance for kNN in mutagenesis. 222
B.4 Detailed results of McNemar's test of statistical significance for kNN in FM. 223
B.5 Detailed results of McNemar's test of statistical significance for kNN in FR. 224
B.6 Detailed results of McNemar's test of statistical significance for kNN in MM. 225
B.7 Detailed results of McNemar's test of statistical significance for kNN in MR. 226

B.8 Detailed results of McNemar's test of statistical significance for kNN in Protein Fingerprints. 227

B.9 Detailed results of McNemar's test of statistical significance for SVM in diterpenes and duke. 228

B.10 Detailed results of McNemar's test of statistical significance for SVM in musk. 229

B.11 Detailed results of McNemar's test of statistical significance for SVM in mutagenesis. 230

B.12 Detailed results of McNemar's test of statistical significance for SVM in FM. 231

B.13 Detailed results of McNemar's test of statistical significance for SVM in FR. 232

B.14 Detailed results of McNemar's test of statistical significance for SVM in MM. 233

B.15 Detailed results of McNemar's test of statistical significance for SVM in MR. 234

B.16 Detailed results of McNemar's test of statistical significance for kNN in diterpenes and duke. 235

B.17 Detailed results of McNemar's test of statistical significance for kNN in musk. 236

B.18 Detailed results of McNemar's test of statistical significance for kNN in mutagenesis. 237

B.19 Detailed results of McNemar's test of statistical significance for kNN in FM. 238

B.20 Detailed results of McNemar's test of statistical significance for kNN in FR. 239

B.21 Detailed results of McNemar's test of statistical significance for kNN in MM. 240

B.22 Detailed results of McNemar's test of statistical significance for kNN in MR. 241

B.23 Significance test results comparing single decompositions vs. combination of decompositions. The values in the + and - columns denote the number of cases when MCML (NCA) are significantly better and worse than the single decompositions, respectively; the values in the = column correspond to the numbers of cases when statistically significant differences were not observed. The total number of comparisons for MCML (or NCA) is 165 (i.e. 11 walk decompositions x 11 set distances + 4 tree decompositions x 11 set distances). 259

B.24 Significance test results comparing *heterogeneous* decompositions (i.e. into both walks and tress) vs. *homogeneous* decompositions (i.e. either into walks or trees). The values in the + and - columns denote the number of cases when the heterogeneous decompositions are significantly better and worse than the both homogeneous decompositions, respectively; the values in the = column correspond to the numbers of cases when the performances of the heterogeneous and both homogenious decompositions are statistically equivalent. The total number of comparisons for MCML (or NCA) is 11 (i.e. total number of set distances). 259
B.25 Significance test results comparing simple weighting schemes (i.e. *isotropic* and *down-weighted*) vs. the adaptive ones (i.e. MCML and NCA). The values in the + and - columns denote the number of cases when simple weighting schemes were significantly better and worse than MCML (or NCA), respectively; the values in the = column correspond to the numbers of cases when the performances of the simple weighting schemes and the adaptive ones were statistically equivalent. The number of comparisons is 11 (i.e. total number of set distances). 270

List of Figures

1.1 A reading guide to the thesis. The alternative way of reading this thesis, represented by the red line, is intended for "impatient" readers. 11

2.1 Schematic structure of Chapter 2. The recommended sections to read are highlighted. 14
2.2 Examples of two trees corresponding to complex objects where only sets (a) and only lists (b) are used. 18
2.3 An example of a mixed tree. 19
2.4 A relational schema corresponding to a sample database from the domain of proteomics. Referenced keys are marked in **bold**, foreign keys in *italics*. 26
2.5 The relational instance that corresponds to protein Pa. The oblique lines indicate where recursion ends due to the appearance of a self replicating loop. 27
2.6 Two relational representations of labeled graphs. The first one (a) corresponds to the decomposition into trees while the second one (b) corresponds to decompositions into tree-like structures. . 33

3.1 Schematic structure of Chapter 3. The most important sections are highlighted. 46
3.2 Triangle inequality violations for $A = \{a, c\}$, $B = \{b\}$ and $C = \{d\}$. 53
3.3 Specific pairs of elements defined by F for the considered set distance measures. 56
3.4 Clustering the relational distances with respect to their rank performance among the datasets examined. 75
3.5 Distributions of set cardinalities for the two versions of musk. . 78
3.6 Behavior of the Tanimoto distance with respect to its θ parameter. Chart (a) presents datasets where instances are sets of vectors, chart (b) gives performance for the mutagenesis and protein fingerprints datasets, and chart (c) presents results for the carcinogenicity datasets. 81

4.1 Schematic structure of Chapter 4. The sections with our main contributions are highlighted. 92

4.2 Spectra of Gram matrices of set kernels used within SVM_{SP}. The solid line denotes the r ratio from Equation 4.35 (left y-axis) and bars denote the estimated accuracies (right y-axis). 127

4.3 Spectra of Gram matrices of set distance substitution kernels. The solid line denotes the r ratio from Equation 4.35 (left y-axis) and bars denote the estimated accuracies (right y-axis). The γ parameter is fixed to 1. 128

5.1 Schematic structure of Chapter 5. 142

5.2 A schematic description of the basic building components of the proposed graph kernel. G_1, G_2 are any two graphs, $\mathcal{G}_i^{t_k}$ is the decomposition of the G_i graph to substructures of type t_k, $d_{set}(\mathcal{G}_1^{t_k}, \mathcal{G}_2^{t_k})$ is a set distance between the sets of decompositions of G_1 and G_2 to substructures of type t_k, $d^2_{tuple,\boldsymbol{A}}(G_1, G_2)$ is the final learned weighted distance that combines different types of decompositions. 152

5.3 Relative importance of the different set distance measures, computed by $\text{METHOD}_{\text{diag}}$ ($\lambda = 0$), for the diterpenes and musk (ver. 1) datasets. The weights are learned on the full training set. For each set distance measure we also provide in parenthesis the corresponding performance of kNN. 157

5.4 Stability of the distance combination techniques in musk (ver. 1). Weights in each of the 10 folds (i.e. rows) are normalized by a Frobenius norm of $\boldsymbol{A}$ ($\boldsymbol{W}$). 158

5.5 Relative importance of the different set distance measures, computed by NCA_{full} ($\lambda = 0$), for the diterpenes and mutagenesis datasets. The weights are computed on the full training set. For each set distance measure we also give the corresponding performance of kNN in parenthesis. 159

5.6 Relative importance of the different set distance measures as these are computed by $\text{MCML}_{\text{full}}$ and NCA_{full}, for different values of λ. The weights are computed in the diterpenes dataset and on the full training set. 159

5.7 The estimated accuracy for $\text{METHOD}_{\text{diag}}$ (using kNN) where $l = 1, \ldots, 6, 11$ top ranked distance measures (according to the assigned coefficients) are combined for computing the actual distance. Additionally, the performance of kNN_{Best} and kNN_{CV} is presented. 160

5.8 Estimated accuracy of different kernels in the proximity space ($k_{P,d_{set}}$) vs. different decompositions for different set distance measures in the mutagenesis dataset. wl denotes walks of length l whereas th denotes trees of height h. The last values in the plot denote the performance of $k_{Pd_{tuple,\boldsymbol{W}}}$ (and $k_{Pd_{tuple,\boldsymbol{A}}}$) where the different decompositions are combined. 164

5.9 Estimated accuracy (SVM) of different kernels in the proximity space, i.e. $k_{PD_{tuple,\boldsymbol{A}}}$ (for $\text{MCML}_{\text{full}}$) and $k_{PD_{tuple,\boldsymbol{W}}}$ (for NCA_{full}), combining only walks, only trees and both walks and trees vs. different set distance measures in the mutagenesis dataset. . . . 164

5.10 Percentage of empty sets vs. lengths of walks for the graph datasets (wl denotes walks of length l). 165

5.11 Medians of cardinalities vs. lengths of walks for the graph datasets (wl denotes walks of length l). 166

5.12 Relative importance of different decompositions (FM dataset) for the d_{SMD} set distance measure both for full and diagonal matrices $\boldsymbol{A}$ (for MCML) or $\boldsymbol{W}^T\boldsymbol{W} = \boldsymbol{A}$ (for NCA). wl denotes walks of length l whereas th denotes trees of height h. It is represented as normalized weights $\boldsymbol{A}$. Weights are normalized by a Frobenius norm of $\boldsymbol{A}$. 167

5.13 Relative importance of different decompositions (mutagenesis and FM datasets) for the d_{AL} set distance (wl denotes walks of length l). 171

5.14 Performances (using SVM) of the various weighting schemes for mutagenesis and MR datasets. 172

5.15 Accuracy of $k_{PD_{tuple,\boldsymbol{A}}}$ (using SVM) for the mutagenesis and FM datasets and for different set distance measures. The results are obtained using $\text{MCML}_{\text{diag}}$ where feature selection in the proximity space is performed using the CFS method (the first bar). For comparison we also report the accuracy of the $k_{PD_{tuple,\boldsymbol{A}}}$ on the full set of prototypes (the second bar). 173

5.16 Number of selected features (prototypes) by the CFS algorithm in the mutagenesis and FM datasets. The results are reported for different set distance measures and using $\text{MCML}_{\text{diag}}$. The first bar denotes the number of instances in the training set. 174

A.1 Schematic representation of a protein fingerprint. Each row is a protein sequence. Solid rectangles denote motifs. 215

A.2 Relational representation of protein fingerprints. 216

B.1 Spectra of Gram matrices of set distance substitution kernels. The solid line denotes the r ratio from Equation 4.35 (left y-axis) and bars denote the estimated accuracies (right y-axis). The γ parameter is fixed to 0.1. 242

B.2 Spectra of Gram matrices of set distance substitution kernels. The solid line denotes the r ratio from Equation 4.35 (left y-axis) and bars denote the estimated accuracies (right y-axis). The γ parameter is fixed to 10. 243

B.3 Relative importance of the different set distance measures, computed by $\text{METHOD}_{\text{diag}}$, in musk (ver. 2), duke, mutagenesis (ver. 1) and FM (ver. 1) datasets. The weights are computed on the full training set. For each set distance measure we also give the corresponding performance of kNN in parenthesis. 244
B.4 Relative importance of the different set distance measures, computed by $\text{METHOD}_{\text{diag}}$, in FR (ver. 1), MM (ver. 1) and MR (ver. 1) datasets. The weights are computed on the full training set. For each set distance measure we also give the corresponding performance of kNN in parenthesis. 245
B.5 Stability of the distance combinations techniques for diterpenes, musk (ver. 1), duke and mutagenesis (ver. 1). Weights in each of the 10 folds (i.e. rows) are normalized by a Frobenius norm of $\boldsymbol{A}$ ($\boldsymbol{W}$). 246
B.6 Stability of the distance combinations techniques in FM (ver. 1), FR (ver. 1), MM (ver. 1) and MR (ver. 1). Weights in each of the 10 folds (i.e. rows) are normalized by a Frobenius norm of $\boldsymbol{A}$ ($\boldsymbol{W}$). 247
B.7 Relative importance of the different set distance measures as these are computed by $\text{Xing}_{\text{full}}$, for different values of λ. The weights are computed in the diterpenes dataset and on the full training set. 248
B.8 Relative importance of the different set distance measures as these are computed by $\text{METHOD}_{\text{full}}$, for different values of λ. The weights are computed in the duke dataset and on the full training set. 248
B.9 Relative importance of the different set distance measures as these are computed by $\text{METHOD}_{\text{full}}$, for different values of λ. The weights are computed in musk (ver. 1) and on the full training set. 249
B.10 Relative importance of the different set distance measures as these are computed by $\text{METHOD}_{\text{full}}$, for different values of λ. The weights are computed in musk (ver. 2) and on the full training set. 250
B.11 Relative importance of the different set distance measures as these are computed by $\text{METHOD}_{\text{full}}$, for different values of λ. The weights are computed in FM (ver. 1) and on the full training set. 251
B.12 Relative importance of the different set distance measures as these are computed by $\text{METHOD}_{\text{full}}$, for different values of λ. The weights are computed in FR (ver. 1) and on the full training set. 252
B.13 Relative importance of the different set distance measures as these are computed by $\text{METHOD}_{\text{full}}$, for different values of λ. The weights are computed in MM (ver. 1) and on the full training set. 253

B.14 Relative importance of the different set distance measures as these are computed by $\text{METHOD}_{\text{full}}$, for different values of λ. The weights are computed in MR (ver. 1) and on the full training set. 254

B.15 The estimated accuracy in musk (ver. 2), duke, mutagenesis (ver. 1) and FM (ver. 1) for $\text{Xing}_{\text{diag}}$, $\text{MCML}_{\text{diag}}$ and NCA_{diag} (using kNN) where $l = 1, \ldots, 7, 11$ top ranked distance measures (according to the assigned coefficients) are combined for computing the actual distance. Additionally, the performances of kNN_{Best} and kNN_{CV} are given. 255

B.16 The estimated accuracy in FR (ver. 1), MM (ver. 1) and MR (ver. 1) for $\text{Xing}_{\text{diag}}$, $\text{MCML}_{\text{diag}}$ and NCA_{diag} (using kNN) where $l = 1, \ldots, 7, 11$ top ranked distance measures (according to the assigned coefficients) are combined for computing the actual distance. Additionally, the performances of kNN_{Best} and kNN_{CV} are given. 256

B.17 Estimated accuracy of different kernels in the proximity space ($k_{P,d_{set}}$) vs. different decompositions for different set distance measures in the FM and FR datasets (wl denotes walks of length l whereas th denotes trees of height h). The last values in the plot denote the performance of $k_{Pd_{tuple,\boldsymbol{A}}}$ (and $k_{Pd_{tuple,\boldsymbol{W}}}$) where the different decompositions are combined. 257

B.18 Estimated accuracy of different kernels in the proximity space ($k_{P,d_{set}}$) vs. different decompositions for different set distance measures in the MM and MR datasets (wl denotes walks of length l whereas th denotes trees of height h). The last values in the plot denote the performance of $k_{Pd_{tuple,\boldsymbol{A}}}$ (and $k_{Pd_{tuple,\boldsymbol{W}}}$) where the different decompositions are combined. 258

B.19 Estimated accuracy of different kernels in the proximity space, $k_{Pd_{tuple,\boldsymbol{A}}}$ (for $\text{MCML}_{\text{full}}$) and $k_{Pd_{tuple,\boldsymbol{W}}}$ (for NCA_{full}), combining only walks, only trees and both walks and trees vs. different set distance measures in the FM and FR datasets. 260

B.20 Estimated accuracy of different kernels in the proximity space, $k_{Pd_{tuple,\boldsymbol{A}}}$ (for $\text{MCML}_{\text{full}}$) and $k_{Pd_{tuple,\boldsymbol{W}}}$ (for NCA_{full}), combining only walks, only trees and both walks and trees vs. different set distance measures in the MM and MR datasets. 261

B.21 Various statistics represented in the form of boxplots on the cardinalities for decompositions into walks for the graph datasets (wl denotes decompositions into walks of length l). The boxes have lines at the lower quartile, median, and upper quartile values. The whiskers extend from each end of the box to the most extreme values in the data. 262

B.22 Relative importance of different decompositions (mutagenesis, FR, MM and MR dataset) for the d_{SMD} set distance measure both for diagonal and full matrices $\boldsymbol{A}$ (for MCML) or $\boldsymbol{W}^T\boldsymbol{W} = \boldsymbol{A}$ (for NCA). wl denotes walks of length l whereas th denotes trees of height h. It is represented as normalized weights $\boldsymbol{A}$. Weights are normalized by a Frobenius norm of $\boldsymbol{A}$. 263
B.23 Relative importance of different decompositions (FR, MM and MR datasets) for the d_{AL} set distance (wl denotes walks of length l). 264
B.24 Performances (using SVM) of the various weighting schemes for FM, FR and MM datasets. 265
B.25 Accuracy of $k_{PD_{tuple,\boldsymbol{A}}}$ (using SVM) in the FR, MM, MR datasets and for different set distance measures. The results are obtained using $\text{MCML}_{\text{diag}}$ where feature selection in the proximity space is performed using the CFS method (the first bar). For comparison we also report the accuracy of the $k_{PD_{tuple,\boldsymbol{A}}}$ on the full set of prototypes (the second bar). 266
B.26 Accuracy of $k_{PD_{tuple,\boldsymbol{A}}}$ (using SVM) for different set distance measures. The results are obtained using $\text{MCML}_{\text{diag}}$ where feature selection in the proximity space is performed using the CFS method (the first bar). For comparison we also report the accuracy of the $k_{PD_{tuple,\boldsymbol{A}}}$ on the full set of prototypes (the second bar). 267
B.27 Number of selected features (prototypes) by the CFS algorithm (using $\text{MCML}_{\text{diag}}$) for the FR, MM and MR datasets and for different set distance measures. The first bar denotes the number of instances in the training set. 268
B.28 Number of selected features (prototypes) by the CFS algorithm (using NCA_{diag}) for different set distance measures. The first bar denotes the number of instances in the training set. 269

List of Algorithms

2.1 Retrieving a tree description of a relational instance. 23
2.2 Operator on relational instances. 34
2.3 Operator on tuples. 35
2.4 Operator on sets. 36
2.5 Operator on lists. 36
2.6 Retrieving a flat description of a relational instance. 42

Résumé

L'objet de cette dissertation est d'examiner divers aspects des distances et des méthodes à base de noyaux dans le cadre de l'apprentissage relationnel. Nous commençons par introduire un formalisme pour la représentation multi-relationnelle des données qui repose sur les concepts de tuples, d'ensembles et de listes que nous avons implémenté dans le langage de l'algèbre relationnel. Par la combinaison de ces types de données, nous sommes en mesure de modéliser une grande diversité d'objets composites tels que des arbres et des graphes.

Nous continuons avec la définition de plusieurs opérateurs de distances et de noyaux sur ce formalisme de représentation. Nous ne sommes pas limités à un opérateur spécifique, mais au contraire, nous sommes libres de choisir l'opérateur à appliquer à un type particulier parmi l'ensemble des opérateurs disponibles pour ce type. L'opérateur final applicable à l'objet composite est donné par une combinaison récursive des opérateurs affectés aux sous-structures qui le composent. Nous nous focalisons sur des opérateurs de projections définies sur des ensembles.

Ensuite, nous proposons trois nouvelles familles de noyaux flexibles sur des ensembles où la similitude est basée sur des projections entre les éléments des deux ensembles. Ces familles diffèrent de la plupart des noyaux existants qui moyennent la similarité entre tous les éléments des deux ensembles.

Enfin, nous proposons un framework pour sélectionner adaptativement les représentations des données complexes et/ou des opérateurs sur les représentations. Plus précisément, notre framework comprends un ensemble d'opérateurs et de représentations prédéfinies qui sont combinées de manière optimal. Nous nous focalisons seulement sur un paradigme à base de distances et nous exploitons des travaux antécédents sur l'apprentissage de métriques dans les données vectorielles. Nous utilisons la combinaison optimale de différents graphes de décompositions en sous-structures d'un type spécifique pour définir des noyaux adaptatifs sur des graphes qui tentent de résoudre les limitations actuels des noyaux sur les structures complexes.

Nous entreprenons une comparaison en profondeur de notre système relationnel à base de distances et de noyaux qui comprend: une évaluation empirique de plusieurs distances composites, en insistant sur la comparaison de distances entre ensembles à base de projections; une évaluation empirique de différents noyaux complexes, en insistant sur la comparaison des noyaux sur ensemble à base de moyenne, et des noyaux sur ensembles à base de diverse projections;

une évaluation empirique de notre framework adaptatif pour la combinaison de distances sur ensembles, et la combinaison de diverse décomposition de graphes en sous-structures de divers types.

L'évaluation empirique du système montre que les paradigmes proposés à base de distances et de noyaux sont efficaces sur plusieurs jeux de données d'essai. De plus, pour tous les problèmes relationnels examinés nous parvenons à des résultats conforment à l'état de l'art, et supérieurs aux meilleurs résultats obtenus avec d'autres systèmes relationnels.

Abstract

The goal of this dissertation is to examine various aspects of the distance- and kernel-based learning paradigms applied in relational settings. We start by introducing a multi-relational representation formalism, at the core of which lie the concepts of tuples, sets and lists, which we implemented over the relational algebra language. By combining these data types we are able to model a variety of composite objects, such as trees and graphs.

We proceed with the definition of various distance and kernel operators over the representation formalism. We are not constrained to a specific operator, instead, we are free to assign an operator, selected from a set of available operators, to a particular data type. The final operator over the composite objects is given as a recursive combination of operators assigned to the sub-structures which constitute the objects. We focus on mapping-based operators defined over sets.

Next, we propose three new and flexible families of set kernels where the overall similarity is based on mappings between the elements of the two sets. These kernels differ from most of the existing set kernels which are based on averaging of the similarities of *all* the elements of the two sets.

Finally, we propose a general framework for adaptively selecting representations of complex data and/or operators over representations. More precisely, our framework assumes a set of predefined representations and operators which are then combined in an optimal way. We focus only on the distance-based paradigm and we exploit previous work on metric learning over vectorial data. We use the optimal combination of different graph decompositions into substructures of specific types to define adaptive graph kernels which address the limitations of the existing kernels over these complex structures.

We undertook extensive comparisons of our distance- and kernel-based relational system, which included: an empirical evaluation of various composite distances, with the focus on comparison of set distances based on mappings; an empirical evaluation of different complex kernels, with the focus on comparison of set kernels based on averaging and set kernels based on various mappings; an empirical evaluation of our adaptive framework for the tasks of combination of set distances, and combination of various graph decompositions into substructures of various types.

The empirical evaluation of the system has shown that the proposed distance- and kernel-based paradigms are effective over a number of relational benchmark

datasets. Additionally, in all of the examined relational problems we achieved state-of-the-art results which are better than the best results obtained using other relational systems.

Chapter 1

Introduction

The traditional and widely-used approach in typical data mining and machine learning methods is based on representing learning instances in a vectorial format. This representation allows for the construction of efficient learning systems and has the advantage of simplicity. However, it also restricts the applicability of the resulting algorithms since individuals must be represented as fixed-length tuples of constants. This feature might be inadequate in domains where the intrinsic structure of learning objects is more complex. Examples of applications requiring a richer representation widely occur in various practical fields including computational biology, chemoinformatics, natural language processing, computer vision and networking applications (Džeroski and Lavrac, 2001; Getoor and Taskar, 2007).

As a result of the above fact, the field of relational data mining has developed and flourished over the last fifteen years (Džeroski and Lavrac, 2001; Getoor and Taskar, 2007). One of its most prominent representatives is the subfield of Inductive Logic Programming (ILP) which is usually described as the intersection of machine learning and logic programming. Within the ILP paradigm learning examples are described using first-order logic concepts (e.g. clauses and terms) or subsets of first-order logic (Džeroski and Lavrac, 2001). Recently, a higher order logic was also used as a representation formalism resulting in even more powerful systems (Lloyd, 2003; Gärtner et al., 2004). The main advantage of this formalism is that it allows for the natural representation of sets and multisets, a main difference from first-order terms. Moreover, the higher-order logic usually considers typed syntax which is important for pruning search spaces and simplifies the process of modeling of semantic of the data (Lloyd, 2003).

While the logic-based formalisms are widely used to represented structured data, the scope has now extended and includes other knowledge representation languages. Although the new formalisms are usually not as expressive as the representations based on logic programs, the full power of the latter is hardly ever needed. In particular, in the last few years the focus in the data mining and machine learning communities was on topological structures such as graphs (Washio and Motoda, 2003), or special kinds of graphs like sequences (Durbin et al.,

1999; Leslie et al., 2003), trees (Culotta and Sorensen, 2004; Bille, 2005) and sets (Eiter and Mannila, 1997; Grauman and Darrell, 2007). Both graph-based and logic-based representations are strongly related in the sense that they allow for modeling of complex data. However, the mining of graph data is more concerned with analyzing topological structures embedded in graph data and hence is more geometry oriented. On the other hand, relational data mining is more powerful and aims to find patterns expressed in some logic language, and hence is more relation (or logic) oriented (Washio and Motoda, 2003).

Several learning paradigms were proposed over the logic- and graph-based representations (Džeroski and Lavrac, 2001; Getoor and Taskar, 2007; Gärtner et al., 2004; Washio and Motoda, 2003). Two prominent and widely used families of algorithms exploited in this context are the distance- and kernel-based algorithms.

Distance-based learning is one of the oldest, yet surprisingly effective paradigms in the field of data mining and machine learning (Duda and Hart, 1973; Aha et al., 1991) and has been used in various learning tasks such as classification, clustering and regression. In classification it is known under various names such as k-Nearest Neighbor classification, instance-based learning or lazy learning (Duda et al., 2001; Aha et al., 1991; Aha, 1997). In clustering it is probably the most widely used approach, exploited in methods such as k-means clustering, hierarchical clustering or self organized maps (Duda et al., 2001; Hastie et al., 2001). In regression it has been used to perform local regression where regression models are fitted locally within neighbors of the learning examples (Hastie et al., 2001). More recently, distance-based algorithms became popular in semi-supervised learning (Chapelle et al., 2006) and for (non-linear) dimensionality reduction (Saul et al., 2006). Finally, in the last years there is a growing interest in adaptive approaches for distance measure learning in either fully supervised settings (Goldberger et al., 2005; Globerson and Roweis, 2006) or using side-information (Xing et al., 2003).

Kernel-based methods are a relatively new development within the machine learning and data mining communities (Shawe-Taylor and Cristianini, 2004; Schölkopf and Smola, 2001; Cristianini and Shawe-Taylor, 2000), nevertheless, because of their simplicity and versatility they quickly become a first-choice tool in diverse areas such as classification, regression, clustering, novelty detection and dimensionality reduction (see e.g. Shawe-Taylor and Cristianini, 2004). One of the main advantages of kernel-based methods is that they combine the flexibility of non-linear algorithms with the efficiency and simplicity of linear methods. The non-linearity is achieved by the application of a positive semi-definite kernel function that enables the data to be non-linearly embedded in some inner-product (or pre-Hilbert) feature space without the explicit computation of the feature map. As a result, any linear algorithm which is based on inner products can be implicitly applied in this feature space and hence become non-linear. The foundation of kernel-based methods in Statistical Learning Theory (Vapnik, 2000) made it possible to apply these methods in cases where the feature space is of high (or even infinite) dimensionality, avoiding over-fitting and without being affected by the "course of dimensionality" (Hastie et al., 2001).

Finally, the learning part in kernel methods usually boils down to a convex optimization problem which has an unique solution and is amenable to efficient optimization techniques.

The distance- and kernel-based approaches are collectively characterized by the fact that they do not require a direct access to the training examples, instead they access the data only by a distance and a kernel function, respectively. As a result, the above two paradigms are easily extended to support input spaces whose representation is more general than attribute-value. This is achieved by defining data specific distance or kernel functions providing the interface with the composite data and incorporating domain knowledge, if such exists.

Most of the work in the distance-based relational learning falls within the ILP paradigm in which the representational language used is most often first-order logic or some subset of it (Horváth et al., 2001; Ramon and Bruynooghe, 1998). In the last few years the focus has turned to special types of complex objects, namely general graphs (Washio and Motoda, 2003), or specific types of graphs such as sequences (Durbin et al., 1999), trees (Bille, 2005) and sets (Eiter and Mannila, 1997; Tatti, 2007; Woźnica et al., 2006a). Ramon (2002) provides an overview of different types of structured problems tackled in distance-based relational learning.

Unlike the distance-based relational learning community, the kernel-based research community has largely ignored the issue of data and problem representation, with the exception of Gärtner et al. (2004), who have introduced a relational learning framework for general structured data based on a typed formalism of a higher-order logic. Other examples of kernels defined over specific types of complex objects include kernels over sets (Kondor and Jebara, 2003; Woźnica et al., 2006a), sequences (Leslie et al., 2003), trees (Collins and Duffy, 2002), and labeled graphs (Gärtner et al., 2003; Kashima et al., 2003; Ramon and Gärtner, 2003; Menchetti et al., 2005). Most of these kernels boil down to the general $\Re$-Convolution kernel proposed by Haussler (1999). Gärtner (2003) provides an overview of different kernels defined on structured learning problems.

The characteristically modular design of the distance- and kernel-based methods makes them amenable to theoretical analysis but also simplifies the computer implementation process (Cristianini and Shawe-Taylor, 2003). More precisely, a general purpose learning module implementing the task of, e.g. classification or clustering, can be easily combined with a data specific distance or kernel modules. The result is that these methods can be used for dealing with "non-standard" learning problems such as classification on graphs, clustering of sets, etc.

1.1 Limitations of the Existing Approaches

In this section we describe some of the limitations of the existing approaches for learning over structured data. We will describe the general problems common to various relational methods as well as problems specific to distance- and kernel-

based learning over complex objects.

In order to design algorithms over composite objects one needs a knowledge representation formalism that is able to accurately and straightforwardly reflect the underlying semantics of the data. In order for the representation language to become acceptable by a wide audience it should have several properties. First, the formalism should naturally extend the propositional representation based on a single table, making it easier to design relational algorithms which naturally encompass existing algorithms operating on a single table. Second, the representation language should be ideally defined in a modular fashion, simplifying the process of data modeling and the design of various data mining operators applied over this representation. Third, a desired property of such formalism is that it is *typed*; as pointed out by Flach et al. (1998), a typed approach usually makes the modeling process easier. Finally, within this formalism it should be easy to store (and possibly index) the complex objects in such a way that they can be efficiently accessed. Most of the existing logic- or topological-based approaches have difficulties reaching most the above goals. The exception is the higher-order logic formalism of Lloyd (2003) which is based on typed λ-calculus; however, because of its flexibility this formalism is not easy to understand and renders the modeling process a non trivial task.

The other problem, which is strongly associated with the lack of modularity of the underlying representation languages, is that most of the existing relational systems can be characterized as monolithic in the sense that they rely on a single type of data mining operator, defined on a specific complex data type. The result is a large number of relational data mining algorithms and systems which have limited applicability to problems that involve only very specific complex data types, most often only a single complex data type, and even more limited, if inexistent, flexibility on the data mining operators that they employ. Džeroski (2007) has stressed the limitations of the existing systems and has emphasized the need for a general framework for data mining which, among others, should allow for the easy definition of new complex data types and data mining operators over these data types.

The above limitations are in particular problematic in the relational distance- and kernel-based paradigms, where most of the existing systems are constrained to a single monolithic type of complex distance or kernel, respectively. It is obvious that there is no single distance (kernel) that is overall better than any other for all types of problems. This has been observed in various empirical comparisons (Aha et al., 1991; Kalousis et al., 2005) as well as confirmed by various "no free lunch theorems" (Schaffer, 1994; Wolpert, 2001). Typically, a practitioner should consider different distances (kernels) to find the one that best matches the problem requirements. To make things even more complicated, it can be that different constituent sub-objects of the same type call for different operators. Thus, a desired property of the final complex distance (kernel) is that it is derived by a combination of heterogeneous distances (kernels). Only a few of the existing relational distance- or kernel-based systems offer this type of flexibility.

The other limitation specific to relational kernel-based learning is that many

of the kernels defined over structured data like sequences, sets and graphs are based on a (sometimes implicit) decomposition of complex structures into substructures of different types. The final kernel is defined as the *Cross Product Kernel* between the corresponding multi-sets of decompositions. For particular decompositions $\mathcal{O}_1^t$ and $\mathcal{O}_2^t$, into substructures of type t, of the composite objects O_1 and O_2, the above kernel can be written as

$$k_{set}(O_1, O_2) = \sum_{(o_1,o_2)\in\mathcal{O}_1^t\times\mathcal{O}_2^t} k(o_1, o_2) \tag{1.1}$$

where k is a kernel over specific types of substructures in $\mathcal{O}^t$, and the summation over the elements of the multi-sets takes into account their multiplicity.

One problem with the kernel given in Equation 1.1 is that it computes the average of similarities given by the sub-kernels applied on *every* pair of elements from the two decompositions. This feature might be inappropriate in cases where only specific elements of decompositions are important; in such cases the kernel based on *specific* elements is expected to perform better. An example of such application is multiple-instance learning where the task is to learn a concept given positive and negative sets of instances (Gärtner et al., 2002). In this setting a set is labeled negative if all the instances are negative and is labeled positive if *at least one* of the instances is positive. The fact that all the elements of the sets are matched, might also be problematic in kernels for graphs where the cross product kernel is their integral part and is applied on the multisets of decompositions of graphs into their parts (walks, paths, trees, etc.). This might adversely affect the generalization of a large margin classifier (e.g. SVM) since due to the combinatorial growth of the number of distinct subgraphs most of the features in the feature space will be poorly correlated with the target variable (Ben-David et al., 2002; Menchetti et al., 2005).

Different solutions to tackle the above problem have been proposed in the literature. For kernels over graphs the possible solutions include down-weighting the contribution of larger subgraphs (Collins and Duffy, 2002; Gärtner et al., 2003), using prior knowledge to guide the selection of relevant parts (Cumby and Roth, 2003) or considering contextual information for limited-size subgraphs (Menchetti et al., 2005). Yet another solution was proposed in (Fröhlich et al., 2005) in which only specific elements of the corresponding multi-sets are matched in such a manner that the sum of similarities of the matched elements is maximum. The underlying idea in this kernel is that the actual matching will focus on the most important structural elements, neglecting the substructures which are likely to introduce noise to the representation. Similar kernels were considered in the computer vision community (Wallraven et al., 2003; Lyu, 2005b; Boughorbel et al., 2004), where explicit correspondences between two sets' features of images are constructed.

The above idea based on explicit correspondences between elements of the two sets can be also generalized by changing the right side of Equation 1.1 such that the sum runs over specific elements of the corresponding multi-sets of decompositions excluding elements which are likely to be irrelevant for a given

target variable. This kernel will be based on specific pairs of elements from the two sets and can be written in a general form as

$$k_{set}(O_1, O_2) = \sum_{(o_1,o_2)\in F} k(o_1, o_2),\ F \in \mathcal{F} \tag{1.2}$$

i.e. it is a sum of pairwise elementary kernels, $k(o_1, o_2)$ over specific pairs which are defined by the mapping $F \subseteq \mathcal{O}_1^t \times \mathcal{O}_2^t$ which belongs to a mapping family $\mathcal{F}$. This kernel naturally generalizes the cross product kernel and kernels from (Fröhlich et al., 2005; Wallraven et al., 2003; Lyu, 2005b; Boughorbel et al., 2004). Moreover, its semantics are similar to the semantics of set distance measures based on mappings. For example, in the kernel of Fröhlich et al. (2005) the mapping family $\mathcal{F}$ contains all the bijections of the larger set to the smaller. The idea of using specific pairs of points in a set kernel is promising, however, it is easy to show that the kernel from Equation 1.2 is not positive semi-definite for a general $\mathcal{F}$. Moreover, some of the existing kernels of the form as in Equation 1.2, which were "believed" to be valid, were recently shown to be non-PSD in general (Vert, 2008; Lyu, 2005b). The other limitation of the above kernels is that the appropriate selection of the specific pairs of elements is application dependent and ideally should be guided by domain knowledge, if such exists. As a result there is a need for other, more flexible set kernels which are not necessarily based on averaging and are positive semi-definite.

Finally, one of the main challenges in applications involving complex objects is that of the proper representation of the learning instances (within a given representation formalism). The choice of the correct representation is crucial for the successful application of machine learning techniques since it renders the actual problem easier (if not trivial) to solve. Within the logic- and graph-based representations the complex objects can be represented in different manners, modeling for different semantics and aspects of the problem. For example graphs, in the context of kernel-based algorithms, can be represented among others as sets of walks of different lengths, trees or more general subgraphs (Ramon and Gärtner, 2003), but in practice it is difficult to specify in advance the appropriate type of substructures for a problem at hand. A common intuition is that by decomposing into more complex subgraphs the expressivity, and consequently the performance, of resulting kernels increase. This is, however, in contrast with some experimental evidence (Menchetti et al., 2005) which show that decompositions into rather simple substructures perform remarkably well with respect to more complex decompositions on a number of different datasets. Although it is in principle possible to simultaneously exploit kernels defined over different representations, this is usually not done because there is a trade-off between expressivity reached by enlarging the kernel-induced feature space and the increased noise to signal ratio (introduced by irrelevant features). In practice a single type of decomposition is used which is then, most often, employed in the context of the cross product kernel of Equation 1.1.

Strongly associated with the problem of selection of the appropriate representation, is that of selection of an appropriate distance or kernel function on

the selected representation. It is possible to have different distance (or kernel) functions for a given representation, where again each distance (kernel) has different semantics. Using again the example of graphs: if these are represented as sets of objects then we can choose among different distance measures or kernels defined over sets.

Adapting (or learning) a distance measure (kernel) for a given problem is a difficult task and in general there are only few systematic approaches that address it. In the context of distance-based learning and for propositional data, one family of such methods consists of finding a set of good feature weights in the input space. Several feature weighting algorithms have been proposed over the last decade (Wettschereck et al., 1997), however, they have two main limitations. First, it is difficult to extend these techniques to non-vectorial data. Second, their expressiveness is limited since most of them do not account for feature interactions. A generalization of the feature weighting approach is based on adjusting (or learning) parameters of a (parameterized) distance measure directly from the data. In this context several attempts have been made recently, either in a fully supervised setting (Goldberger et al., 2005; Globerson and Roweis, 2006; Domeniconi and Gunopulos, 2002; Weinberger et al., 2006; Yang et al., 2006) or using side information (Xing et al., 2003; Bar-Hillel et al., 2005; Kwok and Tsang, 2003; Hertz et al., 2004; Schultz and Joachims, 2004). The methods in the latter class consider side information in the form of: (i) relative, qualitative examples (e.g. "A is closer to B than B is to C") (Schultz and Joachims, 2004), (ii) multiple, absolute, qualitative examples (e.g. "A, B and C are similar") (Bar-Hillel et al., 2005) (iii) pairwise, absolute, qualitative examples (e.g. "A and B are similar" or "B and C and not similar") (Xing et al., 2003; Kwok and Tsang, 2003; Hertz et al., 2004). The distance measures in the above methods are usually restricted to the Mahalanobis metric family[1] parameterized by a positive semi-definite matrix. All these methods were developed for vectorial data and are similar in the sense that the actual problem is cast as a mathematical optimization task. However, these algorithms differ with respect to the actual objective function that is being optimized and hence they implicitly assume different distributions of the data.

It should be mentioned that the automatic adaptation of kernels (Lanckriet et al., 2004; Ong et al., 2005; Bousquet and Herrmann, 2003) is more general than metric learning since a valid kernel k can be directly used to compute a pseudo metric in the feature space by $\sqrt{k(x,x) - 2k(x,y) + k(y,y)}$. The proposed methods for kernel combination differ in the objective functions (e.g. cross-validation risk, margin based, Kullback-Leibler divergence, etc.) as well as in the classes of kernels that they consider (e.g. finite or infinite set of kernels, etc.). Nevertheless, the problem with learning kernel combinations is that the combined elements should, obviously, be valid kernels on the different types of decompositions. However, as we mentioned previously this type of kernels are based on the cross product kernel that requires the complete matching of the

[1]The Mahalanobis metric parameterized by a positive semi-definite matrix $\boldsymbol{A}$ is defined as: $d_{\boldsymbol{A}}(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x} - \boldsymbol{y})^T \boldsymbol{A} (\boldsymbol{x} - \boldsymbol{y})$ for $\forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$.

individual components, raising the problems that were described before. The other problem with the existing methods for kernel combination is that they work only in a transductive setting, i.e. completing the labeling of a partially labeled dataset, which in turn limits the application area of such methods.

To summarize, the two important constituents of any distance and kernel-based algorithms are the representation of the learning instances and the operator employed on that representation. Ideally, both of them should be determined on the basis of domain knowledge; however, even in the presence of domain knowledge, it can be far from obvious which complex representation should be used or which operator should be applied on the chosen representation. An obvious question is how to select from the set of plausible couplings of representations and operators the one that best fits the requirements of the problem at hand. A simple solution is to *select* it by cross-validation; however, the main drawback of this approach is that only one representation-operator couple per training set is selected which limits the expressiveness of the resulting method. Additionally, this approach is limited to a small number of representations and operators, due to computational constraints, and requires the use of extra data. Given the above problems a better solution would be to automatically *combine* a number of the representation-operator couples where each couple is assigned a weight, which specifies its importance. Ideally, the establishment of these weights should be a part of the learning process.

1.2 Contributions

The goal of this thesis is to examine various aspects of the distance- and kernel-based learning paradigms applied in relational settings. The representation formalism we used to represent composite objects is based on concepts from relational algebra. Over this representation we proposed various distance and kernel operators which are defined as a recursive combination of operators associated with the sub-structures which constitute the learning instances. We also extended the flexibility of the existing kernels over sets and proposed three new and flexible families of set kernels where the overall similarity is based only on specific elements of the two sets. Finally, we proposed a general framework for combining representations of complex data and/or operators over a given representation. We exploited this adaptive framework to define a flexible and powerful class of graph kernels. Finally, we undertook extensive and in-depth analysis of our distance- and kernel-based relational system.

In what follows, we provide a short summary list of the main contributions of this work.

1. Representation Language (**Chapter 2**)

   (a) Definition of a multi-relational representation formalism, at the core of which are three data types, i.e. tuples, sets and lists. Based on these data types one can directly model a variety of complex structures such as trees, graphs or more general structures that do not fall

to a specific topological category. The representation language is implemented over the relational algebra language where the description of an object is spread across different interconnected tables which constitute a relational database.

(b) Definition of a general class of data mining / machine learning binary operators over the proposed representation. The operators are given as a recursive combination of operators assigned to the substructures which constitute the learning instances. Different operators can be used on the different building blocks of the relational instances, and it is up to the practitioner to declare the particular operators' assignment.

2. Distances (**Chapter 3**)

   (a) Definition and theoretical analysis of various distance operators over composite objects represented in the relational formalism, with the focus on set distances based on mappings between sets. We start with distances on simple domains and we gradually build more complex distances on more complex domains, using the simpler distances as building blocks.

   (b) An in-depth analysis of the performance of composite distances on a number of relational benchmark datasets.

3. Kernels (**Chapter 4**)

   (a) Definition and theoretical analysis of various kernel operators over composite objects represented in our formalism. Similar to distances, kernels on simple objects form a basis for definitions of more complex kernels.

   (b) Definition of three families of set kernels which are not based on averaging; instead they take into account only specific pairs of elements from the two sets. The considered kernels are kernels in proximity space induced by set distances, set distance substitution kernels and kernels directly based on specific pairs of elements of the form of Equation 1.2. The semantics of mappings between specific sets' elements is similar to the one used in set distances.

   (c) Rigorous empirical evaluation of the proposed kernels on a number of relational problems.

4. Adaptive Approaches (**Chapter 5**)

   (a) Definition of a general framework for learning a "good" combination of different representations of complex data and/or a "good" combination of complex distances over a given representation. We focus only on the distance-based paradigm and we exploit the previous work on metric learning over vectorial data. The learning problem is defined as a mathematical optimization task. Additionally, we address the problem of regularization of the learned combinations.

(b) Evaluation of the proposed framework for the task of combination of set distances.

(c) Evaluation of the proposed framework for the task of combination of graph decompositions into subgraphs of various types. This evaluation is performed in the context of graph kernels which are based on combinations of different representations. These kernels form a powerful and flexible class of graph kernels which address some of the limitations of the existing kernels over these complex objects.

1.3 Reading Guide

To help those "impatient" readers who want to quickly grasp the main contributions of this thesis, we advice, after reading the present chapter, to move directly to the first part of Chapter 6 (without Section 6.1) where we give an overview of the main findings and observations.

We also provide a more detailed reading guide through all Chapters 1 to 6, pointing to sections which should not be omitted at a fist reading. First, for an overview of the limitations of the current approaches an informal account of the material that follows, the reader is advised to read all of the present chapter. Next, in Chapter 2 it is recommended to read Section 2.1 and most of Section 2.3 where we informally introduce our representation language and describe how to model graphs, respectively. The remaining sections (2.2 and 2.4) can be skipped during the first reading; they are intended for readers interested in practical implementation of our formalism. In Chapter 3 the reader should not omit Sections 3.2.2, 3.2.3 and 3.3 where we describe distances on tuples, sets and trees; these concepts will be widely used in the remaining part of this study. The most important experimental results from this chapter are presented in Sections 3.5.4 and 3.5.7. Our main contributions from Chapter 4 are the theoretical presentation of the set kernels based on mappings (Section 4.2.4), kernels on trees (Section 4.3), and the experiments with set kernels based on mappings (Section 4.4.4). The reader is expected to study all of Chapter 5 as it presents novel techniques for adaptive learning over composite representations. Finally, it is important to read the first part of Chapter 6 (without Section 6.1) for an overview of the main contributions of this work. The reading guide to this thesis, with the material which should not be omitted, is schematically presented in Figure 1.1.

Finally, we mention that yet another way to read this dissertation is to start with the present chapter, then move to the first part of Chapter 6 for the summary of the main contributions, and the fill in the details by reading Chapters 2 to 5.

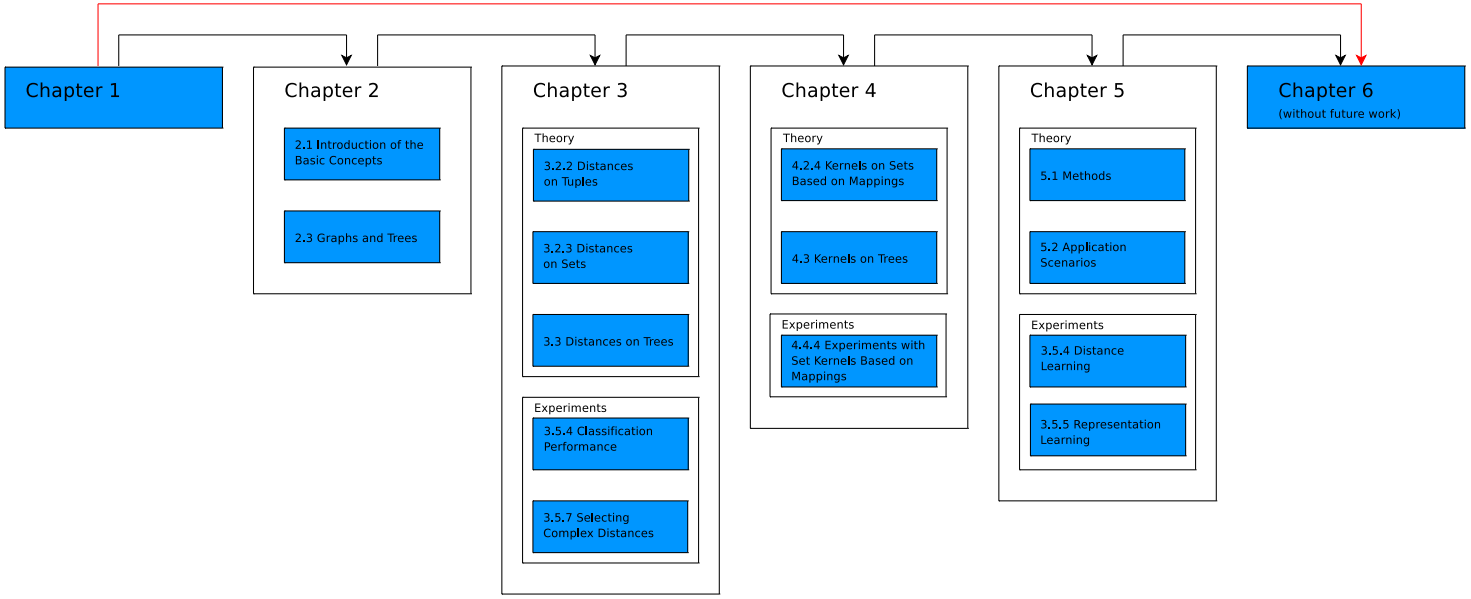


Figure 1.1: A reading guide to the thesis. The alternative way of reading this thesis, represented by the red line, is intended for "impatient" readers.

1.4 Bibliographical Note

The content of this dissertation is based on the existing publications. In particular some parts of Chapter 2 are based on (Woźnica et al., 2005a; Kalousis et al., 2005; Woźnica et al., 2005b, 2006b) while the material from Chapter 3 is based on (Kalousis et al., 2005; Woźnica et al., 2005b, 2006a). Our work presented in (Woźnica et al., 2005a, 2004, 2006c) has laid the groundwork for the material in Chapter 4. Finally, the content of Chapter 5 is based on (Woźnica et al., 2007, 2006b,d, 2008).

Chapter 2

Representation Language

The traditional and still widely used approach to representing complex learning instances[1] is based on first- or higher-order logic. Recently, also the representations based on graphs, or special types of graphs, have become very popular. In this study we adopt a different representational approach and define learning instances using solely concepts from *relational algebra*.

Relational databases are probably the most common way of storing structured information nowadays. In the relational representation the description of an object is spread across a number of interconnected tables which constitute a relational database. There are several advantages of representing composite objects using this formalism. First, relational algebra provides a robust data modeling tool with well understood semantics by a large audience. Second, learning algorithms directly operating over this representation can readily tackle any kind of relational problem in which training data is stored within a typical relational database, with no need for a change of representation. Third, the relational formalism naturally extends the typical attribute-value representation, and hence it is in principle easier to design a relational learning algorithm, naturally encompassing an existing algorithm operating over a single table. Fourth, a computer implementation of a learning system over complex objects is simplified, since the existing relational database systems can be exploited to store learning instances. Finally, relational databases provide the functionality of indexing the instances so that they can be accessed in an efficient way.

Before going to the detailed description of the relational algebra formalism it is useful to first provide a high-level informal and intuitive presentation of the underlying concepts. More precisely, we will consider learning instances represented in this formalism as composite objects which can be decomposed into simpler sub-parts of various types. These sub-objects are further decomposed until non-decomposable (i.e. primitive) objects such as numbers or nominal attributes are encountered. The non-primitive building blocks are tuples, sets and lists. Tuples allow to represent fixed-length collections of objects of possibly

[1] In this work we will use terms *complex*, *composite* and *structured* instances (or objects, examples, etc.) interchangeably.

different types, while sets (lists) are used to represent unordered (ordered) collection of objects, each of which is of the same type. Based on these building blocks one can directly represent a variety of composite structures such as trees or graphs, the latter through various approximations.

The main reason why we consider learning instances in this modular way is that it simplifies the process of data modeling and the design of the data mining operators. The data mining operators we consider are defined in a declarative manner where the practitioner first defines the different data types which constitute the learning examples, and then declares which operator should be used for each data type. In general there is a number of different data mining operators associated with different data types, each one with different semantics. In particular, it is possible to define different operators even for objects of the same general type e.g. sets of lists, and sets of tuples, etc. The final data mining operator over the full complex learning instances is then automatically composed by simply combining, in a recursive manner, the operators assigned to the sub-structures which constitute the learning instances. Obviously, there are as many different instantiations of the final data mining operator as there are combinations of data mining operators over the components of the full complex learning instances.

The remaining part of this chapter is organized as follows. In Section 2.1 we give an informal introduction to the representation language by defining various building blocks of learning instances; we consider primitive attributes, tuples, sets and lists. In Section 2.1.5 we show how the above building blocks can be combined so that we obtain general composite objects in the form of trees. In Section 2.2, we give a description of relational algebra and provide a link with the high-level concepts. In Section 2.2.3 we illustrate the main ideas of our approach with an example from the field of proteomics. In Section 2.3 we demonstrate how the proposed relational representation formalism can by exploited to model (labeled) graphs. In Section 2.4 we define a general class of recursive machine learning / data mining operators applied on our representation language. In Section 2.5 we give an overview of the related work and place our framework within that context. Finally, we conclude with Section 2.6.

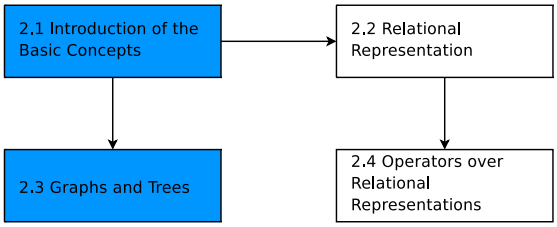


Figure 2.1: Schematic structure of Chapter 2. The recommended sections to read are highlighted.

The reader is recommended to read at least Section 2.1 where we informally

describe concepts of our representation language, and most of Section 2.3 where we discuss how to model trees and graphs. Sections 2.2 and 2.4 can be omitted at a first reading; they are mainly intended for readers interested in practical implementation of our formalism. The organization of this chapter is schematically presented in Figure 2.1.

2.1 Introduction of the Basic Concepts

As already argued, it is of crucial importance to be able to deal with structured data; handling complex data is attracting an increasing amount of attention within the data mining and machine learning communities. In the remaining part of this section we will present the basic concepts of our relational representation language in an informal way, trying to keep it as intuitive as possible.

A structured individual datum is an object that has its own structure, i.e. it consists of values for several attributes which may be of different types and may take values in different ranges (Džeroski, 2007). We require that all data items – learning instances – provided to a learning algorithm are of the same type and share the same structure. We assume a set of elementary data types, such as *symbolic* or *numeric*. Other elementary data types might include $symbolic(S)$, where S is a finite set of identifiers, or *integer*. These are the most basic building blocks characterized by the fact that they are not decomposable into simpler sub-parts. In addition, we are given a number of type constructors, such as *tuple*, *set* and *list*, that can be used to construct more complex data types from the existing ones. For example, $tuple(boolean, number)$ denotes a data type where each datum consists of a pair of a boolean value and a real number, while $list(tuple(boolean, number))$ denotes a data type where each datum is a list of such pairs. Different data types can be combined such that we can define e.g. tuples where nodes are objects of type *set*, or sets where elements are lists, etc.

To summarize, in modeling complex data the practitioner has the freedom to combine different data types such as elementary attributes, tuples, sets and lists. In general, the process of data modeling should be guided by the available background knowledge and application requirements. Moreover, computational constraints should be taken into account since in general more complex representations impose a higher computational burden on the corresponding operators defined over the selected representations. It should be mentioned that the background knowledge-driven data modeling might be problematic in practice since we rarely have a solid description of the learning problem. This issue will be tackled in Chapter 5 where we will show how the proper representation of the data can be automatically adapted to a problem at hand.

In the remaining part of this section we will present in more detail the primitive (*symbolic* and *numeric*) as well as composite (*tuples*, *sets* and *lists*) data types that will be used in our study. Then we will show how these data types can be combined to define general composite objects.

2.1.1 Primitive Attributes

We start our presentation by defining the most basic objects, which we call *primitive objects*, sometimes also referred to as *elementary features* or *primitive attributes*. These objects are collectively characterized by the fact that they are not decomposable into simpler sub-parts.

In this study we focus only on the *symbolic* (or *nominal*) and *numeric* (or *quantitative*) features. The type of the former is denoted as $symbolic(S)$, where S is a finite and discrete set of identifiers. Symbolic features represent a finite set of possible values, symbols or modalities. Moreover, they can be counted, but not ordered. For example, in order to create a symbolic object representing an atom we could set $S = \{H, C, CL, O, \ldots\}$. The other type of primitive objects are numerical attributes whose type is denoted by $numeric(a, b)$. This attribute takes values on an interval $[a, b] \subset \mathbb{R}$. Here we do not make a distinction between real numbers and subsets thereof (e.g. integers). This means that in general the elements of these attributes are uncountable, but ordered.

2.1.2 Tuples

A *tuple* is a finite and fixed-length array of objects, where each object has a specific data type; different objects might have different data types. Tuples are also sometimes referred to as *records* (Stonebraker, 1996). In the rest of this study we will sometimes make a distinction between tuples and *vectors*. By the latter we mean tuples whose elements are the basic (i.e. numerical and symbolic) data types[2]. Vectors are probably the simplest composite objects and are widely used to represent data in the "traditional" areas of machine learning and data mining (Duda et al., 2001; Hastie et al., 2001) as well as in multivariate statistics (Mardia et al., 1979; Wasserman, 2004).

More formally, consider data types T_i for $i = 1, \ldots, n$. In general $T_i \neq T_j$ for $i \neq j$, and T_i are not necessarily primitive data types. We also assume that there is a type constructor *tuple* used to provide the tuple data type. Thus $tuple(T_1, \ldots, T_n)$ denotes the data type of tuple whose elements are of data types T_i. Let $\mathcal{X}_i$ be a set of objects of data type T_i. Then a tuple consisting of objects $o_1, \ldots, o_n$, where $o_i \in \mathcal{X}_i$, is denoted by $o = (o_1, \ldots, o_n)^T \in \boldsymbol{\mathcal{X}}$, where $\boldsymbol{\mathcal{X}} = \mathcal{X}_i \times \ldots \times \mathcal{X}_n$. In particular, for objects $o_1, \ldots, o_n$ of type $numeric(\mathbb{R})$, $o = (o_1, \ldots, o_n)^T \in \mathbb{R}^n$ is a standard vector in the Euclidean space $\mathbb{R}^n$. For a tuple o, $o(i)$ denotes its i-th element.

2.1.3 Sets

The other important data type we consider in this work are finite, unordered *sets* and more generally *multi-sets*, i.e. sets which can contain the same element

[2]Conventionally, vectors were limited to contain only numerical values, however, symbolic attributes can be easily converted to orthogonal vectors with 0 and 1 as elements. In any case, this conversion does not influence the various operators over tuples, which will be defined in the subsequent chapters.

several times. We only focus on (multi-)sets containing elements which are of the same type.

Application domains, where it is most natural to represent each training example as a (multi-)set of objects, include chemoinformatics where chemical molecules may be described by a set of possible decompositions into different paths (Ramon and Gärtner, 2003). Similarly, in computer vision, images may be represented as sets of pixels encoding the corresponding coordinates, intensities and colors (Jebara and Kondor, 2003; Kondor and Jebara, 2003); in bioinformatics, and more specifically in proteomics, a mass-spectrum could be described as a set of peaks of the form *(mass,intensity)* (Kalousis et al., 2005); in natural language processing the most widely used representation is based on bags (multi-sets) of constituent words (Joachims, 2002).

More formally, let T be a data type. Moreover, we assume that there exists a type constructor *set* used to provide the set data types and $set(T)$ denotes the set data type whose elements are of data type T. Let also $\mathcal{X}$ be a set of objects of data type T. We denote a set consisting of (not necessarily different) objects $o_1, \ldots, o_n$, where $o_i \in \mathcal{X}$, by $\{o_1, \ldots, o_n\}$.

2.1.4 Lists

Lists are important and widely used composite objects. Similarly to sets we assume that the elements of lists are of the same type. Lists are used in cases where it is important to account for the *order* of elements in a given collection of objects. In fact, this is the main difference between lists and sets where the order is not given. In practice the elements of lists are assigned index values $i \in \mathbb{N}$ specifying the position within the collection.

Consider a data type T and assume that there is a type constructor *list* used to provide the list data type. Then $list(T)$ is the data type of lists whose elements are of data type T. Let $\mathcal{X}$ be a set of objects of data type T. We will denote a finite list ℓ of objects $\ell_1, \ldots, \ell_n$ ($\ell_i \in \mathcal{X}$) as $\ell = [\ell_1, \ldots, \ell_n] \in \mathcal{X}^n$.

Some definitions and notations will be useful in the remainder of this work. Sequences are lists which consist of elements of primitive data types. $|\ell|$ is the *length* of ℓ. We denote by $\boldsymbol{i}$ a sequence $1 \leqslant i_1 < i_2 < \ldots < i_n \leqslant |\ell|$ of indices; we say that $i \in \boldsymbol{i}$ if i is one of the sequence indices. We denote with $l(\boldsymbol{i})$ the length of $\boldsymbol{i}$. For list ℓ, $\ell[k]$ is its k^{th} element. Finally, we denote the set of all lists of objects from $\mathcal{X}$ by $\mathcal{L}(\mathcal{X})$.

2.1.5 General Composite Objects

In this section we will show how the different objects defined so far can be exploited to represent general composite instances. Informally, the general complex objects are recursively defined using the primitive data types as well as composite data types, i.e. tuples, sets and lists. To get the complete description of a complex object, O, we have to traverse its full structure, starting from the top-level "root" element of O (denoted as $root(O)$), and following all associations between the nested sub-parts. Each time composite sub-objects are found,

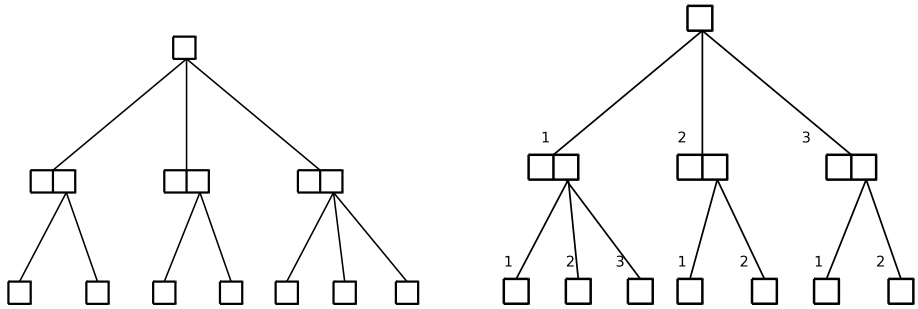


Figure 2.2: Examples of two trees corresponding to complex objects where only sets (a) and only lists (b) are used.

they are further decomposed. This procedure continues until primitive objects are encountered.

The way a complex object, O, is constructed suggests that it can be in fact regarded as a tree[3] whose root corresponds to $root(O)$. The root is connected with the nodes of the second level via a *set* or *list* data type, i.e. the root has as a part of its description a set of objects or list of objects from the second level. In the same way nodes at level d of the tree are also sub-objects related with one of the elements found in nodes at level $d-1$. A node is a leaf if it is of either elementary (i.e. *numeric* or *symbolic*) or of *tuple* type. In case the connections between the nodes are determined only by the application of objects of type *set* we obtain *unordered trees*. On the other hand, if only objects of type *list* are used we obtain *ordered trees*. It should be mentioned that the resulting structures are *labeled trees* (with some labels being possibly empty), where labels are in the form of vectors (or primitive types) and are only assigned to nodes (and not to edges). Figure 2.2 presents two examples of trees generated from complex objects where the links correspond to sets and lists, respectively. In the first plot (a) only sets are used and the tree corresponds to a composite object of type

$$set(tuple(T_{prim}, set(T_{prim})))$$

where T_{prim} is some primitive data type, either *numeric* or *symbolic*. In the second plot (b) we use only objects of type *list* and hence the tree is ordered; this tree corresponds to an object of type

$$list(tuple(T_{prim}, list(T_{prim}))).$$

In general, we can use both sets and lists where children of a given node are divided into non-overlapping groups, so that we obtain *mixed trees*. An example of such tree corresponding to an object of type

$$tuple(set(list(T_{prim})), list(set(T_{prim})))$$

[3]The formal definition of trees will presented in Section 2.3.

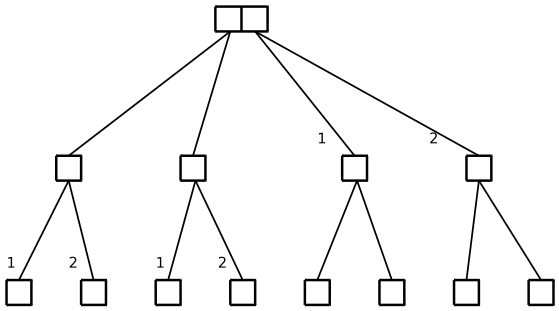


Figure 2.3: An example of a mixed tree.

is given in Figure 2.3.

To summarize, labeled trees can be easily represented using simpler objects. In particular, it means that possible operators on trees could be defined as a combination of operators over simpler structures[4].

2.1.6 Operators

In this section we will briefly describe general data mining operators applied over the proposed representation; data mining operators will be the main focus in the remaining part of this study.

Informally, our operators work in a recursive manner traversing the full tree structures of the input composite objects, as described in Section 2.1.5. During the recursion, the components of the input objects are visited in exactly the same order as when these were constructed. The operators are defined as a recursive combination of operators defined on composite objects' sub-parts, both primitive (i.e. *numeric* and *symbolic*) and composite (*tuples*, *sets* and *lists*). The data mining operators we consider in this study are defined in a declarative manner. i.e. the practitioner specifies which operators should be used for each of the sub-objects. In general, there is a number of different data mining operators, each one with different semantics, and it is up to the analyst to declare which specific operator should be used for a given data type. A more formal presentation of the data mining operators will be given in Section 2.4.

2.2 Relational Representation

In the previous sections we presented the high-level concepts of our representation language. In this section we provide a detailed presentation of relational algebra and make a link with the definitions presented above. We show that the relational paradigm is a natural choice when it comes to modeling composite

[4]Other operators on trees taking into account a more global view are possible. Different distance and kernel global operators will be discussed in Sections 3.6 and 4.5, respectively

objects which are decomposed into primitive attributes, tuples and sets. The "standard" relational formalism has a limitation when it comes to modeling lists, however we propose an extension which allows for representation of these composite structures. We conclude this section by illustrating the modeling process with a simple example from the field of proteomics where we start with a problem specification and model the data using the relational formalism.

It should be mentioned that the relational representation can be seen as a simplified version of the formalism presented in (Lloyd, 2003; Gärtner et al., 2004) which is based on the higher-order logic. However, because of its flexibility the latter representation language is not intuitive and difficult to use in practice. The higher-order formalism will be discussed in some details in Section 2.5.

We also mention that the actual relational learning system we exploited in all our experiments was our multi-relational extension of the WEKA machine learning / data mining toolkit (Witten and Frank, 2005). We decided to exploit this system because of its versatility and the availability of the existing code. Moreover, it was natural to extend it to the multi-relational representation by redefining its input data in the form of a relational database, i.e. it consists of a set of interconnected tables, where each table is stored in an ".arff" file (i.e. the native WEKA format for storing tables). This set of interconnected tables precisely defines a relational schema.

2.2.1 Relational Algebra

The following relational algebra definitions are adapted from (Ullman, 1982). A relation R_i is a set of tuples, more specifically a subset of some Cartesian product $D_1 \times \ldots \times D_{z_i}$ where each D_l is a domain (i.e., a set of objects of primitive types). The relation schema of a relation R_i is the set of all attributes of R_i and we denote it as $R_i(A_1, \ldots, A_{z_i})$. Each attribute A_l has an associated *domain* $dom(A_l) = D_l$. The number of attributes z_i of a relation R_i is called the *arity* of the relation. A *tuple* R_{i_j} of a relation R_i is a particular row in R_i with $R_{i_j} = (v_{j1}, v_{j2}, \ldots, v_{jz_i})$ and v_{jl} the value of the A_l attribute in the R_{i_j} tuple; v_{jl} will be also denoted as $R_{i_j}.A_l$. A relational database schema is a set of relation schemes $\mathcal{R} = \{R_1, \ldots, R_n\}$.

An attribute $A_k \in R_i(A_1, \ldots, A_{z_i})$ is called a *potential key* of relation R_i if it assumes a unique value for each tuple of the relation. An attribute $A_l \in R_j(A_1, \ldots, A_{z_j})$ of relation R_j is a *foreign key* if it references a potential key A_k of relation R_i, in that case we will also call A_k a *referenced key*. A *standard link* is defined on the basis of a foreign key relation and is a quadruple of the form $sl(R_i, A_k, R_j, A_l)$ where either A_l is a foreign key of R_j referencing a potential key A_k of R_i, or vice versa. The association between A_k and A_l models one-to-many relations, i.e. one element of R_i can be associated with a set of elements of R_j. The notion of links built on top of the foreign key relations is critical for our relational representation since it will provide the basis for the definition of the *set* type that lies in the core of our relational representation.

One of the main limitations of the relational algebra representation is that, although it is ideal for modeling tuples and sets, it can not naturally model

lists. To be able to represent lists we extend the standard relational algebra by defining an extension of standard link which we call *list link* (*ll*) that adds the order information in the corresponding collection of objects. More precisely, a list link as a quintuple $ll(R_i, A_k, R_j, A_l, LIST(A_l))$ where R_i, A_k, R_j, A_l are defined as before and $LIST(A_l)$ is a list of values from $D(A_l)$ defining the order of the elements of the list. As with the standard link, the association between A_k and A_l encoded in *ll* models one-to-many relations; while for *sl* one element of R_i is associated with a set of elements of R_j, in the case of *ll* one element of R_i is connected with a list of elements from R_j[5].

As already mentioned, the objects of type *set* and *list* are based on the notion of links. In fact each link in which a relation R_i participates will give rise to an object of type *set* or *list*, depending on the type of the link. We will denote the set of sets related with relation R_i with $\mathcal{I}_{SL,R_i}$. Similarly the set of lists will be denoted with $\mathcal{I}_{LL,R_i}$. We will call the set of attributes of a relation R_i that are not keys (i.e. referenced keys, foreign keys or attributes defined as keys but not referenced) *standard* attributes and denote it with $\mathcal{I}_{A,R_i}$.

The above concepts provide a direct link with the definitions introduced on the conceptual level presented in Section 2.1. In particular, primitive data types that constitute a relation correspond to the primitive objects presented in Section 2.1.1. Tuples whose elements are primitive attributes (from $\mathcal{I}_{A,R_i}$ for $R_i \in \mathcal{R}$), sets (from $\mathcal{I}_{SL,R_i}$) and lists (from $\mathcal{I}_{LL,R_i}$) correspond to tuples presented Section 2.1.2. As already mentioned, the notion of set and list links which are defined by foreign key associations allow for direct modeling of objects of types *set* and *list* from Sections 2.1.3 and 2.1.4, respectively. Finally, in the next section we will describe how we can exploit the structures provided by relational algebra in order to define representation of composite objects, which directly correspond to general complex objects presented in Section 2.1.5.

2.2.2 Relational Instance

We will now describe how we can retrieve the description of a relational instance. Each tuple-instance, R_{i_a}, of a relation, R_i, can give rise to a *relational instance*, $R^+_{i_a}$. A relational instance is defined recursively in terms of the instances with which R_{i_a} is related. It is a tree structure whose root contains R_{i_a}. Each node at the second level of the tree is a set (list) of instances from some relation $R_j \in \mathcal{R}$ related via a link $sl(R_i, A_l, R_j, A_k)$ (or $ll(R_i, A_l, R_j, A_k, LIST(A_l))$) with instance R_{i_a}. In the same way nodes at level d of the tree are also sets (or lists) of instances from a given relation. Each of these sets (lists) is related with one of the instances found in a set (list) of nodes at level $d-1$. One can view a relational instance $R^+_{i_a}$ as that snapshot of the dataset that we get when we start from instance R_{i_a} of R_i and retrieve instances by recursively following the links defined in the relational schema.

To get the complete description of $R^+_{i_a}$ one will have to traverse possibly all the relational schema according to the relation associations, i.e., links, de-

[5]More generally one element of R_i can be associated with a list which elements are sets of tuples from R_j.

fined in the schema. This is done via the recursive application of the function $\boldsymbol{R_t^+(R_i, R_{i_a}, \cdot, \cdot)}$ (Algorithm 2.1), on the associated tuples of R_{i_a} in the relations given by the links $SL(R_i)$, $SL^{-1}(R_i)$ for sets, and $LL(R_i)$, $LL^{-1}(R_i)$ for lists. The function takes as input an instance R_{i_a} of a relation R_i for which it returns its corresponding relational instance $R_{i_a}^+$. The reason we follow links in both directions is that an entity is described both in terms of the entities it refers to, $SL^{-1}(R_i)$ (or $LL^{-1}(R_i)$), but also in terms of the entities that refer to it $SL(R_i)$ ($LL(R_i)$).

More precisely, for a given tuple, R_{i_a}, of any relation, R_i, the function $\boldsymbol{R_t^+(R_i, R_{i_a}, \cdot, \cdot)}$ will create a relational instance $R_{i_j}^+$ that will have the same set of standard attributes $\mathcal{I}_{A,R_i}$ and the same values for these attributes as R_{i_a} has (Algorithm 2.1, Lines 13-15). Furthermore, for each link $sl(R_i, A_l, R_j, A_k) \in SL(R_i) \cup SL^{-1}(R_i)$ it will add in $R_{i_j}^+$ one attribute of type *set*, constructing in this way the set attributes, $\mathcal{I}_{SL,R_i}$. The value of an attribute of type *set* is defined based on the link *sl* with which the attribute is associated, and it will be the set of tuples – relational instances – with which R_{i_a} is associated in relation R_j when we follow *sl* (Lines 17-19). This set is retrieved via the application of the function $\boldsymbol{set(R_{i_a}.A_l, R_j, A_k, \cdot, \cdot)}$ (Line 18). Similarly, to obtain attributes of type list we follow all the list links (Lines 21-23). Then the corresponding lists are retrieved by using $\boldsymbol{list(R_{i_a}.A_l, R_j, A_k, LIST(A_l), \cdot, \cdot)}$.

The ***set*** and ***list*** functions, (given in the second part of Algorithm 2.1), first perform a simple SQL query which returns the collection of tuples of relation R_j for which $A_k = R_{i_a}.A_l$, i.e. the set or lists of tuples related with R_{i_a} in the R_j relation (Algorithm 2.1, ***set*** function, Line 2 and ***list*** function, Line 2). Then it returns the corresponding set (list) of relational instances computed by the $\boldsymbol{R_t^+}$ function for each of the elements of the initial set (list).

To summarize, a relational instance $R_{i_a}^+$ consists of three parts. The first one corresponds to the set of $\mathcal{I}_{A,R_i}$ attributes of R_{i_a}, the second one to the attributes of type set, $\mathcal{I}_{SL,R_i}$, and the third one to the attributes of type list, $\mathcal{I}_{LL,R_i}$. The last two types are constructed on the basis of the links in which R_{i_a} participates.

Traversing the relational schema in order to retrieve the complete description of a given relational instance can easily produce self replicated loops. For example, when we follow twice in the row the same link: if we are at an instance R_{i_j} of relation R_i and we follow the link $sl(R_i, A_l, R_j, A_k)$ whose opposite link, i.e. $sl(R_j, A_k, R_i, A_l)$ lead us to R_{i_j}, then among other instances of relation R_j we will also visit again that instance of R_j that brought us to R_{i_j}, whose information has already been accounted for. The same situation can appear when there are two foreign keys, $A_{f_{1_k}}, A_{f_{2_k}}$, in relation R_j on the same potential key A_k of a relation R_i. We have chosen to terminate the acquisition of information when a self replicated loop appears. To do that we keep track of all the instances of the different relations that appear in a given path of the recursion, this is the role of the *instStack* variable in Algorithm 2.1. The moment an instance appears a second time in the given recursion path the recursion terminates (Algorithm 2.1, Line 8). In the next section we will give an example of the

Algorithm 2.1 Retrieving a tree description of a relational instance.

1: $\boldsymbol{R_t^+(R_i, R_{i_a}, d, instStack)}$
2: // R_{i_a}: instance for which we want to create its relational instance, $R_{i_a}^+$
3: // R_i: the relation to which R_{i_a} belongs to
4: // d: the depth of recursion to which we are
5: // $instStack$: instances visited so far in the current recursion path
6:
7: **if** $R_{i_a} \in instStack$ **then**
8: **return** null // Came to a loop
9: **else**
10: $stack.push(R_{i_a})$
11: **end if**
12: // Get the values of all standard attributes (i.e. attribute-value) for $R_{i_a}^+$
13: **for** $A_l \in \mathcal{I}_{A,R_i}$ **do**
14: $R_{i_a}^+.A_l \leftarrow R_{i_a}.A_l$
15: **end for**
16: // Recuperate the values of all set attributes for $R_{i_a}^+$
17: **for all** $sl(R_i, A_l, R_j, A_k) \in SL(R_i) \cup SL^{-1}(R_i)$ **do**
18: $R_{i_a}^+.A_l \leftarrow set(R_{i_a}.A_l, R_j, A_k, d+1, instStack)$
19: **end for**
20: // Recuperate the values of all list attributes for $R_{i_a}^+$
21: **for all** $ll(R_i, A_l, R_j, A_k, LIST(A_l)) \in LL(R_i) \cup LL^{-1}(R_i)$ **do**
22: $R_{i_a}^+.A_l \leftarrow list(R_{i_a}.A_l, R_j, A_k, LIST(A_l), d+1, instStack)$
23: **end for**
24: **return** $R_{i_a}^+$

1: $\boldsymbol{set(R_{i_a}.A_l, R_j, A_k, d, instStack)}$
2: $set \leftarrow \{R_{j_m} \in R_j \ : \ R_{i_a}.A_l = R_{j_m}.A_k\}$
3: $set' \leftarrow \emptyset$
4: **for all** $R_{j_b} \in set$ **do**
5: $R_{j_b}^+ \leftarrow R_t^+(R_j, R_{j_b}, d, instStack)$
6: $set' \leftarrow set' \cup R_{j_b}^+$
7: **end for**
8: **return** set'

1: $\boldsymbol{list(R_{i_a}.A_l, R_j, A_k, LIST(A_l), d, instStack)}$
2: $list \leftarrow [R_{j_m} \in R_j \ : \ R_{i_a}.A_l = R_{j_m}.A_k$ ORDER AS $LIST(A_l)]$
3: $list' \leftarrow []$
4: **for all** $R_{j_b} \in list$ **do**
5: $R_{j_b}^+ \leftarrow R_t^+(R_j, R_{j_b}, d, instStack)$
6: $list'.add(R_{j_b}^+)$
7: **end for**
8: **return** $list'$

construction of a relational instance. Within the example we will also provide a discussion of the issues related to self replicating loops. For the moment let us simply note that not allowing them seems a reasonable choice, since these would result in the introduction of redundant information; however, taking a closer look at the problem makes the other alternative equally plausible.

Finally, to define a learning problem one of the relations in $\mathcal{R}$ should be defined as the *main* relation, M, i.e. the relation on which the learning problem will be defined. Then for classification tasks one of the attributes of this relation should be defined as the class attribute, M_c, i.e. the attribute that defines the classification problem. Each of the instances of the relation M will give rise to their corresponding relational instances which are the ones that will be used during learning.

The use of the links defined on foreign keys guides the acquisition of the information related with a given instance. At each moment we know on which relations we should be looking and which attribute we should be using to retrieve that information. In Section 2.4 we will define a class of general binary operators over relational instances. The proposed operators are defined as (recursive) combinations of the operators defined over the building blocks, i.e. primitive attributes, tuples, sets and lists. In the next chapters we will focus on distance and kernel operators and propose various distances and kernels over these building blocks. We should note that we are not necessarily limited to distance and kernel based approaches. If the appropriate operators are defined for tuples, set and list attributes one can imagine relational learners of different paradigms, e.g. decision trees, linear learners etc. In Section 2.5 we will show how the relational representational paradigm that we have established here relates to some of the most common approaches used in Inductive Logic Programming.

2.2.3 An Example

We will now illustrate the main ideas of our approach with a simple example from the field of proteomics. More precisely, we will describe how by starting from a problem specification we can model the composite data and obtain its relational representation.

We consider a learning problem where the goal is to characterize proteins as belonging to a positive or negative class, based the description of the different substances with which they interact and the family to which they belong. We assume that as a problem specification we know that a given protein can interact in various ways with a different number of substances. Moreover, a given protein family can have various member proteins. Finally, protein families, substances as well as protein-substance interactions are characterized by a number of attributes (numbers and nominal attributes) and the number of these attributes is fixed for each type of the objects. For example, a protein family can be characterized by its type as well as some numeric attributes specifying coherence of member proteins.

Given the above definition of the problem we are now in position to propose a specific data representation. Different representations of this data are possible;

the one selected by a practitioner should reflect the semantics of the data as closely as possible. From the specification we can describe a protein family by a number of primitive objects of types $T_{f_1}, \ldots, T_{f_k}$ for some $k \in \mathbb{N}$, where T_{f_i} is either $numeric(a, b)$ or $symbolic(S)$ for some S, $i = 1, \ldots, k$. Similarly, substances and interactions are characterized respectively by $T_{s_1}, \ldots, T_{s_m}$ and $T_{i_1}, \ldots, T_{i_n}$ $(m, n \in \mathbb{N})$, where again T_{s_i} and T_{i_j} are $numeric(a, b)$ or $symbolic(S)$ for some S, $i = 1, \ldots, m$ and $j = 1, \ldots, n$. Since we know that the number of these attributes is fixed for a given type, we can group these primitive data types into tuples, i.e. we define $T_f = tuple(T_{f_1}, \ldots, T_{f_k})$, $T_s = tuple(T_{s_1}, \ldots, T_{s_m})$ and $T_i = tuple(T_{i_1}, \ldots, T_{i_n})$. As a result, an object of type T_f characterizes protein families while objects of types T_s and T_i characterize substances and interactions, respectively. A given protein can interact with a number of substances, and hence the description of a protein should contain information about both the *set* of substances with which it interacts and the interactions themselves. This suggests the following data type to be used for this purpose

$$set(tuple(T_i, T_s)).$$

Finally, a protein contains information both about interactions and protein families so that we obtain the following data type which characterizes proteins

$$tuple(T_f, set(tuple(T_i, T_s))).$$

As in (Gärtner et al., 2004), the formal specification of this data can be presented as follows

```
type Protein = tuple(Family, I-Ss)
type I-Ss = set(I-S)
type I-S = tuple(Interaction, Substance)
type Family = tuple(Prim,...,Prim)              // length k
type Substance = tuple(Prim,...,Prim)           // length m
type Interaction = tuple(Prim,...,Prim)         // length n
type Prim = numeric | symbolic
```

The above representation corresponds to a relational schema containing four tables. One corresponding to descriptions of families (T_f), *Families*, with primary key **F.ID**; a second one, *Substances*, with primary key **S.ID**, corresponding to descriptions of substances (T_s); a third one contains *Proteins*, with primary key **P.ID**; and finally a table defining the interactions *P-S-Interaction* (T_i). The latter has two foreign keys *P.ID, S.ID,* pointing respectively to the *Proteins* and *Substances* tables. To declare the family to which a protein belongs we add a foreign key, *F.ID*, to the table of *Proteins* pointing to the table of *Families*. The resulting relational schema is given in Figure 2.4. From the above relational description it is clear that the concept of a set is created on the basis of a foreign key, e.g. the *P.ID* foreign key in the *P-S-Interaction* table, which points to the *Proteins* table, models the I-Ss set. The example of particular values in this database is given in Table 2.1.

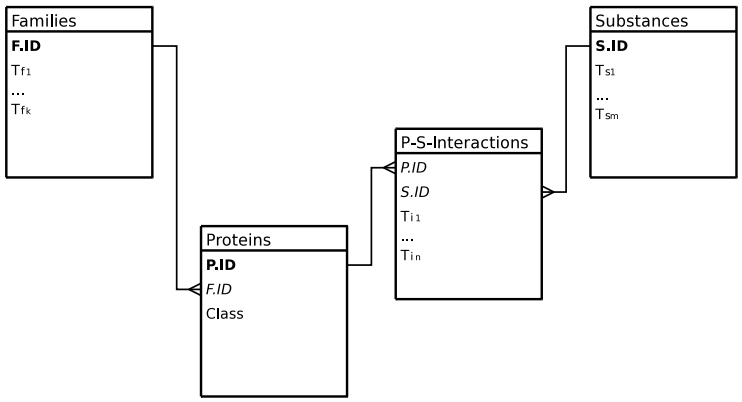


Figure 2.4: A relational schema corresponding to a sample database from the domain of proteomics. Referenced keys are marked in **bold**, foreign keys in *italics*.

Families		Proteins			P-S-Interaction		Substances	
F.ID	T_f	**P.ID**	*F.ID*	Class	*P.ID*	*S.ID*	T_i **S.ID**	T_s
F_a		P_A	F_a	+	P_A	D_a	D_a	
F_b		P_B	F_a	-	P_A	D_b	D_b	
F_c		P_C	F_c	+	P_B	D_z	D_z	

Table 2.1: The example database corresponding to the relational schema from Figure 2.4.

Let's suppose that we want to access the information related with the protein P_A in order to construct the corresponding relational instance. To that end we should traverse the complete relational schema in a recursive manner using the defined links. Looking at the *Proteins* relations we see that there is a foreign key *F.ID* referencing the *Families* relation so we have to move there to recuperate the description of the family to which the protein P_A belongs. The description of the family F_A associated with our protein P_A will add *one* attribute of type *set* to the attributes of the *Proteins* relation. Examining now the *Families* relation we see that it does not have any foreign keys that we could follow to another relation. However, its key **F.ID** is referenced by the *F.ID* of the *Proteins* relation; following that link back for the F_A brings us to a set of two instances from the proteins relation, this set of instances now adds one more attribute of type *set* to the description of F_A family. Nevertheless, P_A is already present in the recursion path so we exclude it from the set, according to the design choice on self replicating loops, and continue only with the P_B protein which via the *F.ID* foreign key will bring us back to the F_A instance of the *Families* relation. But F_A is also already present in the recursion path so we terminate here the recursion.

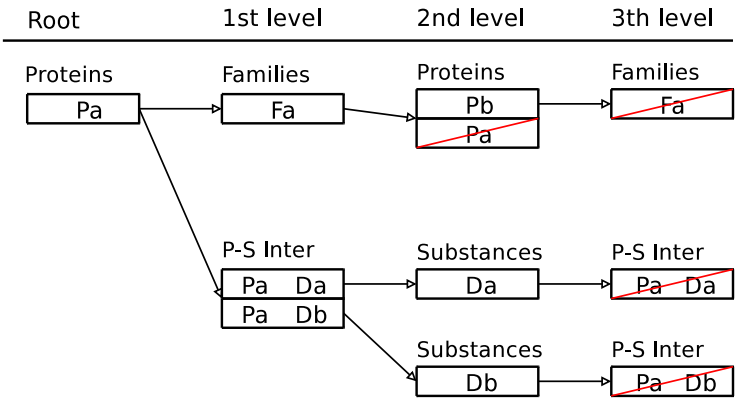


Figure 2.5: The relational instance that corresponds to protein Pa. The oblique lines indicate where recursion ends due to the appearance of a self replicating loop.

The relation between *Proteins* and *Families* clearly illustrates why links should be followed in both directions. A protein is described by the family to which it belongs so we should follow the link from proteins to families; at the same time a family is described by the proteins that form it, so we should now follow the same link in the opposite direction. It is questionable whether ignoring protein P_A when we come from the family F_A is the perfect choice. Ignoring P_A leaves somehow incomplete the description of F_A, especially if P_A is a typical or important member of the family. On the other hand the information that P_A brings has already been accounted for, although at a shallower level of the recursion.

Back in the *Proteins* relation the key **P.ID** is referenced by a foreign key in table *P-S-Interaction* so now we have to move there and recuperate all the instances of the *P-S-Interaction* table associated with our protein P_A. We see that there are two such instances. Again, this set of instances will contribute one more attribute of type *set* to the attributes of the *Proteins* relation. At relation *P-S-Interaction* we see that there is one foreign key *S.ID* pointing to the *Substances* relation which we should follow in order to get a description of each of the substances interacting with our protein. Again this description will contribute an attribute of type set in table *P-S-Interaction*. At the *Substances* relation, following back the links in which it participates will bring us to instances that have already been accounted for, so the recursion ends here.

The result is that the final description of P_A will consist of two set attributes. The first one describes the family to which P_A belongs, a family which is described in terms of its protein members except P_A. The second set attribute describes the set of substances with which P_A interacts, substances whose description was retrieved from the *Substances* table. The resulting relational instance is given in Figure 2.5.

2.3 Graphs and Trees

In Section 2.2 we introduced the extended relational algebra representation and provided a link with the high-level concepts presented in Section 2.1. In particular, within the proposed formalism it is possible to directly represent composite objects such as tuples, sets and lists. Moreover, general composite objects modeled using relational formalism can be viewed as tree structures. In this section we show how the relational representation can be exploited to model other, more general structures. In particular, we demonstrate how to represent *labeled graphs*.

Informally speaking, graphs and trees consist of *vertices* (or *nodes*) and *edges* that connect pairs of vertices. Graphs are widely used tools to model pairwise relations between objects from a certain collection. As a result, graphs are among the most common and well-studied combinatorial structures in computer science and are widely used to model complex data. In the area of machine learning and data mining these complex structures have been used to model data in bioinformatics (e.g. RNA secondary structures) (Zhang and Shasha, 1989; Passerini et al., 2006), chemoinformatics (e.g. chemical compounds) (Ralaivola et al., 2005; Ramon and Gärtner, 2003; Deshpande et al., 2003), Natural Language Processing (parse or dependency trees) (Collins and Duffy, 2002; Zelenko et al., 2003; Culotta and Sorensen, 2004) and various types of network data (e.g transportation systems, communication networks, social network) (Chakrabarti and Faloutsos, 2006).

We will not directly represent graphs, instead our representation is based on decompositions of these complex structures into sub-graphs of different types. In particular, we focus on decompositions into three types of structures: walks, trees and tree-like structures. These representations provide only an approximation of these complex structures, nevertheless this approach is widely used in practice, as it reduces the computational complexity of graph-based algorithms, and in terms of predictive performance it has been proved to be quite effective. We mention that within the relational algebra formalism it is possible to model labeled graphs without a loss of information, however, because of the above mentioned advantages of different approximations, we will not exploit the "exact" representation in the remaining part of this study.

In order to formally introduce walks, trees and other related structures we first provide some basic concepts and notations of the mathematical graph theory. For more in-depth discussion of graphs and related concepts the reader is referred to (Diestel, 2005).

Definition 2.1 (Graph)**.** A *graph* is defined as an ordered pair $G = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = \{v_1, \ldots, v_k\}$ is a finite set of *vertices* and $\mathcal{E}$ is a finite set of *edges*. In the case of *undirected graphs* $\mathcal{E} = \{\{v_i, v_j\} : v_i, v_j \in \mathcal{V}\}$, while for *directed graphs* $\mathcal{E} = \{(v_i, v_j) : v_i, v_j \in \mathcal{V}\}$.

We denote the number of vertices in a graph G by $|G(\mathcal{V})|$ (or simply by $|G|$) and the number of edges by $|G(\mathcal{E})|$. Moreover, we shall use the following notation for edges: $e_{ij} = \{v_i, v_j\}$ (or $e_{ij} = (v_i, v_j)$ for directed graphs). In the

remaining part of this study we will focus on graphs where each vertex and edge is labeled.

Definition 2.2 (Labeled Graph)**.** A *labeled graph* is a graph $G = (\mathcal{V}, \mathcal{E})$ where there is additionally a set of vertex labels $\mathcal{L}_\mathcal{V}$ and edge labels $\mathcal{L}_\mathcal{E}$ together with functions $l_\mathcal{V} : \mathcal{V} \to \mathcal{L}_\mathcal{V}$ and $l_\mathcal{E} : \mathcal{E} \to \mathcal{L}_\mathcal{E}$ that assign labels to vertices and edges, respectively.

Usually we assume that all the elements in $\mathcal{L}_\mathcal{V}$ are of the same type denoted by $T_\mathcal{V}$. Similarly, all the elements in $\mathcal{L}_\mathcal{E}$ are assumed to be of type $T_\mathcal{E}$. Traditionally, only graphs with symbolic labels have been considered in the literature (i.e. $T_\mathcal{V} = T_\mathcal{E} = symbolic(S)$ for some S), however, in general there is no restriction on the types of objects in both $\mathcal{L}_\mathcal{V}$ and $\mathcal{L}_\mathcal{E}$. In the rest on this work we will usually assume that the labels are vectors of the same type, i.e. $\mathcal{L}_\mathcal{V} \subseteq \mathbb{R}^m$ and $\mathcal{L}_\mathcal{E} \subseteq \mathbb{R}^n$ for some values of m and n. We will use $lab(x)$ to denote, in a more general form, the label of x; whether $lab(x) = l_\mathcal{V}(x)$ or $lab(x) = l_\mathcal{E}(x)$ will be clear from the type of the x argument, i.e. vertex or edge. By $dim(lab(x))$ we will denote the dimensionality of the label vector $lab(x)$ which obviously will be the dimensionality of either $\mathcal{L}_\mathcal{V}$ or $\mathcal{L}_\mathcal{E}$ depending again on the type of the x argument.

One the of most popular approaches for representing and handling graph structures is based in an algebraic framework. The central concept here is the *adjacency matrix*.

Definition 2.3 (Adjacency Matrix)**.** The *adjacency matrix* $\boldsymbol{A}$ of $G(\mathcal{V}, \mathcal{E})$ is defined as $A_{ij} = 1 \iff \{v_i, v_j\} \in \mathcal{E}$ (or $A_{ij} = 1 \iff (v_i, v_j) \in \mathcal{E}$ for directed graphs) and $A_{ij} = 0$ otherwise.

It can be shown that there exists an unique adjacency matrix for each graph (up to permuting rows and columns), and it is not the adjacency matrix of any other graph (Diestel, 2005). Now we are in position to formally define different specialized types of graphs.

Trees and Other Types of Graphs

Some special types of graphs that are relevant to our work are *walks* and *trees*.

Definition 2.4 (Walk)**.** A *walk* W in a graph G is a sequence of vertices and edges, $W = [v_1, e_{12}, v_2, \ldots, e_{s,s+1}, v_{s+1}]$, such that $v_j \in \mathcal{V}$ for $1 \leqslant j \leqslant s+1$ and $e_{ij} \in \mathcal{E}$ for $1 \leqslant i \leqslant s$. Walks may end in an edge, e.g. $W = [v_1, e_{12}, v_2, \ldots, e_{s,s+1}]$.

The length l of a walk W, denoted as $length(W)$, is the number of vertices and edges in W; for $W = [v_1, e_{12}, v_2, \ldots, e_{s,s+1}, v_{s+1}]$ it is $l = 2s+1$. We denote $W[i]$ its element at the position $i = 1, \ldots, l$. Finally, the set of all walks of length l in a graph G is $\mathcal{W}(G)^l$.

Definition 2.5 (Tree)**.** A *tree* T is defined as graph where any two vertices are connected by exactly one path, i.e. a tree is a *connected* and *acyclic* graph.

As in the case of general graphs we are mainly interested in labeled trees where labels are assigned to both nodes and edges. Similarly to graphs the size of a tree T is denoted by $|T|$, i.e. the number of nodes in T. The *root* of T is denoted by $root(T)$; obviously $root(T) \in \mathcal{V}$. In this work we focus on *directed rooted trees* where the "direction" is from the root down to the leafs. The height h of a tree T, denoted as $height(T)$, is the length of the longest walk from the root node to any of the leaf nodes. It should be mentioned that as in walks we allow trees to have edges as leafs. We denote a set of trees by $\mathcal{T}$, and a set of all trees of height h in a graph G by $\mathcal{T}(G)^h$. For directed graphs and trees we define the neighborhood of a node, v, as $\delta(v) = \{e : e = (v, u) \in \mathcal{E}\}$, and the neighborhood of an edge, e, as $\delta(e) = \{u : e = (v, u) \in \mathcal{E}\}$; note that in fact the neighborhood of an edge is a one element set, containing a single node. For undirected graphs and trees the corresponding neighborhoods are defined as $\delta(v) = \{e : e = \{v, u\} \in \mathcal{E}\}$ and $\delta(e) = \{u : e = \{v, u\} \in \mathcal{E}\}$. We call a node v a *leaf* iff $\delta(v) = \emptyset$. Similarly we call an edge e a leaf iff $\delta(e) = \emptyset$. Finally, for a node v the elements of $\delta(v)$ are called the *children edges* of v and the elements of $\{u : (v, u) \in \mathcal{E}\}$ are the *children nodes* of v. It is important to realize that a tree T can be alternatively defined by recursive alternating application of $\delta(v)$ and $\delta(e)$, where the recursion starts with $\delta(root(T))$. Finally, we note that in the trees we consider, elements at a given height are of the same type, and will either $T_{\mathcal{V}}$ (for nodes) or $T_{\mathcal{E}}$ (for edges).

The main focus in this work is on *unordered trees*, i.e. trees where no order is imposed among the elements of $\delta(v)$ for $v \in \mathcal{V}$. If the order among elements of $\delta(v)$ is given (e.g. by the labels of the edges) we obtain *ordered trees*. We could also combine ordered with unordered trees (i.e. $\exists v \in \mathcal{V}$ where the order among $\delta(v)$ is given, and $\exists v \in \mathcal{V}$ for which the order among $\delta(v)$ is not specified) such that we obtain *mixed trees*.

2.3.1 Representation of Graphs

As already mentioned, graphs are not directly represented within our framework. Instead, we represent graphs as sets resulting from their decompositions into different sub-parts. Depending on the actual type of elements in the decompositions, different graph representations are obtained. It should be stressed that in any case the resulting representation provides only an approximation of graphs, and depending on the decompositions the resulting representation could be more or less accurate. For example, it is expected that decompositions into a set of atoms will be less accurate than decompositions into, say cyclic patterns as considered by Horváth et al. (2004). Moreover, the actual representation directly influences the computational complexity of the different machine learning algorithms applied over this representation. As a result, decompositions into a set of atoms might be appropriate for a given problem and decomposition into other substructures will be unnecessarily complex.

This approach of representing graphs is widely used and has been proved to be quite effective in practical applications. For example, in almost all the kernels over labeled graphs, the decompositions are among others into walks (Gärtner,

2002; Kashima et al., 2003; Mahé et al., 2004), shortest paths (Borgwardt and Kriegel, 2005) and subtrees (Ramon and Gärtner, 2003)[6]. The problem of selecting the "proper" decomposition from a number of predefined decompositions (or more generally combination of these decompositions) will be the focus in Chapter 5.

In this work we focus on decompositions into three types of substructures: (i) walks of various lengths, (ii) trees of different heights and (iii) particular types of tree-like structures. In the rest of this section we will describe these decompositions in more detail and show how these structures can be modeled within our relational representation.

The two first decompositions based on walks and trees are obtained from a depth first exploration, that includes both vertices and edges, emanating from each node in a graph and yielding all the walks of length l and all the trees of height h. We mention that we only consider *unordered trees* where the order among the siblings at a given height is not important. In particular, for $l = 1$ (and $h = 1$) a graph is represented as the set of all its vertices. For $l = 2$ a graph is decomposed into a set of two-element tuples with the first element of each pair being a vertex, and the second – one of its adjacent edges. Similarly for $h = 2$ the corresponding decomposition is into trees with vertices as roots connected to all their adjacent edges. More formally, the decomposition into trees of height h of a graph $G = (\mathcal{V}, \mathcal{E})$ can be represented as a set of trees $\mathcal{T}(G)^h = \{T_1, \ldots, T_r\}$ of various heights where the root of each tree is a node from G, i.e. $root(T_i) \in \mathcal{V}$ for $i = 1, \ldots, r$ and $height(T_i) = h$. Obviously $r = |G(\mathcal{V})|$. For decompositions into walks of length l, a graph G can be represented as a set of walks $\mathcal{W}(G)^l = \{W_1, \ldots, W_s\}$ where $\forall_{W_i} \exists_{v \in \mathcal{V}}$ such that $W_i[1] = v$. Moreover, $length(W_i) = l$ and in general for long walks the cardinalities of the decompositions will be much larger that number of vertices in a graph, i.e. $s \gg |G(\mathcal{V})|$.

In our decompositions into walks and graphs we do not allow repetitions of the same two-nodes-cycles. As a result, we avoid the problem of *tottering*. That is, we exclude from the decomposition walks of the form $W = [v_1, e_{12}, v_2, \ldots, e_{s,s+1}, v_{s+1}]$ with $v_i = v_{i+2}$ for some i since this is likely to remove noise from the representation of a graph. As an example consider a chemical molecule represented as a labeled graph where vertices are atoms and edges are covalent bonds. In such case the existence of a walk with labels C-C-C might indicate a succession of 3 C-labeled vertices in the graph, or alternatively it might correspond to a succession of 2 C-labeled vertices visited by a tottering random walk (Mahé et al., 2004). In the case of trees we do not allow such walks in the trees' descriptions. The problem of tottering was first recognized in (Mahé et al., 2004) where the authors proposed a modification of the random walk model introduced by Kashima et al. (2003), however, their algorithm did not lead to an improvement in predictive performance. Similar observations were also reported in (Fröhlich et al., 2005) where the authors compared kernels based on trees with, and without, tottering. Even though the graph representation based

[6]Sometimes the decompositions are constructed implicitly. Examples include the works of Gärtner (2002) and Kashima et al. (2003).

on tottering-free walks does not seem to bring an improvement to the predictive performance, it has the advantage of inducing decompositions smaller in size. This has a direct influence on the computational complexity of the different data mining operators applied over this representation[7].

The last type of sub-graphs on which we focus in this study are recursive tree-like structures which are constructed by a depth first exploration emanating from each edge and where the recursion continues further through the two adjacent nodes. This is in contrast with "standard" trees which are constructed by recursive exploration which starts from $\delta(v)$ for $v \in \mathcal{V}$; tree-like structures are defined by alternating application of $\delta(e)$ and $\delta(v)$ and the recursion starts with $\delta(e)$ for $e \in \mathcal{E}$. In the case we limit the recursion depth to $d = 1$ the corresponding set of decomposition contains only edges, while for $d = 2$ the elements are tree-like structures having edges as roots which are connected to a set of corresponding vertices. For a graph $G = (\mathcal{V}, \mathcal{E})$ decomposed into tree-like structures with recursion depth limited to d we obtain a set $\{T'_1, \ldots, T'_t\}$ where $root(T'_i) \in \mathcal{E}$, $i = 1, \ldots, t$ and $t = |G(\mathcal{E})|$.

Relational Representation of Graphs

Decompositions into the above structures can be easily represented within our relational framework. In the remainder of this section we will formally model a labeled graph $G = (\mathcal{V}, \mathcal{E})$ for which we assume that $T_{\mathcal{V}}$ and $T_{\mathcal{E}}$ are the data types corresponding to labels of vertices and edges, respectively. We start with decompositions into walks and let $tuple_l(T_{\mathcal{V}}, T_{\mathcal{E}})$ be the data type corresponding to a walk W with length l, constructed by concatenating the labels of elements of W, i.e. $tuple_l(T_{\mathcal{V}}, T_{\mathcal{E}}) = tuple(\underbrace{T_{\mathcal{V}}, T_{\mathcal{E}}, \ldots}_{l})$. In this case G is of the following data type

$$set(tuple_l(T_{\mathcal{V}}, T_{\mathcal{E}})).$$

To formally define a data type of G, where decomposition into trees is used we define $tree_h(T_{\mathcal{V}}, T_{\mathcal{E}})$ as the data type of a tree of height h, i.e. for $h = 1$ the corresponding data type is $tree_1(T_{\mathcal{V}}, T_{\mathcal{E}}) = T_{\mathcal{V}}$; for $h = 2$ it is $tree_2(T_{\mathcal{V}}, T_{\mathcal{E}}) = tuple(T_{\mathcal{V}}, set(T_{\mathcal{E}}))$; for $h = 3$ we have $tree_3(T_{\mathcal{V}}, T_{\mathcal{E}}) = tuple(T_{\mathcal{V}}, set(tuple(T_{\mathcal{E}}, T_{\mathcal{V}})))$, etc. In this case G is of the following data type

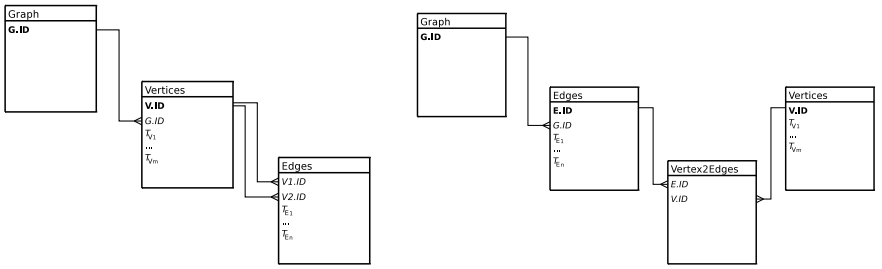
$$set(tree_h(T_{\mathcal{V}}, T_{\mathcal{E}})).$$

Finally, the data type of G, where decomposition into tree-like structures of depth d is used, is of the following data type

$$set(ts_d(T_{\mathcal{E}}, T_{\mathcal{V}}))$$

where $ts_d(T_{\mathcal{E}}, T_{\mathcal{V}})$ is a data type corresponding to a tree-like structure of depth d, and is defined in a similar way as $tree_h(T_{\mathcal{V}}, T_{\mathcal{E}})$.

[7]On the other hand, one problem with the decomposition of molecules into sets of walks of length l, in which tottering is not allowed, is that small molecules for large values of l will be associated with an empty set of walks. This problem will be discussed in detail in Section 5.4.



(a) (b)

Figure 2.6: Two relational representations of labeled graphs. The first one (a) corresponds to the decomposition into trees while the second one (b) corresponds to decompositions into tree-like structures.

Assuming that $T_{\mathcal{V}}$ and $T_{\mathcal{E}}$ are tuples of primitive data types, i.e. $T_{\mathcal{V}} = tuple(T_{\mathcal{V}_1}, \ldots, T_{\mathcal{V}_m})$ and $T_{\mathcal{E}} = tuple(T_{\mathcal{E}_1}, \ldots, T_{\mathcal{E}_n})$ where $T_{\mathcal{V}_i}, T_{\mathcal{V}_j}$ are either of type *numeric* or *symbolic* ($i = 1, \ldots, m$ and $j = 1, \ldots, n$) the trees and tree-like structures can be easily represented in a relational database. The two corresponding relational representations are presented in Figure 2.6. The relational representation in the case of trees is given in chart (a) while a relational structure corresponding to decomposition into tree-like structures is given in plot (b). Referenced keys are marked in **bold**, foreign keys in *italics*. Both of the representations assume that descriptions of edges in given in relation "Edges" and description of vertices are encoded in relation "Vertices". The main difference between these relational schemes is that in the decomposition into trees the recursion starts with the "Vertices" relation, while in decompositions into tree-like structures the recursion starts with "Edges".

2.4 Operators over Relational Representations

In this section we will define general data mining operators applied over the proposed relational representation; data mining operators will be the main focus in the remaining part of this work. In this study we only consider binary operators which take two relational instances as inputs and return a numeric value as an output. In particular, in Chapter 3 we focus on distance functions where the output value corresponds to the dissimilarity between two relational instances, while in Chapter 4 we propose various kernels which for two input objects return their similarity (or equivalently the inner product in some feature space). It should be mentioned that other data mining operators are also possible, e.g. unary operators performing relational feature selection, operators used within relational decision trees, etc.

As already mentioned in Section 2.1.5, our operators work in a recursive manner traversing the full tree structures of the input relational instances. During the recursion, the components of the relational instances are visited in exactly

the same order as when these were constructed (see Algorithm 2.1). The operators are defined as a recursive combination of operators defined on relational instances' sub-parts, both primitive (i.e. *numeric* and *symbolic*) and composite (*tuples*, *sets* and *lists*). For computational reasons the depth of recursion is controlled by a depth parameter which affects the estimation of the final operator value (in Section 3.4 we will examine its impact in the context of distances). Finally, the data mining operators we consider in this study are defined in a declarative manner. i.e. the practitioner specifies which operators should be used for each of the sub-objects. In general, there is a number of different data mining operators, each one with different semantics, and it is up to the analyst to declare which specific operator should be used for a given data type.

We denote the operators over two objects of data type *type* as follows

$$operator_{type}(\cdot,\cdot) \rightarrow \mathbb{R}$$

where *type* is *primitive*[8], *tuple*, *set* or *list*. We mention that in some cases the co-domain of $operator_{type}$ is a subset of $\mathbb{R}$, e.g. for the distance operators from Chapter 3, $operator_{type}$ takes values in $\mathbb{R}_0^+$. Moreover, we usually assume that $operator_{type}$ returns a normalized value, i.e. $\forall x, y\ operator_{type}(x, y) \leqslant 1$.

Algorithm 2.2 Operator on relational instances.

1: ***operator*** $(\boldsymbol{R_{i_a}^+}, \boldsymbol{R_{i_b}^+})$
2: // $R_{i_a}^+, R_{i_b}^+$: relational instances
3:
4: $R_{i_a} \leftarrow$ the main tuple-instance of $R_{i_a}^+$
5: $R_{i_b} \leftarrow$ the main tuple-instance of $R_{i_b}^+$
6: $R_i \leftarrow$ the relation to which R_{i_a} and R_{i_b} belong
7:
8: **return** $operator_{tuple}(R_{i_a}, R_{i_b}, R_i, [], [], 1)$

The full procedure of computing the operator over relational instances is given by Algorithm 2.2 which exploits the function defined in Algorithms 2.3 (Algorithm 2.3 uses the functions from Algorithms 2.4 and 2.5). The ***operator*** function, given by Algorithm 2.2, shows how the computation of the operator between any two relational instances, $R_{i_a}^+$, $R_{i_b}^+$ is done. This function directly calls the $\boldsymbol{operator_{tuple}}$ function from Algorithm 2.3 which takes as arguments the two tuples R_{i_a}, R_{i_b}, of a given relation R_i, which are the main tuple-instances of $R_{i_a}^+$ and $R_{i_b}^+$, respectively. After the current recursion depth (given by the d parameter) and the self replicated loops (variables *aStack* and *bStack*) are controlled the $\boldsymbol{operator_{tuple}}$ function is divided in two independent blocks. In the first part of this algorithm (Lines 20 to 31), from the two input tuples

[8]For simplicity, in this section we do not make a distinction between different primitive data types and assume that the operator $operator_{primitive}$ can be applied over objects of types *numeric* and *symbolic*.

Algorithm 2.3 Operator on tuples.

1: $\boldsymbol{operator_{tuple}(R_{i_a}, R_{i_b}, R_i, aStack, bStack, d)}$
2: // R_{i_a}, R_{i_b}: the tuple-instances from R_i
3: // $aStack$, $bStack$: instances visited so far during the recursion
4: // d: current depth of recursion
5:
6: **if** $d > MAX - DEPTH$ **then**
7: **return** null
8: **end if**
9: **if** $R_{i_a} \in aStack$ **then**
10: **return** null // A loop has occurred, R_{i_a} could be used to describe R_{i_a}
11: **else**
12: aStack.push(R_{i_a})
13: **end if**
14: **if** $R_{i_b} \in bStack$ **then**
15: **return** null // A loop has occurred, R_{i_b} could be used to describe R_{i_b}
16: **else**
17: bStack.push(R_{i_b})
18: **end if**
19:
20: // Recuperate elements of tuples and apply corresponding operators
21: $array \leftarrow$ vector of size $|\mathcal{I}_{A,R_i}| + |\mathcal{I}_{SL,R_i}| + |\mathcal{I}_{LL,R_i}|$
22: **for** $A_k \in \mathcal{I}_{A,R_i}$ **do**
23: $array(k) \leftarrow operator_{primitive}(v_{ak}, v_{bk})$
24: **end for**
25: **for** $sl(R_i, A_k, R_j, A_l) \in SL(R_i, \_) \cup SL^{-1}(R_i, \_)$ **do**
26: $array(k) \leftarrow operator_{set}(v_{ak}, v_{bk}, R_j, aStack, bStack, d+1)$
27: **end for**
28: **for** $ll(R_i, A_k, R_j, A_l, LIST(A_l)) \in LL(R_i, \_) \cup LL^{-1}(R_i, \_)$ **do**
29: $array(k) \leftarrow operator_{list}(v_{ak}, v_{bk}, R_j, aStack, bStack, d+1)$
30: **end for**
31: aStack.pop(); bStack.pop();
32:
33: // Combine results
34: **return** $combine_{tuple}(array)$

Algorithm 2.4 Operator on sets.

1: $\boldsymbol{operator_{set}(v_a, v_b, R_j, aStack, bStack, d)}$
2: // v_a, v_b: *sets* of relational instances from R_j
3: **if** $v_a = \emptyset$ XOR $v_b = \emptyset$ **then**
4: **return** $onlyOneEmptyValue$
5: **else if** $v_a = \emptyset$ AND $v_b = \emptyset$ **then**
6: **return** $bothEmptyValue$
7: **end if**
8:
9: // Recuperate elements of sets and apply operators on tuples
10: $bag \leftarrow |v_a| \times |v_b|$ $array$
11: **for all** $(R_{j_k}, R_{j_l}) \in v_a \times v_b$ **do**
12: $o \leftarrow operator_{tuple}(R_j, R_{j_k}, R_{j_l}, aStack, bStack, d)$
13: **if** $o \neq$ null **then**
14: $bag.add(R_j, R_{j_k}, R_{j_l}, o)$
15: **end if**
16: **end for**
17:
18: // Combine results
19: **return** $combine_{set}(bag)$

Algorithm 2.5 Operator on lists.

1: $\boldsymbol{operator_{list}(v_a, v_b, R_j, aStack, bStack, d)}$
2: // v_a, v_b: *lists* of relational instances from R_j
3: **if** $v_a = \emptyset$ XOR $v_b = \emptyset$ **then**
4: **return** $onlyOneEmptyValue$
5: **else if** $v_a = \emptyset$ AND $v_b = \emptyset$ **then**
6: **return** $bothEmptyValue$
7: **end if**
8:
9: // Recuperate elements of lists and apply operators on tuples
10: $bag \leftarrow |v_a| \times |v_b|$ $array$
11: **for all** $(R_{j_k}, R_{j_l}) \in v_a \times v_b$ **do**
12: $o \leftarrow operator_{tuple}(R_j, R_{j_k}, R_{j_l}, aStack, bStack, d)$
13: **if** $o \neq$ null **then**
14: $bag.add(R_j, R_{j_k}, R_{j_l}, o)$
15: **end if**
16: **end for**
17:
18: // Combine results
19: **return** $combine_{list}(bag)$

***operator*$_{tuple}$** recuperates all the values (v_{ak} and v_{bk}) of an attribute A_k which is of type *primitive* (Line 22), *set* (Line 25) and *list* (Line 28). The corresponding objects are matched by means of the appropriate operators and the results are stored in an one-dimensional array of size being the total number of all attributes. In the second part of the algorithm, after all pairwise operators between the relational instances have been computed, the $combine_{tuple}(\cdot) \rightarrow \mathbb{R}$ function is applied on that collection and combines the corresponding results. Different combination strategies are possible and the one used is selected according to the user's choice. For example, in the context of distance measures, we can define this combination as a normalized Euclidean metric (Definition 3.28); for kernels the possible combinations include the simple (normalized) sum (Definition 4.5) or product (Definition 4.6) of the corresponding sub-kernels.

If A_k is an object of type set (Line 25) or list (Line 28) then the values v_{ak}, v_{bk}, are actually the two sets (lists) of relational instances that are associated with R_{i_a} and R_{i_b} in relation R_j when we follow the link $sl(R_i, A_k, R_j, A_l)$ ($ll(R_i, A_k, R_j, A_l, LIST(A_k))$). In order to compute the operator between these two sets (lists) we use the function $operator_{set}$ ($operator_{list}$) given in Algorithm 2.4 (2.5). If both input objects are empty then $operator_{set}$ ($operator_{list}$) will return a value specified by $bothEmptyValue$; if only one of them is empty then the corresponding operator will return a value $onlyOneEmptyValue$. The values of both constants depend on the actual type of operator: for distances it is natural to set $bothEmptyValue$ to be 0, and $onlyOneEmptyValue$ to the maximum possible distance (since we are working with normalized operators this will be 1). In the case of kernel operators, to assure that we obtain valid kernels (Definition 4.2), we set $bothEmptyValue$ to 1 and $onlyOneEmptyValue$ to 0. In the case when both objects are not empty, $operator_{set}$ and $operator_{list}$ will work in a similar manner as the operator defined on tuples. More precisely, we first recuperate all the elements of sets (lists) and compute all the pairwise operators between the elements of the two sets (lists). Remember that these sets (lists) are sets (lists) of relational instances whose distances have to be computed by using again the $operator_{tuple}$ function given by Algorithm 2.3. The results of the matching process are stored in a two dimensional array where its rows and columns correspond to the elements of the two corresponding sets (lists). After all pairwise operators between the relational instances have been computed we apply the function $combine_{set}$ ($combine_{list}$) combining the results. Again, the particular combination schema is specified by a practitioner and should reflect the semantics of the problem at hand. For example, in the context of distances $combine_{set}$ can be defined as the minimum of all the distances over sets' elements (Definition 3.15), while for kernels $combine_{set}$ is usually defined as the average value of all the corresponding sub-kernels (Section 4.2.3).

To summarize, the general data mining operators $operator_{type}$ over relational instances are defined as a particular combination of operators defined over objects sub-parts. In all the sub-operators, with the exception of $operator_{primitive}$, we first recuperate the actual structure of objects from the corresponding relational representations and compute all the pairwise operators between the objects' decompositions. It is the internal structure of a composite object which

determines which elements could be matched. For example, in case of tuples only elements corresponding to a given tuple's dimension can be compared. On the other hand, sets and lists are less structured, and hence for these objects we have more freedom in matching their sub-parts. The actual value returned by the operator is computed by combining the corresponding results of pairwise sub-parts matching. This is implemented in the $combine_{type}$ function. Finally, we mention that in case of $operator_{primitive}$, the above procedure is greatly simplified since primitive attributes are not decomposable, and hence the corresponding retrieval and combination processes are trivial.

An Example

As already mentioned our system is declarative in nature so that the practitioner can declare which operator should be used for each building block that constitutes a learning instance. In this section we will provide an example of how it can be done in practice; we will show how this can be done using the problem from proteomics from Section 2.2.3. Assuming that we work with distances, we could declare the operators for different data types as follows

```
type Protein = tuple(Family, I-Ss)          : distanceTuple1
type I-Ss = set(I-S)                        : distanceSet
type I-S = tuple(Interaction, Substance)    : distanceTuple2
type Family = tuple(Prim,...,Prim)          : distanceTuple2
type Substance = tuple(Prim,...,Prim)       : distanceTuple1
type Interaction = tuple(Prim,...,Prim)     : distanceTuple1
type Prim = numeric | symbolic
symbolic                                    : distanceSymbolic
numeric                                     : distanceNumeric
```

where $distanceTuple1$ and $distanceTuple2$ are different distances over *tuple* data types, and $distanceNumeric$, $distanceSymbolic$ and $distanceSet$ are distances over *numeric*, *symbolic* and *set* data types, respectively. Note, that although $Interaction$ and $Family$ are the same general data types, i.e. *tuple*, they have been assigned different distances. On the other hand, the $Interaction$, $Substance$ and $Protein$ data types are associated with the same distance over tuples, even though they occur on different depths of the composite object.

Parameterized Operators

In the remaining part of this study we will also consider *parameterized* operators which are characterized by the fact that the matching of the corresponding parts (implemented by the $combine_{type} \rightarrow \mathbb{R}$ function in Algorithms 2.3, 2.4 and 2.5) depends on a number of adjustable parameters. The main advantage of such parameterized operators is that we can adjust their parameters (preferably within the learning process) so that they better reflect the semantic of a problem at hand.

More formally, we assume that the (numeric) parameters are given by $\boldsymbol{P}$. In order to incorporate these parameters to the various operators we extended the definition of the function $combine_{type}(\cdot) \rightarrow \mathbb{R}$ as

$$combine_{type,\boldsymbol{P}}(\cdot) \rightarrow \mathbb{R}.$$

In the context of distances, one particular example of the above operator that we consider in this work is the extension of $combine_{tuple}(t)$ (t is a vector) exploited in Algorithm 2.3. It is straightforward to define $\boldsymbol{P}$ as a vector of real numbers where each dimension corresponds to the "importance" of each element in t. In the case $combine_{tuple}(t)$ is the Euclidean metric, then $combine_{tuple,\boldsymbol{P}}(t)$ defines a weighted Euclidean metric. More generally, $\boldsymbol{P}$ can take a form of a positive semi-definite matrix such that $combine_{tuple,\boldsymbol{P}}(t)$ corresponds to the Mahalanobis distance measure (Definition 3.14). In Chapter 5 we will examine different ways such that $\boldsymbol{P}$ can be automatically adjusted for a problem at hand.

2.5 Related Work

In this section we will describe some of the related work which is relevant to relational algebra representation.

As already mentioned the proposed relational formalism can be considered as a specialized version of the formalism based on the higher-order logic of Lloyd (2003); Gärtner et al. (2004). The representation language that they used is that of typed λ-calculus which allows for the natural representation of sets and multisets, a main difference from first order terms. The proposed approach allows also for the modeling of more complex structures like lists, trees or graphs. Individuals are represented as terms of the typed λ-calculus formal logic. The composition of individuals from their parts is expressed in terms of a fixed type structure that is made up of function types, product types and type constructors. Function types are used to represent types corresponding to sets and multisets, product types represent types corresponding to fixed size tuples and type constructors (of various arities) for structured objects such as lists and trees. Moreover, type constructors of arity 0 are used to construct elementary types (called closed types) such as booleans and numbers. The definition of graphs in this framework closely corresponds to the mathematical definition of labeled graphs (Section 2.3). Each type defines a set of terms that represent instances of that type.

The concepts used in (Lloyd, 2003) can be directly mapped to concepts used in our work. In particular, function types correspond to sets while product types are directly related to tuples. Moreover, in (Lloyd, 2003) the types constructors of arity 0 give rise to primitive data types while a particular class of type constructor corresponds to lists. Our relational formalism is less expressive since it is function-free. Additionally, our formalism is less formal, however, it is more intuitive, easier to understand and straightforward to use in practice. There is also a difference when dealing with objects of type set and list. For example, lists considered by (Gärtner et al., 2004) could in fact contain elements

of different type. This might introduce a computational burden while applying a data mining operator (e.g. a distance or a positive semi-definite kernel) over these structures since the elements of different types need to handled correctly (see Gärtner et al., 2004, Example 4.1). On the other hand multi-sets in the λ-calculus framework are potentially handled more efficiently since the multiplicity of the corresponding elements is already given and does not have to be determined each time from the sets' structure.

The modular representation of data is widely used in object-oriented databases (Kim, 1990; Cattell et al., 2000). Most of the existing systems are in fact object-relational databases where the support to object-oriented data modeling is built "on top" of the standard relational functionality (Stonebraker, 1996). In addition to the data types considered in this study, other data types are useful in practical applications. Examples include e.g. objects of type *stack*, *queue* and *array* (Stonebraker, 1996). The other major difference is that the existing object-oriented database systems support objects' inheritance, which allows for more flexibility in defining new data types.

Comparison with Inductive Logic Programming

There is a straightforward mapping of the concepts of relational algebra to these of logic programming (Džeroski and Lavrac, 2001):

- a relation name R_i is a predicate symbol R_i,
- a relation R_i, i.e. a set of tuples, corresponds to a predicate R_i defined extensionally as a collection of ground facts,
- an attribute A_l of the relation R_i is an argument of the predicate R_i,
- a tuple R_{i_j} corresponds to the j^{th} ground fact built on the predicate R_i.

With respect to the above definitions and mappings the main difference of relational algebra from logic programming is the direct ability of the former to define types, i.e. each attribute of a relation has a given type defined by its domain, and associations between tuples based on the notion of foreign keys.

One of the first systems directly exploiting the concepts of relational algebra and foreign keys was *MIDOS* (Wrobel, 1997). However, the work of Wrobel (1997) is was only focused on the KDD subgroup discovery task whereas our system can be used in various learning paradigms.

Two of the most popular learning paradigms in relational learning are learning from *interpretations* and learning from *entailment* (Muggleton, 1995; Raedt, 1997; Džeroski and Lavrac, 2001; Blockeel, 1998). We will briefly sketch the representation differences between the two settings. In relational learning knowledge is usually separated to two parts, knowledge concerning the training examples and background knowledge, B, i.e. knowledge which is common to many examples. The differences in the two approaches are in the way knowledge is represented and in the notion of coverage that they use. We will focus mainly on the knowledge representation.

In learning from interpretations each example is described by a set of ground facts, e. The interpretation that represents the example is the set of all the ground facts that are entailed by $e \cap B$, i.e. the minimal Herbrand model of $e \cap B$. Usually each example comes in the form of a Prolog program; background knowledge is also represented in the form of a Prolog program. In learning from interpretations there is a clear separation of examples and background knowledge. There is also a clear separation between examples since one of its main assumptions is that these are independent of each other, i.e. there are no associations between them. For example, a coverage test would examine only a single example at the time.

In learning from entailment learning examples are in general represented by clauses but in practice are most often represented by a single fact, everything else even information concerning an individual example is placed within B, usually represented as a Prolog program. Thus, unlike learning from interpretations, there is no clear separation of the information concerning the examples and the background knowledge. Furthermore, no assumption is done about the independence of examples, so training examples can be linked to other training examples. This on the one hand makes it possible to learn more complex concepts, like recursive concepts, but considerably increases the computational burden since every time the complete information has to be considered, i.e. *all* training examples together with the background knowledge. For example a coverage test on a single example in principle would have to consider not only the background knowledge but also all other examples.

To place the representation constructed by $\boldsymbol{R}_{\boldsymbol{t}}^{+}(\cdot)$ (Algorithm 2.1) in the context of the representations constructed by the two aforementioned approaches let us first consider a second alternative in acquiring the complete information related with an instance R_{i_j} that avoids the use of recursion. Let us define the function set' as

$$set'(R_{i_j}) = \{R_{j_m} \in R_j \ : \ \exists_{R_j \in \mathcal{R}}, \ \exists_{A_l \in \mathcal{I}_{k,R_i}}, \ \exists_{A_k \in \mathcal{I}_{k,R_j}}, \ R_{j_m}.A_k = R_{i_j}.A_l\}$$

This function will return the set of instances found in any relation of the relational database schema that have a key attribute that shares a value with one of the key attributes of R_{i_j}. The produced set is the set of instances associated with R_{i_j} at level one. To get all instances associated with R_{i_j} in a depth d we simply have to apply the set function to all the instances retrieved at the $d-1$ depth. To get the complete set of instances we simply get the union of all sets for each depth up to the maximum depth (Algorithm 2.6, function $\boldsymbol{R}_{\boldsymbol{f}}^{+}(\cdot)$). The union guarantees that each instance appears at most once. We will first examine the similarities and differences of the example representation constructed by $\boldsymbol{R}_{\boldsymbol{f}}^{+}(\cdot)$ with respect to existing representational paradigms in relational learning and then consider in more detail the differences of $\boldsymbol{R}_{\boldsymbol{t}}^{+}(\cdot)$ from $\boldsymbol{R}_{\boldsymbol{f}}^{+}(\cdot)$.

Algorithm 2.6 is equivalent to the *INTERPRETATIONS* algorithm given by Blockeel (1998) in order to retrieve the interpretation of an instance from the relations of a relational database[9] in the context of learning from interpreta-

[9]Excluding these relations that are part of the background knowledge.

Algorithm 2.6 Retrieving a flat description of a relational instance.

1: $\boldsymbol{R_f^+(R_{i_j})}$
2: // S the complete set of instances associated with R_{i_j}
3: // S' the set of instances associated with R_{i_j} at level d
4: $S = S' = R_{i_j}$
5: **for** $d = 1$ to $MAX - DEPTH$ **do**
6: $\quad S' = \cup_{x \in S'} set'(x)$
7: $\quad S = S \cup S'$
8: **end for**
9: **return** S

tions. In fact, the set of instances returned by $\boldsymbol{R_f^+(R_{i_j})}$ is the interpretation that corresponds to R_{i_j}, including also the related part of the background knowledge since we make no distinction between relations that belong or do not belong to the background knowledge. This is exactly the minimal Herbrand model of $e \cap B$ as it is used in learning from interpretations. There is however an important difference with respect to learning from interpretations: we do not make any assumption about the independence of learning examples. The collected set of instances describing a given learning example, R_{i_j}, could very well incorporate information about other learning examples with which R_{i_j} is associated. In that sense the representation is closer to the representation used in learning from entailment where learning examples can be described in terms of their associations with other learning examples and their properties, but unlike learning from entailment learning operators are not going to be applied to the complete database but only to that part which is relevant to R_{i_j} and is contained in $\boldsymbol{R_f^+(R_{i_j})}$.

The representations constructed by $\boldsymbol{R_t^+(R_{i_j})}$ and $\boldsymbol{R_f^+(R_{i_j})}$ contain exactly the same set of instances; the union of all instances found in the $\boldsymbol{R_t^+(R_{i_j})}$ gives us $\boldsymbol{R_f^+(R_{i_j})}$. Nevertheless, contrary to the flat representation assumed by $\boldsymbol{R_f^+(R_{i_j})}$, $\boldsymbol{R_t^+(R_{i_j})}$ provides a structured representation in which an instance of a relation can be present more than once inside the tree, the only restriction is that it can appear at most once in a given path from the root to a leaf. The multiple appearances are not redundant but describe different local aspects of the relational instance with each appearance occurring in a different local context. The advantage of the tree representation is that it provides readily information about the structural properties of the learning example, i.e. how the various tuples-ground facts that constitute a relational instance are related to each other.

2.6 Conclusions

We proposed a novel and general representational formalism that builds directly over the concepts of (extended) relational algebra, unlike existing work which was mainly logic programming oriented. Our algorithm works in a recursive

manner traversing the relations of the relational schema in order to gather the relevant information. The recursion among the different relations is guided by the concept of links which are based on the notion of foreign keys. Foreign keys provide a natural and intuitive way to provide a declarative bias and render unnecessary type definitions extensively used in inductive logic programming. At each moment they provide direct access to the relevant information providing increased efficiency. The system is directly operational on any learning problem represented by means of a classical relational database, without any need for conversion as it is the case with almost all first order based systems. The algorithmic concepts are expressed in terms of relational algebra terminology bringing the underlying notions much closer to the intuition of the database community thus increasing the potential of wider use by less expert users. Finally, the relational representation has also the advantage of having as a special case the typical propositional representation, which simplifies the design of a relational algorithm, naturally encompassing an existing method operating on a single table.

It is advantageous to consider relational instances as composite objects which can be decomposed into simpler sub-parts. These sub-objects are further decomposed until non-decomposable (primitive) objects such as numbers or nominal attributes are encountered. The non-primitive building blocks are tuples, sets and lists. More complex objects such as trees and graphs can be also represented, however, the latter only through various approximations.

This modular representation of complex objects is very intuitive and considerably simplifies the design of learning algorithms operating over composite structures, since various (parameterized) data mining operators can be defined as a combination of operators defined on the building blocks of learning instances. More precisely, the data mining operators we consider in this study are defined in a declarative manner where the practitioner first defines the different data types which constitute the learning examples, and then declares which operator should be used for each data type. In general, there is a number of different data mining operators, each one with different semantics, and it is up to the analyst to declare which specific operator should be used for a given data type. It is even possible to define different operators for objects of the same general type e.g. sets of lists, and sets of tuples, etc. Various data mining operators will be the main subject of the subsequent chapters. In particular, in Chapter 3 we discuss distance operators, in Chapter 4 we focus on kernels, and finally in Chapter 5 we consider adaptive operators.

Chapter 3

Distances

Distance-based learning is one of the oldest, yet surprisingly effective paradigm in the fields of machine learning and data mining (Duda and Hart, 1973). When combined with the available background knowledge, distance-based methods have advanced the state-of-the-art in various domains (Globerson and Roweis, 2006; Wettschereck et al., 1997; Saul et al., 2006).

It is clear that the choice of a distance measure is crucial for the successful applications of distance-based algorithms, rendering the learning problems easier to solve. Depending on how training instances are represented, defining a distance measure which is able to accurately and naturally model the underlying semantics of the data is a challenging task. In most of the existing distance-based relational systems the representation of learning instances is based on logic programming approach typically employed within the inductive logic programming community (Horváth et al., 2001; Ramon and Bruynooghe, 1998). More recently, the focus has turned to special types of complex objects, namely specific topological structures such as general graphs (Washio and Motoda, 2003), or specific types of graphs such as sequences (Durbin et al., 1999), trees (Bille, 2005) and sets (Eiter and Mannila, 1997; Tatti, 2007; Woźnica et al., 2006a).

Almost all the existing relational distance-based approaches from the literature are constrained to a single monolithic type of a complex distance measure. Nevertheless, it is obvious that there is no single distance measure that is overall better than any other for all types of problems. Typically, a practitioner should consider different distances to find the one that best matches the problem requirements. Thus one can imagine that the final relational distance measure is derived by a combination of distance measures defined over the sub-parts of composite objects. To make things even more complicated it can be that different constituent sub-objects of the same general data type long for different distance measures. None of the existing relational distance-based systems offers this type of flexibility.

In this chapter we address the above limitations and define various distance operators over the relational data representation presented in Chapter 2. The resulting distances are based on decompositions of complex structures into sub-

parts of various types that are compared via appropriate distance functions. There are a number of different distances which are associated with each basic data type, each one with different semantics, and it is up to the analyst to declare which specific distance should be used for a given data type. It is even possible to define different operators for objects of the same general type e.g. sets of primitive data types, and sets of tuples, etc. The final distance over the full complex learning instances is given as a recursive combination of distance functions assigned to the sub-structures which constitute the learning instances. There are as many different instantiations of the overall distances as there are combinations of distances over the components of the full complex learning instances.

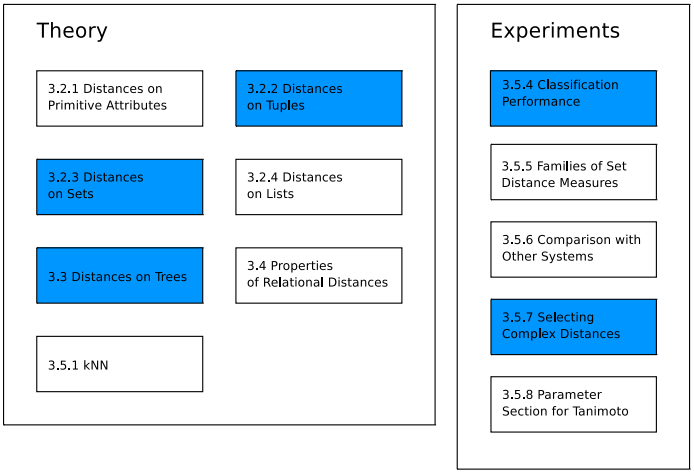


Figure 3.1: Schematic structure of Chapter 3. The most important sections are highlighted.

The remainder of this chapter is organized as follows. In Section 3.1 we briefly review some of the terminology and the definitions used to characterize the different distances explored in this study. Then, in Sections 3.2.1–3.2.4, we formally define distances over the different building blocks of relational instances as presented in Chapter 2. More precisely, in Section 3.2.1 we present distances over objects of elementary types while in Sections 3.2.2 , 3.2.3 and 3.2.4 we define distances on tuples, sets and lists, respectively. Section 3.3 describes distance measures on trees. In the second part of the this chapter (Section 3.5) we present the experimental results where different distance measures are used in a common framework exploiting the k-Nearest Neighbor algorithm, defined in Section 3.5.1. In Section 3.6 we place our relational distances in the context of related work. Finally, we conclude with Section 3.7.

Most of the important theoretical work is presented in Section 3.2.2 where we describe (parameterized) distances on tuples which will be widely exploited

in Chapter 5, in Section 3.2.3 where we introduce set distances based on mappings (used throughout the rest of the thesis), and in Section 3.3 where we describe distances on trees. The most important experimental results are presented in Sections 3.5.4 and 3.5.7. In these experiments we respectively analyse the performance of the above distances and examine a learner which is based on cross-validated distance selection; the latter will form a baseline method for the algorithms from Chapter 5. The organization of this chapter is schematically presented in Figure 3.1.

3.1 Preliminaries

Before going to the description of different distance measures we will briefly review some of the terminology and the definitions used to characterize the different distances explored in this study. In the rest of this chapter we assume that $\mathcal{X}$ is a set whose elements are general complex objects, and not necessarily a standard numerical space.

Definition 3.1 (Dissimilarity)**.** A function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_0^+$ is a *dissimilarity function* on a nonempty set $\mathcal{X}$ iff it is *reflexive*, i.e.

$$\forall x \in \mathcal{X} \ : \ d(x,x) = 0$$

Definition 3.2 (Distance)**.** A function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_0^+$ is a *distance* iff it is a dissimilarity function and it is *symmetric*, i.e.

$$\forall x, y \in \mathcal{X} \ : \ d(x,y) = d(y,x)$$

Definition 3.3 (Metric)**.** A function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_0^+$ is a *metric* iff it is a distance function and

- it is *strict* i.e.
$$\forall x, y \in \mathcal{X} \ : \ d(x,y) = 0 \Rightarrow x = y$$
- and satisfies the *triangle inequality* i.e.
$$\forall x, y, z \in \mathcal{X} \ : \ d(x,z) \leqslant d(x,y) + d(y,z)$$

Definition 3.4 (Pseudo-metric)**.** A function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_0^+$ is a *pseudo-metric* iff it is reflexive, symmetric and satisfies the triangle inequality.

Definition 3.5 (Semi-metric)**.** A function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_0^+$ is a *semi-metric* iff it is reflexive, symmetric and strict.

Any distance measure d that is not reflexive can be made reflexive by:

$$d(x,y) := d(x,y) - \frac{1}{2}(d(x,x) + d(y,y)) \tag{3.1}$$

Additionally any distance measure d that is not symmetric can be symmetrized by

$$d(x,y) := \frac{1}{2}(d(x,y) + d(y,x)) \tag{3.2}$$

It should be noted that the strict and reflexive properties are sometimes identified collectively as reflexivity

$$\forall x, y \in \mathcal{X} \ : \ d(x,y) = 0 \text{ iff } x = y.$$

For reasons of readability we will call all of the above *distance measures*; when it is necessary we will make clear whether a distance measure is a dissimilarity, distance, metric etc.

In many cases it is desirable to upper bound the distance measures.

Definition 3.6 (r-bounded distance)**.** A distance measure $d \ : \ \mathcal{X} \times \mathcal{X} \to \mathbb{R}_0^+$ is r-bounded ($r \in \mathbb{R}^+$) iff

$$\forall x, y \in \mathcal{X} \ : \ d(x,y) \leqslant r.$$

In this work we will consider the 1-bounded distance measures. It should be noted that we can always compute the so called *trivial* distance measure between composite objects.

Definition 3.7 (Trivial Distance)**.** A distance measure $d \ : \ \mathcal{X} \times \mathcal{X} \to \mathbb{R}_0^+$ is said to be *trivial* iff

$$\forall x, y \in \mathcal{X} \ : \ d(x,y) = \begin{cases} 0 & \text{if } x = y \\ 1 & \text{otherwise.} \end{cases} \tag{3.3}$$

It is straightforward to show the following proposition.

Proposition 3.1. *The trivial distance measure is a metric.*

While the trivial metric can be computed efficiently for many types of complex objects it is of little practical interest and we will be mainly interested in distance measures which assign dissimilarities between two objects in a smooth way.

We conclude this section by the definition of a concept of a distance measure which is isometric to an L_2-norm. This distance measure will be used in Chapter 4.

Definition 3.8 (Distance Isometric to L_2 Norm)**.** We call a distance measure $d \ : \ \mathcal{X} \times \mathcal{X} \to \mathbb{R}_0^+$ isometric to an L_2-norm iff the data can be embedded in a Hilbert space $\mathcal{H}$ by $\phi \ : \ \mathcal{X} \to \mathcal{H}$ such that

$$d(x,y) = \|\phi(x) - \phi(y)\|_{\mathcal{H}}.$$

We will sometimes call a distance measure isometric to an L^2-norm a *Hilbertian metric.*

3.2 Distances over Relational Building Blocks

After defining the basic concepts we move to definitions of various distance operators on building blocks of relational instances, as defined in Section 2.1. In particular, in Section 3.2.1 we present elementary distances, while in Sections 3.2.2, 3.2.3 and 3.2.4 we define various distances over tuples, sets and lists, respectively.

3.2.1 Distances on Primitive Attributes

In this section we define the distance measures for the most basic objects, i.e. numerical and symbolic data types.

We start with numerical data types. Let x, y be objects of numeric data type with range $[a, b]$ $(a < b)$, i.e. $type(x) = type(y) = numeric(a, b)$. The distance between objects of numeric data type we used in this study is defined as follows.

Definition 3.9 (Distance between Numerical Values)**.** The distance measure between two numerical values $x, y \in \mathbb{R}$ is defined as

$$d_{num}(x, y) = \frac{|x - y|}{b - a} \tag{3.4}$$

It is straightforward to show that the d_{num} distance measure is a metric.

Proposition 3.2. *The d_{num} distance measure is a metric on $\mathbb{R}$.*

Let $\mathcal{X}$ be a finite set of symbolic values and let x, y be objects of symbolic data type, i.e. $type(x) = type(y) = symbolic(S)$ for some finite set S of identifiers. The most widely used distance measure over symbolic values is the δ *distance* which is an instantiation of the trivial metric of Definition 3.7. This is the distance we will use in this work.

Definition 3.10 (δ Distance)**.** For two values $x, y \in \mathcal{X}$ we define the δ *Distance* as

$$d_\delta(x, y) = \begin{cases} 0 & \text{if } x = y \\ 1 & \text{otherwise.} \end{cases} \tag{3.5}$$

More generally, when not all elements in the set $\mathcal{X}$ play the same role, one can also define a $|\mathcal{X}| \times |\mathcal{X}|$ matrix that defines the distance between different elements (Ramon, 2002). Such approach is frequently used e.g. for computing distances between amino acids in proteins' sequences (Durbin et al., 1999) where the actual distance between any two elements is derived from how often different amino acids replace other amino acids in evolution. Obviously by setting all the diagonal elements of such matrix to 1 and off-diagonal elements to 0 we get the distance measure of Equation 3.5.

3.2.2 Distances on Tuples

In this section we will define various distance measures defined over the tuple data type. One of the most widely used distance on tuples is the *Minkowski distance* measure.

Definition 3.11 (Minkowski Distance)**.** Let $\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_m$ be sets on which the corresponding distance measures $d_1, d_2, \ldots, d_m$ are defined, i.e. $d_i : \mathcal{X}_i \times \mathcal{X}_i \to \mathbb{R}_0^+$ for $i = 1, \ldots, m$. Moreover, let $\boldsymbol{\mathcal{X}} = \mathcal{X}_1 \times \mathcal{X}_2 \times \ldots \times \mathcal{X}_m$ be the set of all tuples $(x_1, x_2, \ldots, x_m)$ such that $x_i \in \mathcal{X}_i$. Then the distance function $d_{tuple,p} : \boldsymbol{\mathcal{X}} \times \boldsymbol{\mathcal{X}} \to \mathbb{R}_0^+$ defined as

$$d_{tuple,p}(\boldsymbol{x}, \boldsymbol{y}) = \left(\sum_{i=1}^{m} d_i^p(x_i, y_i) \right)^{\frac{1}{p}} \tag{3.6}$$

is called a *Minkowski distance* measure, $p \in \mathbb{N}$.

The following proposition holds.

Proposition 3.3. *The $d_{tuple,p}$ distance measure is a metric on $\boldsymbol{\mathcal{X}} = \mathcal{X}_1 \times \mathcal{X}_2 \times \cdots \times \mathcal{X}_m$ provided all $d_1, d_2, \ldots, d_m$ are metrics.*

The Minkowski distance measure is also referred to as the L_p norm. The L_1 norm is sometimes called the *Manhattan* or *city block* distance. In this study we will only focus on the L_2 norm which is the well-known *Euclidean metric*.

The generalized version of Euclidean metric is the *Mahalanobis distance*. In order to define this distance measure we need to introduce a concept of a *positive (semi-)definite matrix*.

Definition 3.12 (Positive Semi-Definite Matrix)**.** A real matrix $\boldsymbol{A} \in \mathbb{R}^{m \times m}$ is called *positive semi-definite* (PSD), denoted as $\boldsymbol{A} \succeq 0$, iff

$$\forall \boldsymbol{x} \in \mathbb{R}^m \ : \ \boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x} \geqslant 0$$

Definition 3.13 (Positive Definite Matrix)**.** A real matrix $\boldsymbol{A} \in \mathbb{R}^{m \times m}$ is called *positive definite* (PD), denoted as $\boldsymbol{A} \succ 0$, iff

$$\forall \boldsymbol{x} \in \mathbb{R}^m \ : \ \boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x} > 0$$

Definition 3.14 (Mahalanobis Distance)**.** Let $\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_m$ be sets on which the corresponding distance measures $d_1, d_2, \ldots, d_m$ are defined, i.e. $d_i : \mathcal{X}_i \times \mathcal{X}_i \to \mathbb{R}_0^+$ and let $\boldsymbol{\mathcal{X}} = \mathcal{X}_1 \times \mathcal{X}_2 \times \cdots \times \mathcal{X}_m$ be the set of all vectors $(x_1, x_2, \ldots, x_m)$ such that $x_i \in \mathcal{X}_i$. Moreover, let

$$\boldsymbol{d}(\boldsymbol{x}, \boldsymbol{y}) = [d_1(x_1, y_1), d_2(x_2, y_2), \ldots, d_m(x_m, y_m)]^T$$

and let $\boldsymbol{A}$ be a $m \times m$ positive semi-definite matrix. Then the distance function $d_{tuple,\boldsymbol{A}} : \boldsymbol{\mathcal{X}} \times \boldsymbol{\mathcal{X}} \to \mathbb{R}_0^+$ defined as

$$d_{tuple,\boldsymbol{A}}(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\boldsymbol{d}(\boldsymbol{x}, \boldsymbol{y})^T \boldsymbol{A} \ \boldsymbol{d}(\boldsymbol{x}, \boldsymbol{y})} \tag{3.7}$$

is called a *Mahalanobis distance* measure.

It should be noted that the Mahalanobis distance can be re-parameterized as:

$$d_{tuple,\boldsymbol{W}}(\boldsymbol{x},\boldsymbol{y}) = \sqrt{\boldsymbol{d}(\boldsymbol{x},\boldsymbol{y})^T \boldsymbol{W}^T \boldsymbol{W}\ \boldsymbol{d}(\boldsymbol{x},\boldsymbol{y})} \tag{3.8}$$

where $\boldsymbol{A} = \boldsymbol{W}^T\boldsymbol{W}$ and $\boldsymbol{W}$ is a $m \times m$ matrix (not necessarily PSD). For any $\boldsymbol{W}$ we have $\boldsymbol{A} = \boldsymbol{W}^T\boldsymbol{W} \succeq 0$. The Mahalanobis distance (with both re-parameterizations) will be used in Chapter 5. The following proposition holds.

Proposition 3.4. *$d_{tuples,\boldsymbol{A}}$ is a pseudo-metric on $\boldsymbol{\mathcal{X}} = \mathcal{X}_1 \times \mathcal{X}_2 \times \cdots \times \mathcal{X}_m$ provided that all $d_1, d_2, \ldots, d_m$ are pseudo-metrics and $\boldsymbol{A} \succeq 0$. If all $d_1, d_2, \ldots, d_m$ are metrics and $\boldsymbol{A} \succ 0$ then $d_{tuple,\boldsymbol{A}}$ is a metric.*

It is easy to show that by restricting $\boldsymbol{A}$ to a diagonal matrix, i.e. $\boldsymbol{A} = \boldsymbol{I}$, the Mahalanobis distance reduces to the Euclidean distance. If matrix $\boldsymbol{A}$ is diagonal, i.e. $\boldsymbol{A} = diag(a_1, \ldots, a_m)$ and $\mathcal{X}_i = \mathbb{R}$ for $i = 1, \ldots, m$, then the resulting distance measure amounts to a weighted Euclidean distance

$$d_{tuple,diag(a_1,\ldots,a_m)}(\boldsymbol{x},\boldsymbol{y}) = \sqrt{\sum_{i=1}^{m} a_i d_i^2(x_i, y_i)}$$

3.2.3 Distances on Sets

The issue that arises when working with sets of objects is how one can use the distance measures defined on $\mathcal{X}$ in order to define distance measures on the power set $2^{\mathcal{X}}$ of $\mathcal{X}$ and under what conditions the set distance measure is a distance or a metric function. For the moment we only focus on standard sets (i.e. we do not allow elements to appear multiple times); multi-sets will be considered later. A number of different measures have been proposed in the literature for defining distances between sets of objects. We will briefly present some of them.

Consider two non-empty and finite sets $A = \{a\} \subseteq \mathcal{X}$ and $B = \{b\} \subseteq \mathcal{X}$. Let d be a distance measure defined on $\mathcal{X}$. The set distance measure d_{set} defined on $2^{\mathcal{X}}$ as:

$$d_{set}\ :\ 2^{\mathcal{X}} \times 2^{\mathcal{X}} \to \mathbb{R}_0^+,\ d_{set}(A,B) = f(\{d(a,b)|(a,b) \in A \times B\}) \tag{3.9}$$

is some function, f, of the set of pairwise distances, $d(a,b)$, of the set of all pairs $(a,b) \in A \times B$. Different definitions of f functions give rise to different set distance measures. Within this framework we can define the following set distance measures.

Definition 3.15 (Single Linkage)**.** The *Single Linkage* set distance measure is defined as the minimum distance of all pairwise distances (Hastie et al., 2001)

$$d_{SL}(A,B) = \min_{(a,b)\in A\times B} \{d(a,b)\} \tag{3.10}$$

It is not a metric or a pseudo-metric even if the underlying distance measure d is a metric because it does not satisfy the strict and triangle inequality properties.

Example 3.1. A counter example for the strict and triangle inequality properties is to consider $A = \{a, b\}, B = \{c, d\}, C = \{a, c\}$ with d being the δ metric from Equation 3.5. We have $d_{SL}(A, C) = 0$, but $A \neq C$. Moreover $1 = d_{SL}(A, B) > d_{SL}(A, C) + d_{SL}(B, C) = 0$.

As a result the following proposition holds.

Proposition 3.5. *If the function d is at least a distance function then d_{SL} is also a distance function.*

Definition 3.16 (Complete Linkage)**.** The *Complete Linkage* set distance measure is defined as the maximum of all pairwise distances (Hastie et al., 2001)

$$d_{CL}(A, B) = \max_{(a,b)\in A\times B} \{d(a, b)\} \tag{3.11}$$

The d_{CL} distance measure is not even a dissimilarity function since it does not satisfy the reflexive property.

Definition 3.17 (Average Linkage)**.** The *Average Linkage* set distance is defined as the sum of all pairwise distances (Hastie et al., 2001)

$$d_{AL}(A, B) = \sum_{(a,b)\in A\times B} \{d(a, b)\} \tag{3.12}$$

Similarly to d_{CL}, d_{AL} does not satisfy the reflexive property and hence is not even a dissimilarity function. All the above set distance measures are widely used in computing set distances in clustering.

Definition 3.18 (Sum of Minimum Distances)**.** The *Sum of Minimum Distances* set distance measure (Eiter and Mannila, 1997) is defined as the sum of the minimum distances of the elements of the first set to the elements of the second set and vice versa. More formally:

$$d_{SMD}(A, B) = \sum_{a\in A} \min_{b\in B}\{d(a, b)\} + \sum_{b\in B} \min_{a\in A}\{d(b, a)\} \tag{3.13}$$

It should be noted that the formulation presented above differs from the one given in (Eiter and Mannila, 1997) where this set distance measure is normalized as $d_{SMD}(A, B) := \frac{d_{SMD}(A,B)}{2}$. The normalization of this set distance measure is discussed in Section 3.2.5. The d_{SMD} distance measure is not a metric even if the underlying distance measure d is a metric because it fails to satisfy the triangle inequality property.

Example 3.2. This example is taken from (Eiter and Mannila, 1997). Consider $A = \{a, c\}$, $B = \{b\}$ and $C = \{d\}$ as in Figure 3.2 together with the Manhattan metric from Equation 3.20 (i.e. $p = 1$). If $d(a, b) = d(b, c) = d(b, d) = 1$ then $d_{SMD}(A, B) = 3, d_{SMD}(A, C) = 6$ and $d_{SMD}(B, C) = 2$, hence $d_{SMD}(A, C) > d_{SMD}(A, B) + d_{SMD}(B, C)$.

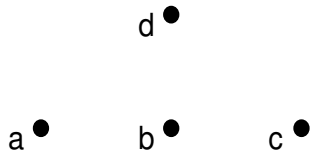


Figure 3.2: Triangle inequality violations for $A = \{a, c\}$, $B = \{b\}$ and $C = \{d\}$.

However, it it easy to show the following proposition.

Proposition 3.6. *If d is at least a semi-metric then d_{SMD} is also a semi-metric.*

The *Hausdorff* set distance measure was discussed in (Eiter and Mannila, 1997; Klein and Thompson, 1984) and is one of the best known distances measures between sets.

Definition 3.19 (Hausdorff)**.** The *Hausdorff* set distance measure is defined as:

$$d_H(A, B) = \max\left(\max_{a \in A}\{\min_{b \in B}\{d(a, b)\}\}, \max_{b \in B}\{\min_{a \in A}\{d(b, a)\}\}\right) \tag{3.14}$$

Intuitively, if the Hausdorff distance measure between sets A and B is d_H then every point of A is within distance d_H of at least one point of B and vice versa. In other words, the notion of resemblance encoded by this distance measure is that each member of A is near some member of B and vice versa (Huttenlocher et al., 1993). Following (Dugundji, 1966) the following theorem holds.

Theorem 3.1. *If d is a metric then d_H is also a metric.*

Definition 3.20 (RIBL)**.** The *RIBL* set distance measure is defined as the sum of the minimum distances of the elements of the smaller set to the elements of the larger, normalized by the cardinality of the smaller set (Emde and Wettschereck, 1996; Horváth et al., 2001):

$$d_{RIBL}(A, B) = \begin{cases} \frac{\sum_{a \in A} \min_{b \in B}\{d(a,b)\}}{|A|} & \text{if } |A| < |B| \\ \frac{\sum_{b \in B} \min_{a \in A}\{d(a,b)\}}{|B|} & \text{otherwise} \end{cases} \tag{3.15}$$

From the definition it is clear that d_{RIBL} is not symmetric. Moreover, it does not satisfy the triangle inequality property.

Example 3.3. Consider $A = \{0, 1\}, B = \{-2, 1\}$ with the standard metric over numbers from Equation 3.4 (to simplify the calculations we assume that it is not normalized by $b - a$). We have $d_{RIBL}(A, B) = \frac{2}{2} = 1$ but $d_{RIBL}(B, A) = \frac{1}{2}$. A simple counter example for the triangle inequality is to consider $A = \{a\}, B = \{b\}, C = \{a, b\}$ with the δ metric. We have $1 = d_{RIBL}(A, B) > d_{RIBL}(A, C) + d_{RIBL}(C, B) = 0$.

Proposition 3.7. *The d_{RIBL} distance measure is a dissimilarity measure provided that d is a dissimilarity measure.*

The d_{RIBL} distance measure can be symmetrized by Equation 3.2. However, the resulting distance will be equivalent (up to a normalization term) to d_{SMD}. As a result, in the remainder of this study we will consider the original version of d_{RIBL}.

So far the distance measures over sets that we have presented are relatively simple ones whose computation is straightforward if one has computed all the pairwise distances among all the pairs of elements defined from the two sets. Another family of more elaborate distance measures is based on the definition of a set of relations $R = \{R_i | R_i \subseteq A \times B\}$ between the two sets. The computation of the distance measure will be based on that $R_i \in R$ that minimizes a distance measure computed on the elements that are part of the relation R_i. We denote $|R_i|$ as the number of elements in R_i.

The relations R considered in this work are *surjections*, *fair surjections*, *linkings* and *matchings*. The extra restrictions of the first two relations is that they define mappings of the set with the larger cardinality to the set of the smaller cardinality.

Definition 3.21 (Surjection)**.** A relation $R_i \subseteq A \times B$ is a *surjection* iff

- $\forall_{(a,b),(c,d) \in R_i} \ : \ (a = b \Rightarrow c = d)$
- $\forall_{b \in B} \exists_{a \in A} \ : \ (a, b) \in R_i$

Definition 3.22 (Fair Sujrection)**.** A surjection $R_i \subseteq A \times B$ is *fair* iff

$$\forall_{b,c \in B} \ : \ ||R^{-1}\{b\}| - |R^{-1}\{c\}|| \leqslant 1$$

In other words a surjection is fair if it maps as evenly as possible the elements of the larger set to the elements of the smaller set.

Definition 3.23 (Linking)**.** A relation $R_i \subseteq A \times B$ is a *linking* iff

- $\forall_{a \in A} \exists_{b \in B} \ : \ (a, b) \in R_i$
- $\forall_{b \in B} \exists_{a \in A} \ : \ (a, b) \in R_i$

In other words a linking is a mapping of one set to the other where all elements of each set participate in at least one pair of the mapping.

Definition 3.24 (Matching)**.** A relation $R_i \subseteq A \times B$ is a *matching* iff

$$\forall_{(a,b),(c,d) \in R_i} \ : \ (a = b \Leftrightarrow c = d)$$

In a matching each element of the two sets is associated with at *most* one element of the other set.

The general form of the set distance measures based on the above families of binary relations can be written as (Ramon and Bruynooghe, 2001)

$$d_{set}(A,B) = \min_{R_i \in R}\Big(\sum_{(a,b)\in R} d(a,b) + (|B - R_i(A)| + |A - R_i^{-1}(B)|) \times \frac{M}{2}\Big) \quad (3.16)$$

where R can be the set of all possible surjections (then d_{set} is denoted as d_S), fair surjections (d_{FS}), linkings (d_L) or matchings (d_M). Moreover, M is the maximum possible distance between two elements. What the second term of the sum in Equation 3.16 actually does is to add an M/2 penalty for these elements of the A and B that do not participate in the relation R_i. In d_S, d_L, d_{FS} the second term of the sum vanishes because all elements of the two sets participate in the relation R_i. We mention here that in Section 3.3 we will exploit d_M as a building block for defining distance on trees.

An algorithm for computing d_S, d_L is given in (Eiter and Mannila, 1997) and is based on graph theoretical concepts, more precisely on minimum weight perfect matching in bipartite graphs. For d_{FS} and d_M the corresponding algorithms are presented in (Eiter and Mannila, 1997) and (Ramon and Bruynooghe, 2001), respectively, and are based on flow networks and the minimum weight maximum flow.

Proposition 3.8. *The d_S, d_{FS}, d_L distance measures do not satisfy the triangle inequality so they are semi-metrics. The counter example for the triangle inequality is the same as the one presented in Example 3.2.*

Proposition 3.9. *The d_M distance measure is a metric provided that d is a metric.*

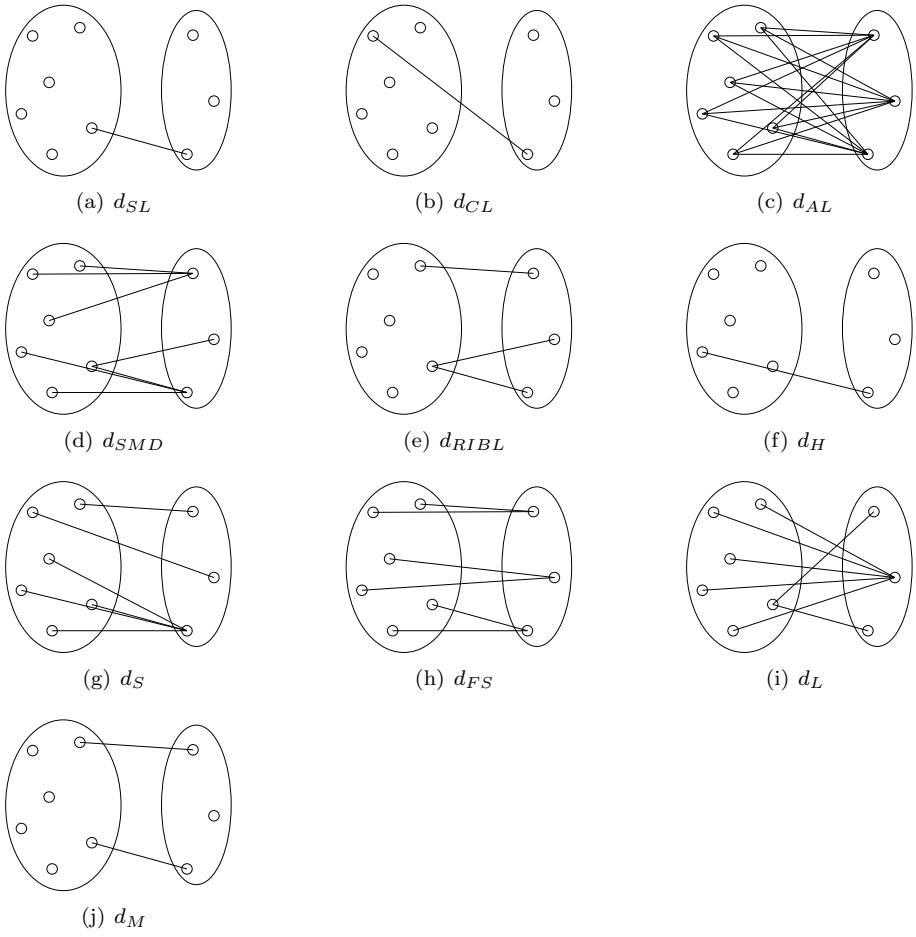
Proof. The proof is given in (Ramon, 2002). □

All of the above set distance measures reduce to $d(a,b)$ in the trivial case that $A = \{a\}$ and $B = \{b\}$. This is a necessary property if we want the set representation to have as a special case the typical attribute-single-value representation. Additionally all of the above set distance measures, with the exception of the d_M set distance measure, can be in also written as

$$d_{set}(A,B) = \sum_{(a,b)\in F} d(a,b) \quad (3.17)$$

i.e. it is the sum of pairwise distances over specific pairs of elements from the two sets defined by $F \subseteq A \times B$. A simple visualization of specific pairs of elements given by F for the considered set distance measures is presented in Figure 3.3.

In this work we also included a set distance measure which can not be written as in Equation 3.17. The *Tanimoto* set distance measure is discussed in (Duda et al., 2001) and is used when we do not have a notion of graded distance or similarity between the elements of two sets, i.e. two elements are simply the same or different.



(a) d_{SL} (b) d_{CL} (c) d_{AL}

(d) d_{SMD} (e) d_{RIBL} (f) d_H

(g) d_S (h) d_{FS} (i) d_L

(j) d_M

Figure 3.3: Specific pairs of elements defined by F for the considered set distance measures.

Distance Measure	Reflexive	Symmetric	Strict	Triangle	Type
d_{SL}	+	+	-	-	distance
d_{CL}	-	+	-	-	-
d_{AL}	-	+	-	-	-
d_{SMD}	+	+	-	-	semi-metric
d_H	+	+	+	+	metric
d_{RIBL}	+	-	-	-	dissimilarity
d_T	+	+	-	-	distance
d_S	+	+	+	-	semi-metric
d_{FS}	+	+	+	-	semi-metric
d_L	+	+	+	-	semi-metric
d_M	+	+	+	+	metric

Table 3.1: Characterization of the different distance measures according to the properties they satisfy.

Definition 3.25 (Tanimoto)**.** The *Tanimoto* set distance measure is defined as

$$d_T(A,B) = \frac{|A| + |B| - 2|A \cap B|}{|A| + |B| - |A \cap B|} \tag{3.18}$$

In order to be able to deal with graded similarities we defined an extension of d_T. Two elements $a \in A$ and $b \in B$ will be considered identical if $d(a,b) \leqslant \theta$ where θ is a user specified threshold parameter. Under this loose definition of identity it is now possible to compute the cardinality of $A \cap B$ with the constraint that each element of a set can be only matched once. This loose definition of identity has as a result that the strictness property and the triangle inequality are no longer satisfied.

Example 3.4. Consider the sets from Example 3.2 and set $\theta = 1$. In this case $d_T(A,B) = \frac{2+1-2\cdot 1}{2+1-1} = \frac{1}{2}$, $d_T(A,C) = \frac{2+1-2\cdot 0}{2+1-0} = 1$ and $d_T(B,C) = \frac{1+1-2\cdot 1}{1+1-1} = 0$, but $B \neq C$. Moreover, $d_T(A,C) > d_T(A,B) + d_T(B,C)$.

It is straightforward to show the following proposition.

Proposition 3.10. *The d_T set distance measure is a distance provided d is a distance.*

The general properties of distance measures are presented in Table 3.1. It should be mentioned that these properties are only guaranteed to hold in cases where the objects are sets and not multisets. For example, the strictness property, even if it is satisfied for some set distance measures, in general it will not hold for multisets. On the other hand, it is straightforward to show that the reflexive and symmetric properties are not influenced if we allow repetitions of the same elements. Finally, it is an open question how the triangle inequality property is affected by this change.

Each of the presented distance measures imposes different semantics on what is important in determining the distance between two sets. For example in d_{SL}

it is only the two most similar elements that determine the distance between the two sets, while in d_{CL} it is exactly the opposite, i.e. the two most dissimilar objects determine the set distance. The main limitation that one could see in d_{SL}, d_{CL} and d_H is that they do not take into account the complete information provided by the whole sets but rather focus on a specific pair of elements. A fact that can be quite problematic in the presence of noise or outliers within the sets. d_{RIBL} and d_{SMD} do not focus on the distance of a single pair of elements but on the set of minimum distances of the elements of one set to the elements of the other set providing a more global measure of how similar are the two sets with respect to their most similar elements. This approach though more global could still be problematic if there is an outlier in one of the sets, let's say in set A, that is much closer than all other elements of set A to the elements of the B set. In that case it will be the minimum distances from that element that will mainly determine the final distance. The problem is more acute for d_{RIBL} since it is not symmetric in its use of minimal distances with respect to the two sets. For d_{SMD} it is less problematic since it will only dominate one of the two sum terms. d_{AL} tries to take into account the complete available information by averaging overall the pairwise distances. However the main problem of this measure is that it does not satisfy the reflexive property which in fact means that the distance of a set from itself can be, and most often will be, bigger than zero. This can lead to awkward situations where a set is more similar to another set than it is to itself. d_{CL} also exhibits the same type of pathology. The reason for which we included these two measures is that they are extensively used in computing set distances in clustering. The Tanimoto based distance measure is a measure of the degree of overlap between the two sets under the loose definition of identity that we have introduced. It is less sensitive to the problem of outliers since each element of a set can be matched at most once. The relation based measures are also less sensitive to the existence of outliers since they take a more global view by seeking a mapping between the elements of the two sets which uses all the available information given by the sets. In the case of d_L, d_S and d_{FS} each element of the two sets will participate at least once in the relation and the resulting distance will be the minimum computed overall permissible relations. In that way the distance computation is spread across all the elements of the two sets making it less sensitive to the presence of noise or outliers. In the case of d_M it is not required that every element participates in the relation, however even the elements that do not participate are accounted for in the distance computation by the penalty term.

Nothing can be said about the general superiority of one distance measure over another. It all depends on the specific problem application and its semantics which should mainly drive the selection of the appropriate measure. For example in some applications it might be that what is most important is the distance between the two most similar elements of the sets, while for others a more global approach that takes into account all the elements might be required.

Distance Measure	Complexity
d_{SL}	$O(n^2)$
d_{CL}	$O(n^2)$
d_{AL}	$O(n^2)$
d_{SMD}	$O(n^2)$
d_{RIBL}	$O(n^2)$
d_H	$O(n^2)$
d_T	$O(n^3)$
d_S	$O(n^3)$
d_{FS}	$O(n^3)$
d_L	$O(n^3)$
d_M	$O(n^3)$

Table 3.2: Characterization of the different distance measures according to their complexities (n is the set cardinality).

Complexity

The computational complexities of the considered set distance measures are presented in Table 3.2. As expected the simple distance measures have the lowest complexity which is quadratic with respect to cardinalities of the sets. It should be mentioned that although for d_{SL} we used the naive implementation which has complexity of $O(n^2)$, it might be possible to exploit the ideas presented in (Toussaint and Bhattacharya, 1983) and reduce this complexity to $O(n\ log(n))$. Similarly, it is sometimes possible to compute d_H faster than in the naive implementation which has complexity of $O(n^2)$, e.g. the time complexity of d_H between two sets of vertices of convex polygons in $\mathbb{R}^n$ is $O(n)$ (Atallah, 1983). It would be interesting to analyze whether this complexity can be reduced for general sets. The time complexity reported for d_T, which is cubic in the set cardinalities, is the worst time complexity. The reason it is higher than in the simple set distance measures is a result of the fact that each element of a set can be matched at most once. The other, more elaborate set distance measures have higher complexity which is approximately cubic[1] in the number of elements in the sets. The time complexity for d_S, d_{FS} and d_L is reported from (Eiter and Mannila, 1997) whereas the complexity of d_M is taken from (Ramon, 2002).

3.2.4 Distances on Lists

In this work we exploit the alignment-based edit distance (Levenshtein, 1966; Durbin et al., 1999) which is probably the most widely used distance measure for sequences over finite alphabets. In edit distance we are given a set of basic edit operations on sequences (replace, insert, delete) together with a cost function that tells us how expensive each operation is. The edit distance of two sequences is the cost of the lowest cost sequence of such operations that transforms the

[1]The complexity is precisely cubic only for d_S.

first sequence into the second. The edit distance does not only give a distance of the two sequences but it also provides the alignment-matching of the elements of the two sequences.

In order to adapt the edit distance so that it can operate on lists that do not necessarily consist of symbolic objects we simply have to change the cost function that it uses. Replacing the cost function with a distance between general complex objects, the edit distance can then be applied to any list of complex objects. We will give now a more formal presentation of the alignment-based edit distance that we are going to use. Lets suppose we are given two lists $\ell_1 = [\ell_{1_1}, \ldots, \ell_{1_n}]$, $\ell_2 = [\ell_{2_1}, ..., \ell_{2_m}]$, where $\ell_{i_j} \in \mathcal{X}$ and let d be a distance measure over set $\mathcal{X}$.

Definition 3.26 (Alignment)**.** An *alignment* $\mathcal{A}$ of ℓ_1 and ℓ_2 is a set of two sequences ℓ'_1 and ℓ'_2 of equal length, l, $l \geqslant \max(|\ell_1|, |\ell_2|)$, constructed from the initial sequences by insertion of gaps, $-$.

Aligning two elements there are only three possibilities:

- the i-th element of ℓ_1 is aligned to a gap (*insert* operation),
- the i-th element of ℓ_1 is aligned to the k-th element of ℓ_2 (*replace* operation),
- the k-th element of ℓ_2 is aligned to a gap (*delete* operation).

The above operations are called *edit operations* and can be also considered as rewriting rules that transform the list ℓ_1 to ℓ_2. More precisely, for $a, b \in \mathcal{X}$, the insert, replace and delete edit operations correspond to the application of rewriting $- \rightarrow a$, $a \rightarrow b$ and $a \rightarrow -$, respectively, where $-$ is an empty element.

The cost of an alignment is simply the sum of the cost of all operations used to derive the alignment $c(\mathcal{A}) = \sum_{i=1}^{l} c(\ell'_{1_i}, \ell'_{2_i})$ where the cost of the replace operation is $c(x, y) = d(x, y)$, $x, y \in \mathcal{X}$, and the cost of the insert and delete operations is usually assumed to be constant α, i.e., $c(x, -) = c(-, y) = \alpha$. α is also known as the *gap penalty*. If d is a 1 bonded distance measure then α is usually set to 1. The cost of an alignment is hence defined as

$$c(\mathcal{A}) = \sum_{i=1}^{l} d(\ell'_{1_i}, \ell'_{2_i})$$

Now we are in position to define the *edit distance measure*.

Definition 3.27 (Edit Distance)**.** The alignment-based *edit distance measure* (also know as the *Levenshtein distance*), $d_{edit}(\ell_1, \ell_2)$, of two sequences, ℓ_1, ℓ_2, is the minimum cost over all possible alignments of the two sequences

$$d_{edit}(\ell_1, \ell_2) = argmin_{\mathcal{A}} c(\mathcal{A}) \tag{3.19}$$

In other words the *edit distance measure* is equivalent to the cost of the lowest cost sequence of operations that turns the first list into the second. Using

dynamic programming (Durbin et al., 1999; Wagner and Fischer, 1974) it is easy to compute the edit distance with a complexity of $O(nm)$. Moreover, it is possible to show the following proposition.

Proposition 3.11 (Edit Distance)**.** *The $d_{edit}(\ell_1, \ell_2)$ distance measure from Equation 3.19 defined over $\mathcal{L}(\mathcal{X} \cup -)$ is a metric provided d defined on $\mathcal{X}$ is also a metric.*

Proof. All the necessary conditions follow directly from the definition of the edit distance measure. □

3.2.5 Normalization

One problem with most of the distance measures presented in this section is that they are influenced by the number of constituent parts of the composite objects. For example, this will happen with the set distances which are based on the definition of specific types of mappings where the final distance is a sum of distances (Equation 3.17), thus its value will depend on the cardinality of the mapping. This could be problematic also in cases when one set has small cardinality and the other has a large cardinality. Similar arguments also hold for the edit distance measure on lists. Finally, the Minkowski metric is influenced by the number of elements in tuples. In this section we show how we can normalized the above distance measures such that they take values between 0 and 1.

Tuples

In this work we will use the normalized Minkowski metric of Equation 3.11.

Definition 3.28 (Normalized Minkowski Distance)**.** Let all notations be the same as in Definition 3.11. The normalized *Minkowski distance* measure is defined as

$$d_{tuple,p}(\boldsymbol{x}, \boldsymbol{y}) = \left(\frac{\sum_{i=1}^{m} d_i^p(x_i, y_i)}{m} \right)^{\frac{1}{p}} \tag{3.20}$$

The above normalized distance measures have the same properties as their non-normalized versions from Section 3.2.2. In particular, the Normalized Minkowski Distance is a metric.

Sets

Of the presented set distance measures the ones that are not normalized are d_{AL}, d_{SMD}, d_S, d_L, d_{FS} and d_M. As in Equation 3.17 the normalized version of the above set distance measures can be written as

$$d_{set}(A, B) = \frac{\sum_{(a,b) \in F} d(a, b)}{|F|}$$

where $F \subseteq A \times B$ defines the pairs of elements and $|F|$ denotes the number of such pairs. In particular the d_{AL} set distance measure is normalized by

$|F| = |A||B|$; for d_{SMD} we have $|F| = |A| + |B|$; for d_S, d_L and d_{FS} we normalize by the cardinality of the surjection, linking or fair surjection on which the minimum distance was computed, i.e. $|F| = |R_i|$. All of the above normalization procedures do not alter the formal properties of the corresponding set distance measures. For d_M we used its normalized version given in (Ramon and Bruynooghe, 2001)

$$d_M(A, B) := \frac{2d_M(A, B)}{d_M(A, B) + (|A| + |B|)/2}.$$

Lists

In the remaining part of this chapter we will use the normalized edit distance.

Definition 3.29 (Normalized Edit Distance)**.** The *Normalized Edit Distance* is defined as:

$$d_{edit}(\ell_1, \ell_2) := \frac{1}{l} d_{edit}(\ell_1, \ell_2) \tag{3.21}$$

where l, as defined above, is given as $l = |\ell'_1| = |\ell'_2|$.

Unfortunately, the normalized d_{edit} is not a metric since it violates the triangle inequality.

Example 3.5. Consider $\mathcal{X} = \{a\}$ with a trivial metric and let $\alpha = 1$. Let $\ell_1 = [\underbrace{a \dots a}_{9}]$, $\ell_2 = [\underbrace{a \dots a}_{10}]$ and $\ell_2 = [\underbrace{a \dots a}_{15}]$. Then $d_{edit}(\ell_1, \ell_2) = \frac{1}{10}$, $d_{edit}(\ell_2, \ell_3) = \frac{1}{3}$ and $d_{edit}(\ell_1, \ell_3) = \frac{3}{5}$. However, $\frac{1}{10} + \frac{1}{3} < \frac{3}{5}$.

Proposition 3.12. *The normalized d_{edit} distance measure is semi-metric on $\mathcal{L}(\mathcal{X} \cup -)$ provided d is a semi-metric.*

3.3 Distances on Trees and Tree-like Structures

In the previous section we proposed various distance operators defined over the different building blocks of our composite objects. In this section we will move to distances, $d_{tree}(T_i, T_j)$, over two trees (or tree-like structures), T_i, T_j, defined in Section 2.3, which are constructed as a particular combination of the basic structures. The reason we devote a separate section to distances on trees is that in the remainder of this work trees (and tree-like structures) will be widely exploited to represent labeled graphs. Moreover, the main difference from distances over general composite objects and d_{tree} is that in the latter we put some additional constraints on the actual distance operators applied over basic structures. We focus here on trees where the labels, $\mathcal{L}_\mathcal{V}$ and $\mathcal{L}_\mathcal{E}$, are vectors in an Euclidean space. Moreover, we only consider unordered trees, such that all children nodes of a given node are of the same type. The extension of the presented ideas to ordered trees (and to other, more general trees) is straightforward.

Let x, y, be two elements of the two trees found at the same height h. These elements are of the same data type and are either two vertices, u, v, or two edges e_i, e_j. Then the distance between x and y is given by

$$d^2(x,y) = \begin{cases} \frac{1}{N'}\left(d^2_{tuple,p}(lab(x), lab(y)) + d^2_M(\delta(x), \delta(y))\right) & \text{if } \delta(x) \neq \emptyset \wedge \delta(y) \neq \emptyset \\ \frac{1}{N'}\left(d^2_{tuple,p}(lab(x), lab(y))\right) & \text{if } \delta(x) = \emptyset \wedge \delta(y) = \emptyset \\ \frac{1}{N'}\left(d^2_{tuple,p}(lab(x), lab(y)) + 1\right) & \text{if } \delta(x) = \emptyset \oplus \delta(y) = \emptyset \end{cases}$$

where $d_{tuple,p}(lab(x), lab(y))$ is, the Euclidean metric (i.e. $p = 2$) between the labels of the x, y from Equation 3.20; $\delta(x)$ is the neighborhood function, defined in Section 2.3, that returns either: the set of edges to which a vertex connects to as a starting vertex, if x is of type vertex, or the vertex to which an edge arrives if x is of type edge; d_M is the matching set distance measure, given in a general form in Equation 3.16, between the sets of elements that are found in the neighborhoods of x and y.

The N' in the denominator is also a normalization factor that corresponds to the dimensionality of the label vectors of x and y plus one to account for the set distance dimension, i.e. $N' = dim(lab(x)) + 1 = dim(lab(y)) + 1$. The reason we exploited the matching distance d_M from Equation 3.16 as the building block of d_{tree} is that it is equivalent to a frequency based distance, i.e. for each of the symbolic substructures in $\delta(x), \delta(y)$, it will count how many times it appears and the final distance will be the sum of differences of these frequencies. More precisely, for $M = 2$ the d_M distance measure between two sets A and B of symbolic elements is equivalent to the standard Euclidean distance of vectors v_A and v_B, of dimensionality equal to the number of distinct elements in $A \cup B$, where each dimension corresponds to the frequency of a specific element in sets A and B, respectively. It should be noted that in principle other set distance measures can be used to compute the distance between sets of siblings.

$d^2(x, y)$ is a recursive distance requiring the computation of set distances between the elements of the trees that are associated to x, y, at height $h + 1$. The final distance between two trees, T_i, T_j, is given by

$$d_{tree}(T_i, T_j) = d\left(root(T_i), root(T_j)\right) \tag{3.22}$$

From the above it is obvious that the computation of $d_{tree}(T_i, T_j)$ requires recursive and alternating computations of distances between nodes and edges. Moreover, in view of the various distances defined in the previous sections, it is obvious that d_{tree} can be equivalently implemented by a recursive application of the Euclidean distance on tuples as well as the Matchings set distance measure.

In order to show the computational complexity of the proposed tree kernel we use BF to denote the maximal out-degree of the considered trees, i.e. $\max_{v \in T(\mathcal{V}), T \in \mathcal{T}}\{|\delta(v)|\}$. The computation of the distance between two trees of height h is proportional to $O((BF^3)^{h-1}) = O(BF^{3(h-1)})$ (here we assume that the root of a tree is at level 1). This is a result of the fact that at level k we have to compute at most BF^{k-1} d_M, each of which has complexity $O(n^3)$ (n is

a cardinality of the corresponding sets). This is the pessimistic estimate of the time complexity and more accurate would be acquired if the average branching factors were used.

3.4 Properties of the Relational Distance

The formal properties of the distance measure applied over general composite objects, i.e. whether it is a metric, distance etc, depend on the formal properties of the constituent distance measures that are used.

For the moment let's consider composite objects whose building blocks are only sets, tuples and elementary attributes. It is obvious that for problems involving only three levels of recursion (i.e. composite objects are sets of tuples and elements of tuples are elementary attributes), the distance $distance(O_1, O_2)$ between two relational objects, O_1, O_2, is a metric iff the set distance measure employed is a metric, the distance on tuples is a metric, and distances on elementary objects are metrics. For problems involving k levels of recursion we can show that each of the distances defined on objects found at the $k-1$ level is a metric according to the arguments given before for the tree levels. Applying recursively the same reasoning we can easily see that at the top level the distance function that we get is also a metric. In general $distance(\cdot, \cdot)$ inherits the "weakest" properties of the constituent distance measures defined on sets, tuples and elementary objects. For example, if the d_{SL} (Equation 3.10) and $d_{tuple,p}$ (Equation 3.20) distance measures are used then the overall distance measure will be only a distance since d_{SL} is a distance, even though $d_{tuple,p}$ is a metric. The above reasoning applies also to more general objects which consist of tuples, sets and lists. The properties of constituent distance measures are propagated from the simpler objects towards the more complicated objects, and finally affect the final distance measure.

The computation of distance between two relational objects, o_1, o_2, is done in a recursive manner by computing the distances between objects' subparts that are associated with o_1, o_2. With an increasing depth the importance of the corresponding sub-objects becomes smaller. Objects that directly constitute the main object have a higher contribution on the distances. It is relatively easy to estimate the contribution of a given object to the final distance in association with the recursion depth at which the former is encountered. Consider a simple scenario where a complex object is defined in terms of elementary attributes, tuples and sets, where the elements of each set at level $k-1$ are in fact tuples which consist of elementary attributes, defined at level $k-1$, and sets defined at level k. Let d be the relational distance between two such composite objects, computed with a recursion depth of n, and $\hat{d}$ its approximation, computed with a recursion depth of $n-1$ (i.e. sets at level n are not included in the distance computation). The approximation error and thus the influence of the sets at level n, in the case of (normalized) d_{AL} and the (normalized) Euclidean distance, can

be shown to be bounded as follows

$$|d_1^2 - \hat{d}_1^2| \leqslant \frac{1}{\prod_{i=1}^{n-1} N_i}$$

where N_i is the number of attributes (standard plus set) of at level i. In other words the elements at level n contribute at most $1/\prod_{i=1}^{n-1} N_i$ to d. This bond is a result of the fact that d_{AL} takes values between 0 and 1, and at each level i the Euclidean distance is normalized by the number of attributes, N_i. By recursively combining the importance of objects at each level we obtain the desired result.

For other set distance measures, the bound is not straightforward to compute. Excluding sets at level n might alter the mappings between elements of sets of level $n-1$, nevertheless these distances too follow roughly the same law. The same reasoning also holds for other types of data where e.g. the edit distance over lists is used (Equation 3.19). A similar result was also given in (Bisson, 1992) where a closely related iterative approach was used for distance computation between complex objects.

A direct consequence of that result is that we can reduce the depth of recursion, thus speeding up classification time, without a significant loss in the accuracy of the distance computation. Moreover, the problem of self replicated loops encountered in relational algebra representation is alleviated since their contribution reduces with the depth of the recursion (see the discussion below Algorithm 2.1).

3.5 Experiments

In this section we will compare different instantiations of the complex distance measures defined in the above sections on a number of relational problems (presented in Section 3.5.3). The learning task will be always classification and we will use the kNN classification rule presented in Section 3.5.1 to decide the classification of a given instance. In the experiments we focus on set and list distance measures, and perform their in-depth empirical analysis (Sections 3.5.4 and 3.5.5). More precisely we would like to examine whether the qualitative characterization of these distances measures is supported by the experimental observations. Next, in Section 3.5.6 we analyze how our distance-based system compares with other state-of-the-art relational systems. In Section 3.5.7 we show how we can automatically select the composite distance measures from a set of predefined distance measures. Finally, in Section 3.5.8 we examine the influence of the θ parameter on the predictive performance of the Tanimoto distance.

3.5.1 K-Nearest Neighbor

Here we describe the *k-Nearest Neighbor* (kNN) method which will be used in the rest of this section as a classification rule to decide the class label of a new unseen composite object. The kNN rule (Duda et al., 2001) is one of the simplest and most popular data mining and machine learning algorithms. In its simplest

form, 1-NN assigns a new object to the class of its nearest neighbor chosen from the training set. In the k-NN rule an unknown object is assigned to the class that is most frequent among its k neighbors.

We assume a training set $D = \{(x_i)\}_{i=1}^{n}$ for $x_i \in \mathcal{X}$ together with class labels $class(x_i) \in C$, where C is a set containing symbolic values, i.e. we only deal with *classification* problems. We assume that for each pair of elements $x_i, x_j \in \mathcal{X}$ it is possible to compute a (normalized) distance measure d such that it assigns values close to 0 for instances being "similar" and close to 1 for instances being "dissimilar". For each point $x_i \in D$ in the training set we can determine the k nearest neighbors denotes as $N(x_i) = \{x_{i_1}, \ldots, x_{i_k}\} \subseteq D$ according to the distance measure d. Usually, the k parameter is assumed to be odd to avoid ties (for two-class problems). We also denote $N_c(x_i) = \{x_i \in \mathcal{X} \mid x_i \in N(x) \ \wedge class(x) = c\}$, i.e. neighbors of x which have class c. The class value for a new instance $x' \in \mathcal{X}$ is determined from a neighborhood $N(x')$ as

$$class_{pred}(x') = \operatorname*{argmax}_{c \in C} \sum_{x \in N_c(x')} \delta(class(x), c)$$

where $\delta(x, y) = 1$ iff $x = y$ and $\delta(x, y) = 0$ otherwise.

The kNN method is a simple and intuitively appealing yet effective algorithm for the task of classification. The advantage of this algorithm comes mainly from the fact that its decision surfaces are non-linear, it can be easily adapted to incremental learning, and there is only one single integer parameter k that can be easily tuned with cross-validation. Moreover, the expected quality of kNN improves with the increased amount of training data, making it asymptotically at most twice as bad as the Bayes rule (Duda et al., 2001). Additionally, it requires no modification for multi-category classification problems. In Chapter 4 we will see that in comparison with SVM (Cristianini and Shawe-Taylor, 2000) kNN performs better for problems with large number of classes. Its main advantage from our perspective, however, is that kNN can be easily adapted to classify composite instances since it does not require a direct access to the training examples, instead it accesses the data only via a distance measure (this is in general true for any distance-based algorithm).

The main disadvantages of this algorithm are large storage and computational requirements as well as sensitivity to outliers. Moreover, if the data is sparsely sampled and the underlying distance measure is not a metric, the classification performance of the kNN algorithm may significantly differ from its asymptotic behavior.

3.5.2 Experimental Setup

The learning task will always be classification. We will use the kNN classification rule presented in Section 3.5.1 to decide the classification of a given instance; the k parameter was optimized in an inner 10-fold cross validation loop over the set $k = \{1, 3, 9\}$. We estimate accuracy using 10-fold stratified cross-validation and control for the statistical significance of observed differences for

all pairs of distance measures using McNemar's test (McNemar, 1947) (significance level=0.05). In order to have a more global picture, than the one provided by the basic pairwise comparisons, of the relative merits of the different distance measures, we establish a ranking schema based on the results of the pairwise comparisons (Kalousis and Theoharis, 1999). More precisely for a given dataset if distance measure d_1 is significantly better than d_2 then d_1 is credited with one point and d_2 with zero points; if there is no significant difference then both are credited with half point. Since we are comparing 11 different relational distance measures it is obvious that if there is a distance measure that is better than all the others it will be credited with ten points, if it is worse than all the others it will be credited with zero points while if all distances are equivalent they will be all credited with five points.

For the above problems we exploited the Euclidean distance on tuples from Equation 3.20 (i.e. we fix $p = 2$). For representations involving sets the various set distance measures presented in Section 3.2.3 were used. The threshold parameter, θ, used in d_T was a priori fixed to 0.01; in Section 3.5.8 we will discuss how the value of this parameter affects classification performance of d_T. For decompositions into trees we used the recursive tree distance presented in Section 3.3 while for lists we exploited the edit distance measure from Section 3.2.4. Finally, the two distance measures on elementary attributes from Section 3.2.1 are used.

3.5.3 Datasets

We will experiment on a number of relational problems which are presented in Appendix A. According to the actual representation of the learning examples the learning problems can be divided naturally into three main subgroups.

The learning instances in the first group are represented as sets of vectors, hence these learning problems can be simply reduced to direct comparisons of sets of fixed-length tuples requiring no further recursion. The datasets in this group are diterpenes, musk (both versions) and duke.

For the datasets in the second group, i.e. graph classification problems, different representations of the data are possible (see Section 2.3). Here we focus on two representations where graphs are decomposed into sets, trees and tree-like structures; decompositions into other sub-structures (i.e. walks) will be considered in Chapter 5. As already mentioned in Section 2.3 the considered types of decompositions are obtained from a depth first exploration emanating either from each vertex or from each edge a graph and yielding all recursive structures of height h. More precisely in the first representation the trees have nodes as roots; we consider trees of height 3 (i.e. a tree contains only the root together with two adjacent edges and vertices). The second representation consists of tree-like structures which have edges as roots. In this case the considered trees have height of 2 (i.e. trees have edges as roots which are connected to a set of corresponding vertices). The graph datasets in this group are mutagenesis and 4 versions of carcinogenicity (i.e. FM, FR, MM and MR).

Finally, we also experimented with the protein fingerprint dataset where

Distance	diterpenes	musk (ver. 1)	musk (ver. 2)	duke
d_{SL}	51.63 (2.0)(+)	81.52 (6.5)(+)	69.61 (5.0)(=)	41.46 (3.5)(=)
d_{CL}	21.82 (0.0)(-)	66.30 (1.5)(=)	68.63 (5.0)(=)	39.02 (3.0)(=)
d_{AL}	39.45 (1.0)(+)	82.61 (6.5)(+)	68.63 (5.0)(=)	58.54 (5.0)(=)
d_{SMD}	95.14 (7.0)(+)	80.43 (6.5)(+)	75.49 (5.0)(=)	*78.05 (6.5)*(=)
d_{H}	84.96 (3.0)(+)	80.43 (6.5)(+)	76.47 (5.0)(=)	60.98 (5.0)(=)
d_{RIBL}	95.14 (6.5)(+)	65.22 (1.5)(=)	61.76 (4.5)(=)	68.29 (6.0)(=)
$d_{T,\theta=0.01}$	*97.41 (10.0)*(+)	51.09 (1.0)(=)	61.76 (4.5)(=)	58.54 (4.5)(=)
d_{S}	95.54 (6.0)(+)	*83.70 (7.0)*(+)	73.53 (5.0)(=)	63.41 (5.5)(=)
d_{FS}	95.61 (6.0)(+)	82.61 (6.5)(+)	*77.45 (6.5)*(+)	58.54 (5.0)(=)
d_{L}	94.48 (5.5)(+)	*84.78 (7.0)*(+)	73.53 (5.0)(=)	73.17 (6.0)(=)
d_{M}	96.14 (8.0)(+)	70.65 (4.5)(+)	61.76 (4.5)(=)	63.41 (5.0)(=)
Def. Acc.	29.81	51.09	61.76	58.54

Table 3.3: Accuracy and rank results (the first parenthesis) on the datasets where instances are represented as sets of vectors. The ranking is based on the scheme described in Section 3.5.2. The sign in the second parenthesis compares the performance of the corresponding distance measures with that of the default classifier (Def. Acc.).

learning instances are presented as general relational structures. We used two different representations of the learning examples. In the first representation each learning instance is associated with a *set* of "motifs"; in the second representation each instance is associated with a *list* of "motifs". We also experimented with the following weighting scheme in the top level description of learning objects: all elementary attributes are assigned equal weights, composite attributes are assigned the same weight multiplied by the number of elementary attributes they have except for "motifs" object where we put double that value. More precisely we assigned value 0.1428 to all the elementary attributes, i.e. 1 divided by the total number of attributes (12), motifs was assigned weight of 2 x 10 x 0.1428, while information corresponding to tables *PropMaj1234*, *NormEnt1234*, *PropAtLeast3or4* and *RHS* were assigned values of 4 x 0.1428, 4 x 0.1428, 2 x 0.1428 and 3 x 0.1428, respectively (see Figure A.2 in Appendix A.3). This weighting schema reflects the fact that the composite objects representing statistics on the set of proteins could be stored in the main table. On the other hand by putting double weight for the "motifs" object we emphasize the importance of the information associated with it. We mention here that the above weighting scheme was assigned in an ad-hoc manner; in Chapter 5 we will propose a framework which allows for setting the weights in an automatic and principled manner.

3.5.4 Classification Performance

The results are presented in Tables 3.3, 3.4 and 3.6; the detailed significance results are given in Appendix B.1. The top ranked distance measures (according

Distance	muta (ver. 1)	muta (ver. 2)	FM (ver. 1)	FM (ver. 2)
d_{SL}	77.13 (5.0)(+)(=)	71.81 (2.5)(+)	59.31 (5.5)(=)(=)	58.74 (5.5)(=)
d_{CL}	79.26 (5.0)(+)(+)	68.09 (1.5)(=)	49.00 (2.0)(-)(=)	48.42 (2.5)(-)
d_{AL}	65.96 (1.0)(=)(=)	68.62 (1.5)(=)	58.17 (5.5)(=)(=)	54.73 (4.5)(=)
d_{SMD}	82.45 (6.5)(+)(=)	83.51 (7.5)(+)	53.87 (4.0)(=)(=)	58.74 (6.0)(=)
d_H	78.72 (5.0)(+)(=)	78.72 (5.0)(+)	59.60 (6.5)(=)(=)	60.46 (6.0)(=)
d_{RIBL}	73.40 (3.0)(+)(=)	78.19 (6.0)(+)	56.73 (5.5)(=)(=)	55.30 (5.0)(=)
$d_{T,\theta=0.01}$	*87.23 (9.0)*(+)(=)	*87.23 (8.5)*(+)	*63.32 (7.5)*(=)(=)	*62.46 (7.0)*(=)
d_S	79.79 (5.5)(+)(=)	78.72 (5.5)(+)	52.72 (3.5)(=)(=)	51.86 (3.0)(=)
d_{FS}	81.91 (6.5)(+)(=)	81.91 (7.0)(+)	55.30 (4.5)(=)(=)	54.15 (4.0)(=)
d_L	80.85 (5.5)(+)(=)	82.98 (7.0)(+)	58.17 (6.0)(=)(=)	61.32 (6.5)(=)
d_M	73.40 (2.5)(=)(=)	73.40 (3.0)(=)	55.01 (4.5)(=)(=)	55.59 (5.0)(=)
Def. Acc.	66.49		59.03	

Distance	FR (ver. 1)	FR (ver. 2)	MM (ver. 1)	MM (ver. 2)
d_{SL}	*65.53 (5.5)*(=)(=)	*65.53 (5.5)*(=)	61.90 (6.0)(=)(=)	61.31 (5.0)(=)
d_{CL}	57.26 (2.5)(-)(=)	58.97 (3.5)(-)	42.86 (0.0)(-)(-)	52.98 (2.5)(=)
d_{AL}	*65.24 (5.5)*(=)(=)	64.39 (5.0)(=)	55.65 (4.0)(=)(=)	60.42 (5.0)(=)
d_{SMD}	63.82 (5.0)(=)(=)	*65.24 (5.5)*(=)	62.80 (6.0)(=)(=)	61.90 (5.5)(=)
d_H	61.82 (5.0)(=)(=)	62.96 (5.0)(-)	58.93 (5.5)(=)(-)	*65.18 (6.0)*(=)
d_{RIBL}	*64.67 (5.5)*(=)(=)	64.39 (5.0)(=)	56.55 (5.0)(=)(=)	59.52 (5.0)(=)
$d_{T,\theta=0.01}$	63.82 (5.0)(=)(=)	62.39 (5.0)(=)	57.14 (5.5)(=)(=)	58.63 (4.5)(=)
d_S	*65.24 (5.5)*(=)(=)	62.39 (5.0)(=)	61.90 (5.5)(=)(=)	60.71 (5.5)(=)
d_{FS}	61.54 (5.0)(=)(=)	61.82 (5.0)(=)	62.20 (5.5)(=)(=)	58.63 (5.0)(=)
d_L	*64.10 (5.5)*(=)(=)	*66.67 (5.5)*(=)	*63.39 (6.5)*(=)(=)	62.80 (5.5)(=)
d_M	62.11 (5.0)(=)(=)	62.68 (5.0)(=)	58.33 (5.5)(=)(=)	61.01 (5.5)(=)
Def. Acc.	65.53		61.61	

Distance	MR (ver. 1)	MR (ver. 2)
d_{SL}	55.52 (5.5)(=)(=)	55.81 (6.0)(=)
d_{CL}	43.90 (0.5)(-)(=)	49.42 (4.0)(=)
d_{AL}	54.65 (5.5)(=)(=)	52.03 (5.0)(=)
d_{SMD}	*61.63 (7.0)*(+)(=)	56.69 (6.0)(=)
d_H	52.91 (4.5)(=)(=)	*57.56 (6.5)*(=)
d_{RIBL}	54.65 (5.5)(=)(+)	47.97 (3.0)(-)
$d_{T,\theta=0.01}$	54.94 (5.5)(=)(=)	*58.43 (6.5)*(=)
d_S	54.36 (4.5)(=)(=)	52.33 (5.0)(=)
d_{FS}	54.94 (5.5)(=)(=)	52.91 (5.5)(=)
d_L	*61.34 (7.0)*(+)(+)	54.94 (5.5)(=)
d_M	49.42 (4.0)(=)(=)	45.93 (2.0)(-)
Def. Acc.	55.81	

Table 3.4: Accuracy and rank results (the first parenthesis) on the graph datasets. For each dataset the performance for two different representation is given. The ranking is based on the scheme described in Section 3.5.2. The sign in the second parenthesis compares the performance of the corresponding distance measures with that of the default classifier (Def. Acc.). The sign in the third parenthesis for version 1 of datasets compares the performance with version 2.

d_{CL}	d_{AL}	d_{RIBL}	d_M	d_{SL}	d_H	d_S	d_T	d_L	d_{FS}	d_{SMD}
2.53	4.38	4.78	4.94	5.03	5.31	5.31	5.34	5.44	5.72	6.16

Table 3.5: Average ranks of set distances over all the datasets. The ranking is based on the scheme described in Section 3.5.2.

to the rank results presented in the first parenthesis) are emphasized. In all the tables the values in the column with signs (-, + and =) indicates whether the performance of the corresponding distance measure is significantly better (the + sign) or worse (the - sign) than that of a specific algorithm (mentioned in tables' descriptions) or there is no significant difference (the = sign). In Table 3.6 the accuracies are given both for the weighted and non-weighted versions of the dataset. Moreover, we report the performance using 10-fold stratified cross validation (the "10-fold CV" columns) and on a independent test set (the "test set" columns). On the independent set we fixed $k = 9$ as this was the most frequent value returned by the internal cross validation.

The rest of this section is divided into three parts. First, we analyze the performance of the various set distance measures (this analysis will be further continued in the subsequent sections). Then, we examine the results obtained for the protein fingerprint dataset where two different weighting schemes and two different representations are used. Finally, we analyze the relative performance of the two different representations used in the graph datasets.

Performances of Set Distances

From the overall results it is obvious that there is no set distance measure which is the overall winner, something that was to be expected. To give an overall picture of the performance of the distance measures we averaged their rankings over the different datasets and report their average ranks in Table 3.5. There are however some distance measures that perform consistently well over a series of problems. The example of such distance measure is d_{SMD} which is top-ranked or its performance is very close to the top performing. In terms of the average rank d_{SMD} is on the best position. What makes this distance measure even more interesting is that it is one of the simplest distance measures with only quadratic complexity. The two other set distance measures which have good average performances are d_{FS} and d_L, however these have cubic complexities.

What came as a surprise was the low performance of the matching based distance measure, d_M, in terms of its average rank it was placed only on the eighth position among the eleven different measures. Quite astonishing was its bad performance on the musk dataset for which previously it was reported to have a very good performance (Ramon and Bruynooghe, 2001). We experimented also with its non-normalized version for which indeed it performed well on the musk problem; for the other datasets there were no differences between the normalized and non-normalized versions (in the above paper it is not clear which version the authors used, i.e., normalized or non-normalized). Another important difference

from the results reported in (Ramon and Bruynooghe, 2001) is the performance of the d_{FS} set distance measure on the musk (ver. 1) and diterpenes datasets; more precisely (Ramon and Bruynooghe, 2001) report accuracies of 49 % and 87 % for the two datasets while our error estimation gives much higher accuracies, 82.61 % and 95.61 % respectively. These big differences cannot be explained by a different experimental setup since they used 10-fold stratified cross-validation. In an initial phase we had similar error estimates only to discover later that this was due to a limitation of the library used to solve the min weight maximum flow problem.

In contrast to d_M the three other relation based distance set measures, d_{FS}, d_L, and d_S, exhibit a rather good predictive performance. In terms of their average ranking they take the second, third and fourth position, respectively. Their main difference from d_M is the fact that each element of a set must be a part of the mapping between the two sets. No element is left outside as it is the case with d_M. Another difference is the way that d_{FS}, d_L, and d_S are normalized, i.e. by the cardinality of the relation over which they were computed. d_{FS}, d_L, and d_S are rather similar to d_{SMD} in these respects, i.e. the normalization[2] and the fact that all the elements are accounted for in the distance computation. Note that these four distance measures were ranked in the top four positions when we average their rankings over all the datasets, Table 3.5.

d_{RIBL} differs from d_{FS}, d_L, and d_S, in the same way that d_M does. d_{RIBL} does not use all the elements of the two sets in the distance computation, many of the elements of the larger set can be left aside. The normalization of the distance is not done by the cardinality of the relation that d_{RIBL} imposes between the two sets but by the cardinality of the larger set. If we take a look on the ranking of d_{RIBL} and d_M averaged over all the datasets we see that they are ranked next to each other, with a small difference in their average ranks, with d_{RIBL} being ranked on the ninth position and d_M on the eighth.

In Section 3.2.3 we also mentioned that d_{SMD} (ranked on the first position) and d_{RIBL} (ranked on the ninth position) are similar in the sense that the former distance measure can be considered as a symmetrized version of d_{RIBL}. This fact indicates that the symmetry of a distance measure is a fundamental property, largely influencing the predictive performance of a distance-based algorithm.

Another distance measure that is similar to d_{FS}, d_S, d_L, and d_{SMD}, with respect to normalization and accounting for all the elements, is d_{AL} which nevertheless was ranked usually on the low positions. d_{AL} exhibits a fundamental flaw, it does not satisfy the reflexivity property, which could explain its bad behavior.

A rather separate family of distance measures are the ones that base their distance only on a single pair of elements, namely d_H, d_{SL} and d_{CL}. The performance of the two former with respect to their rankings averaged over all the datasets is very similar, in fact they are ranked close to each other taking the fifth and seventh position. d_{CL} though has a very bad performance which could

[2]In d_{SMD} normalizing by the sum of the cardinalities of the two sets is equivalent to normalizing by the cardinality of the relation defined by d_{SMD}.

Distance	no weights		with weights	
	10-fold CV	test set	10-fold CV	test set
d_{SL}	83.32 (6.0)(5.0)(+)	78.87	85.34 (7.5)(6.5)(+)(+)	82.25
d_{CL}	83.25 (6.5)(5.5)(+)	78.31	83.39 (2.0)(1.5)(+)(=)	81.13
d_{AL}	83.86 (6.5)(5.5)(+)	79.44	84.80 (5.0)(4.5)(+)(=)	83.10
d_{SMD}	84.33 (7.5)(6.5)(+)	79.16	*86.01 (9.0)*(8.0)(+)(+)	82.25
d_H	83.86 (6.5)(5.5)(+)	78.87	84.80 (5.5)(5.0)(+)(=)	81.41
d_{RIBL}	83.66 (6.5)(5.5)(+)	79.72	84.40 (4.5)(4.0)(+)(=)	82.53
$d_{T,\theta=0.01}$	79.96 (1.0)(0.5)(+)	74.65	81.71 (1.0)(1.0)(+)(+)	79.15
d_S	84.13 (6.5)(5.5)(+)	79.15	85.81 (8.5)(7.5)(+)(+)	82.25
d_{FS}	84.20 (7.0)(6.0)(+)	79.44	*85.88 (9.0)*(8.0)(+)(+)	81.41
d_L	78.28 (0.5)(0.5)(+)	75.21	83.46 (3.0)(2.5)(+)(+)	78.03
d_M	*85.41 (10.0)*(9.0)(+)	82.25	85.07 (7.0)(6.5)(+)(=)	82.53
d_{edit}	81.10 (1.5)(+)	76.34	83.86 (4.0)(+)(+)	86.15
Def. Acc.	54.40			

Table 3.6: Accuracy and rank (the first parenthesis) results on the protein fingerprint dataset, both with and without weights. The ranking is based on the scheme described in Section 3.5.2. The second parenthesis contains ranks results where d_{edit} is not considered (when computing the ranks from the first parenthesis, d_{edit} was included). The sign in the third parenthesis compares the performance of the corresponding distance measures with that of the default classifier (Def. Acc.). The sign in the fourth parenthesis for weighted dataset compares the weighted with non-weighted versions.

be explained by the fact that it is sensitive to outliers and does not satisfy the reflexivity property.

The Tanimoto based distance is quite different from all the above. Although it constructs a mapping between the elements of the two sets based on the threshold parameter[3] it does not compute the distance between the two sets based on the distances included in the mapping. It uses the mapping to compute a degree of fuzzy overlap between the two sets. Some clarifications should be given here on its parameter setting. When we performed the experiments we have chosen the value of the threshold to be equal to 0.01 since we considered that to be a sensible choice. We did that a priori and without any tests. However, experimenting afterward with the three datasets (duke, musk ver. 1, 2) in which it did not perform well we found parameter settings for which it performed much better, a more informative way of selecting the value of the threshold would result in better performance (see Section 3.5.8). Nevertheless we have chosen to report results only on this initial setting so that the results are not biased. In terms of its average rank it performs quite well taking the sixth position over all distance measures.

No general statement can be done about the superiority of one distance measure over another. There are though some distance measures that exhibited a

[3]It does not account for all the elements of both sets. In this respect d_T is more similar to d_{RIBL} and d_M.

good and stable performance over the small number of datasets that we examined here. For example d_{SMD}, the good performance of which coupled with its quadratic computational complexity make it a good first choice. However, in general everything depends on the type of application and its underlying assumptions which should mainly guide the selection of the distance measure. In the absence of such assumptions the obvious question is how distance selection could be done. In Section 3.5.7 we will show one way to achieve that.

Performances for Protein Fingerprints

From the results presented in Table 3.6 it is clear that the weighting scheme in the top level description of the learning instances has a positive impact on the predictive performance of the kNN algorithm (the results are reported in the fourth parenthesis for the weighted version of the dataset). For all the distance measures (except for d_M) there is an improvement in predictive accuracy in comparison with the unweighted schema; the difference in performances is statistically significant in seven out of twelve cases. The biggest advantage is for the d_L set distance measure where the difference is 5.11 %. This observation suggests, considering that the weights were selected in an ad-hoc manner, that there is a space for improvement if other, more elaborate weighting methods, e.g. adaptive, are used. Such algorithms will be proposed in Chapter 5.

The other observation is that in terms of predictive performance the approach based on sets have better representation power that the one based on list. In the weighted version of the dataset and for 10-fold CV the d_{edit} distance measure performs poorly and it is ranked on the eighth position together with d_{RIBL}. On the other hand for the weighted case there is a clear advantage of the representation based on lists when comparing the results on the independent test set. This could indicate that the representation based on lists reflects the semantics of the data, and the performance could be improved if other operators (e.g. other distances or kernels on lists) are exploited.

Performances of Graph Representations

Finally, we compare the two different graph representations described in Section 3.5.3, i.e. version 1 where decompositions are into trees of height 3, and version 2 where decompositions are into tree-like structures of height 2. From the results it is clear that depending on which set distance is used, the optimal decomposition is different for different datasets. For example, in mutagenesis for d_{CL} the better result is obtained in the version 1 of the dataset, while in the MR dataset it is the other representation which has a higher predictive performance. The first representation is significantly better in two cases, while the second is significantly better in three cases out of fifty five cases; in other case the differences are not statistically significant.

The above results indicate that in general it is difficult to specify in advance the appropriate type of substructures for a given problem. The appropriate type(s) of the subgraphs should be specified on the basis of domain knowledge

and applications requirements, but, even in the presence of domain knowledge, it can be far from obvious which representation should be used. A common intuition is that by decomposing into more complex subgraphs the expressivity, and consequently the performance, of resulting resulting algorithms increase. This is, however, in contrast with some experimental evidence (Menchetti et al., 2005) in the context of graph kernels which show that decompositions into rather simple substructures perform remarkably well with respect to more complex decompositions on a number of different datasets. This observation suggests that there is a need for a more automated way of determining a good decomposition for a problem at hand. This problem will be tackled in Chapter 5 where we propose a framework for learning "good" decompositions of graphs. More precisely, we will show how to automatically combine a number of predefined decompositions.

3.5.5 Families of Set Distance Measures

Based on qualitative characteristics of the different set distance measures, there is a division of these distances based on how they account for the elements of the two sets. Possible options are:

1. base distance only on a single pair of elements
2. account for subsets of the two sets
3. account for all the elements of the two sets

This division is reflected on the performances of the measures as these are given by their average rankings presented in Table 3.5. Measures that share common features have in general a similar average ranking among the different problems.

Going one step further we have tried to cluster the different set distance measures based on their ranking on each dataset. Each measure was described by a sixteen element vector where each feature of the vector corresponds to the ranking of the given measure in one of the datasets[4]. We used an agglomerative hierarchical clustering algorithm (Duda et al., 2001), together with *Ward*'s minimum-variance method to determine which clusters should be merged at each agglomeration step. The resulting dendrogram is given in Figure 3.4.

It should be stressed that the assumption we make here is that semantically similar set distances will lead to similar relative performances and will be grouped in one cluster. However, the opposite is not true, i.e. set distances with different qualitative characteristics might have similar relative performances (e.g. d_{SL} and d_S have the same ranks in musk (ver. 2), even though these distances have different characteristics).

We can see that this performance based clustering roughly agrees with the conceptual differences among the distance measures. The two measures that exploit all the instances and normalize by the cardinality of the relation that

[4]For the protein fingerprints dataset we computed the ranking excluding d_{edit}. This ranking is presented the second parenthesis in Table 3.6

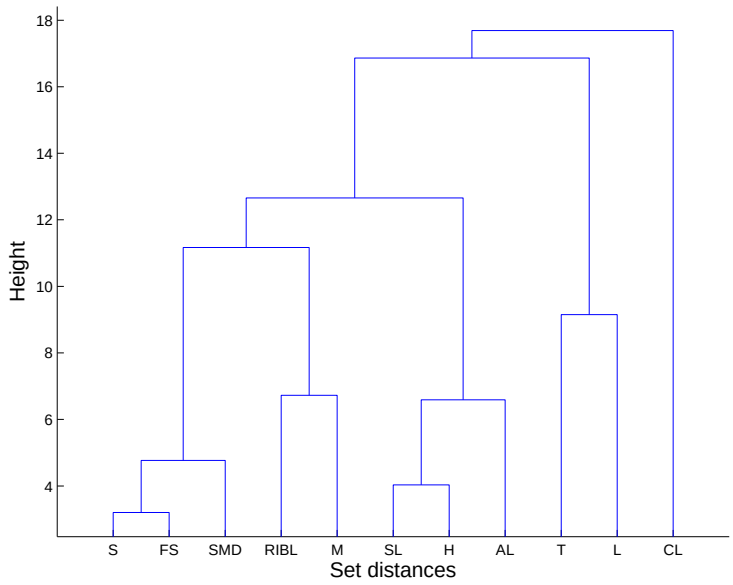


Figure 3.4: Clustering the relational distances with respect to their rank performance among the datasets examined.

they induce, d_{SMD}, d_{FS} and are first to be merged. Then this cluster is merged with d_S which intuitively makes sense since d_S and d_{FS} both use surjection in order to match elements of the two sets, with d_{FS} imposing the extra constraint that surjections should be fair. What came as a surprise is that d_L is not together with d_{SMD}, d_{FS}, d_S, even though it also uses all the instances and normalizes by the cardinality of the optimum linkings.

The next cluster merging is that of two measures that are based on a single pair of instances and respect the reflexivity property, d_{SL}, d_H, further merged with d_{AL}. The latter set distance measure takes into account all the elements, but it can be also considered as computing the distance measure between means of the corresponding two sets. The next merge is that of d_M and d_{RIBL}, creating a cluster of distance measures that do not account for all the elements of the two sets. What comes as a surprise is the fact that d_T and d_L are placed in one cluster; this an example where set distance measures with different characteristics are grouped together as they happen to have similar relative performances. Finally, the fact that d_{CL} is placed in its own cluster indicates which reflect the fact that in comparison with other distances d_{CL} has consistently a poor performance among different datasets as can be seen from Table 3.5. It should be mentioned that given the small number of datasets examined it is an open question whether this performance based clustering would persist and reflect in the same manner the qualitative differences of the measures if more datasets are considered.

Loosely speaking a set distance measure computes a distance in a high dimensional space where the number of dimensions equals the product of cardinalities of the two sets whose distance is computed. Each dimension is defined by a specific pair of elements of the two sets. Depending on the set distance measure used all, some, or a single dimension, are considered in order to compute the final distance. One can view that as a weighted distance computation where the dimensions that are not used have a weight of zero. This view is directly reflected on the first dimension of the qualitative characterization of the set distance measures. Normalizing then by the cardinality of the relation that is used to compute the set distance is simply averaging over the dimensions that are accounted for in the distance computation.

The clustering of the distance measures according to their relative performances and the characterization of the clusters according to the dimensions given in the beginning of this section is strongly connected to the different types of problems that we can face. Relational classification problems can be characterized along the first two qualitative dimensions, e.g. whether all, subsets or a specific pair should determine classification, for example multi-instance classification problems fall in the last category. If we have a way to characterize a classification problem along these dimensions then we would expect the associated cluster of distance measures to have the best performance for that problem.

3.5.6 Comparison with Other Systems

The right choice of distance measure results in a classification performance that compares favorably with the performances of state-of-the-art learning systems. In Table 3.7 we give the performance of some relational systems along with the performance of the best distance measure which is reported in the kNN_{Best} column (the entries in the kNN_{CV} column will be discussed in Section 3.5.7). The best distance measure was selected according to the ranking results. We note that for some datasets there are more than one best distance measures. For the graph datasets we choose the representation with the highest performance. Finally, for protein fingerprints we exploited the weighted version of the dataset. The results of other systems are divided into three groups: (i) results for distance-based classifiers, (ii) results of kernel-based algorithms and (iii) other general well-known systems. In the table we only report results on the same problem formulation, with the same accuracy estimation procedure (by default it is 10-fold cross-validation; if the evaluation method is different we explicitly indicate it). When more than one result is available we always take the best.

The results in diterpenes for *Tilde*, *RIBL*, *FOIL* and *ICL* are reported from (Džeroski et al., 1996). The result for *Matching* are from (Ramon and Bruynooghe, 2001) while *KeS* and *DeS* are from (Gärtner et al., 2004) and denote kernel and distance based learners, respectively. For both versions of the musk datasets the algorithms compared are *EM-DD* (Zhang and Goldman,

2002)[5], *DD* (Maron and Lozano-Pérez, 1998), *IAPR* (Dietterich et al., 1997), *mi-SVM* and *MI-SVM* (Andrews et al., 2002), *MIK* (Gärtner et al., 2002) and *Des* and *KeS* of Gärtner et al. (2004). Moreover, for musk (ver. 1) the *TILDE* result was the best result reported in (Blockeel and De Raedt, 1997), while for *Matchings* the result is from (Ramon and Bruynooghe, 2001).

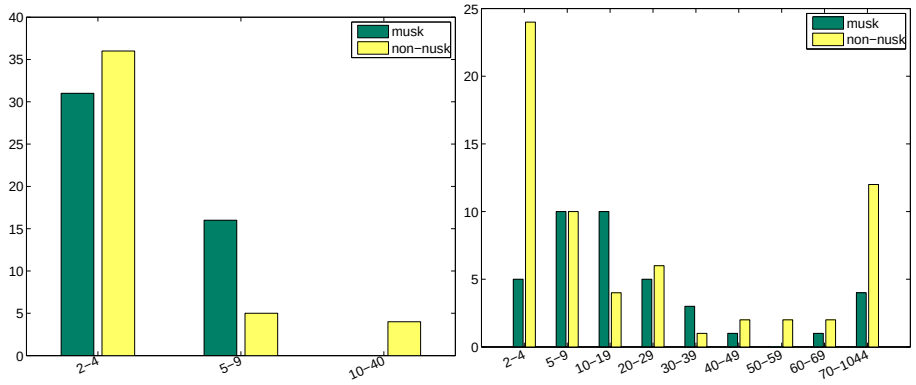
For mutagenesis the results are taken from (Blockeel and De Raedt, 1997) on the B_2 formulation of the problem that corresponds to our version 2 of mutagenesis and from (Ramon, 2002) for the *Matchings*. For the *KES* and *DES* systems the results are from (Gärtner et al., 2004) while *MIK* is taken from (Gärtner et al., 2002). *K1*, *K2*, *K3* are the graph kernels (evaluated using leave-one-out cross validation) from (Kashima et al., 2003), (Mahé et al., 2004) and (Ralaivola et al., 2005), respectively. *RK* is the relational kernel of Cumby and Roth (2003). For mutagenesis there is a variety of other results reported in the literature showing even better performance. Nevertheless, these have been achieved on a different problem formulation that contained more information. However, for completeness we will mention few of them. *G-NET* (Anglano et al., 1998) achieved an accuracy of 91.2% on the merge of the regression-friendly and unfriendly datasets (see Appendix A.2) using as input: information on atoms and bonds, global properties of the molecules (five attributes, e.g. hydrophobicity) and chemical structures present in the molecules, e.g. benzenic rings. On similar representation (Lodhi and Muggleton, 2005) presented results based on an ensemble method using *Aleph*-generated theories in which they achieved a cross-validated accuracy of 95.8 %. Similarly, in all the carcinogenicity datasets we only cite works which use similar features to describe atoms and bonds. In particular our results on this dataset are not directly comparable with the ones reported e.g. in (Fröhlich et al., 2005).

The results for the protein fingerprints classification are taken from (Hilario et al., 2004) and were obtained using both 10-fold cross-validation and on an independent test set.

For the diterpenes the best distance is better than all the other reported performances. The best distance is also close to the best results in the FM, FR and MM datasets. For mutagenesis the best distance ranked fourth after *MIK*, *K2* and *K3*. For musk (ver. 1) the best distance is ranked 7th (out of 8 competitors), however, the differences in performance are probably not significant considering the small size of this dataset. The poor performance on the musk (ver. 2), although the same type of application as with version 1, could be explained by the fact that the distributions of set cardinalities are very skewed and range from one to more than one thousand. Figure 3.5 shows the cardinality distribution for the two problems.

For the datasets on which the best distance is not the winner, i.e. musk and graph datasets the best performance is achieved by a large-margin classifier

[5] As noted by Andrews et al. (2002) the results of *EM-DD* presented in (Zhang and Goldman, 2002) (96.8 % for version 1 of musk and 96.0 % for version 2 of musk) are optimistically biased since the test data was used to select the optimal solution obtained from multiple runs of the algorithm. Here, the presented results are from (Andrews et al., 2002) who used the corrected version of *EM-DD*.



(a) musk (ver. 1) (b) musk (ver. 2)

Figure 3.5: Distributions of set cardinalities for the two versions of musk.

using a specialized kernel, based either on sets or on graphs. In Chapter 4 we will propose kernel functions which compare favorably with these state-of-the art kernels. In protein fingerprints the performance of d_{SMD} using 10-fold cross-validation is better than the best result, however the performance of the same distance measure on an independent test is lower that the corresponding predictive accuracy from (Hilario et al., 2004).

It is obvious that the above comparison between the best distance and the other relational learners is optimistically biased since the former is the result of extensive experimentation[6] and of an a-posteriori selection of the best. The comparison would have been fair if the selection was done a priori without looking at the performance on the test set. Actually this is related to the problem of selecting among different models – parameterizations of a learning algorithm – or among different learning algorithms the most appropriate for the problem at hand. In the next section we will see how we can perform distance selection in the absence of any assumptions about the problem at hand. The selection strategy will also provide a basis for fair comparison with previous work.

3.5.7 Selecting Complex Distances

The problem of selecting the appropriate model among a set of candidate models or selecting the appropriate classification algorithm from a set of classification algorithms has received considerable attention in the machine learning and data mining community. Here we describe one of the simplest and most often used strategies to tackle the above problem, which is based on cross-validation on the training set (Schaffer, 1993). In this method the error of all classification algorithms is estimated on the training set using cross-validation, the algorithm

[6] Although the same could be argued for most of the results of the other relational systems given in Table 3.7 since when multiple results where available we reported on the best.

kNN_{Best}	kNN_{CV}	Distance-based	Kernel-based	General
		diterpenes		
97.41 d_T	97.41 (=)	RIBL 91.20	KeS 94.70	Tilde 90.40
		DeS 97.10		Foil 78.30
		Matchings 93.50		ICL 86.00
		musk (ver. 1)		
84.78 d_L	80.43 (=)		KeS 81.00	EM-DD 84.80
			MIK 91.60	DD 88.90
			mi-SVM 87.40	IAPR 92.40
			MI-SVM 77.90	Tilde 91.20
		musk (ver. 2)		
77.45 d_{FS}	70.59 (=)	Matchings 88.00	KeS 85.50	EM-DD 84.90
			MIK 88.00	DD 82.50
			mi-SVM 83.60	IAPR 84.90
			MI-SVM 84.30	
		duke		
78.05 d_{SMD}	70.73 (=)			
		mutagenesis		
87.23 d_T	87.23 (=)	Matchings 83.00	MIK 93.00	Progol 81.00
			K1 85.1 (loo)	Tilde 79.00
			K2 91.0 (loo)	Foil 61.00
			K3 91.5 (loo)	
			RK 85.4	
		FM		
63.32 d_T	57.31 (-)		K1 63.4 (loo)	
			K3 64.5 (loo)	
		FR		
66.67 d_L	64.10 (=)		K1 66.1 (loo)	
			K3 66.9 (loo)	
		MM		
65.18 d_H	62.80 (=)		K1 64.3 (loo)	
			K3 66.4 (loo)	
		MR		
61.63 d_{SMD}	61.92 (=)		K1 58.4 (loo)	
			K3 65.7 (loo)	
		Protein Fingerprints		
86.01 d_{SMD}	85.47 (=)		85.91	
82.25 (test set)			85.92 (test set)	

Table 3.7: Accuracy results of the best (k_{Best}) and CV (k_{CV}) distances together with performances of other relational systems. The references of the specific systems are given in the text.

selected for application is the one that has the highest estimated accuracy. Once a specific algorithm is selected it is retrained on the complete training set and the induced classification model is tested on the test data.

We adopt the same strategy in order to select the appropriate complex distance. That is, all complex distances are cross-validated on the training set and we choose for application the one that minimizes the estimated error. To measure the classification performance of the distance selection strategy we used the same 10-fold stratified cross-validation as in the previous experiments. For each of the training folds there is now a second, inner 10-fold stratified cross-validation[7] over all the complex distances that performs the distance selection. The results of this automatic distance selection are given in Table 3.7 under the entry kNN_{CV}. The representation of the graph dataset used to compute this automatic distance selection is the same as the one reported for the best complex distance measure. The sign in the parenthesis for kNN_{CV} indicates whether the performance of this method is significantly better (the + sign) or worse (the - sign) than that of kNN_{Best} or there is no significant difference (the = sign).

For most of the problems, with the exception of the diterpenes and mutagenesis, the performance of kNN_{CV} is different than that of kNN_{Best}, however, the differences in performances are not statistically significant. Moreover, the kNN_{CV} is usually lower than the one of kNN_{Best}, which makes sense since as already mentioned the performances of kNN_{Best} are optimistically biased. The differences in performances indicate that the actual distance measure selected within the internal cross-validation process is different from the one corresponding to kNN_{Best}. Finally, we mention that in diterpenes the performance of kNN_{CV} is the same as for best distance and better than that reported for any of the other relational systems.

Although the results from Table 3.7 show that this method is quite efficient in practice, its main limitation is that it selects only one complex distance. More elaborate methods, which are able to combine a number of existing distance measures will be presented in Chapter 5.

3.5.8 Parameter Selection for the Tanimoto Distance

In this work we introduced a modified version of the Tanimoto distance measure, d_T, from Equation 3.18, which can be used with graded similarities. This adaptation included the definition of a threshold parameter, θ, which in our previous experiments was a priori set to 0.01. We will take a closer look on how the value of this parameter affects the classification performance of d_T. In order to do that we estimated the classification error when θ was taking the following values: $0.001 \ldots 0.009$ with a step of 0.001; $0.01 \ldots 0.09$ with a step of 0.01; $0.1 \ldots 0.5$ with a step of 0.1. The results are depicted graphically in Figure 3.6. For some of the datasets (diterpenes, mutagenesis and carcinogenicity) the default value of the parameter is very close to the best achieved performance from

[7]The first inner 10-fold stratified cross-validation as already mentioned is performed over the k parameter.

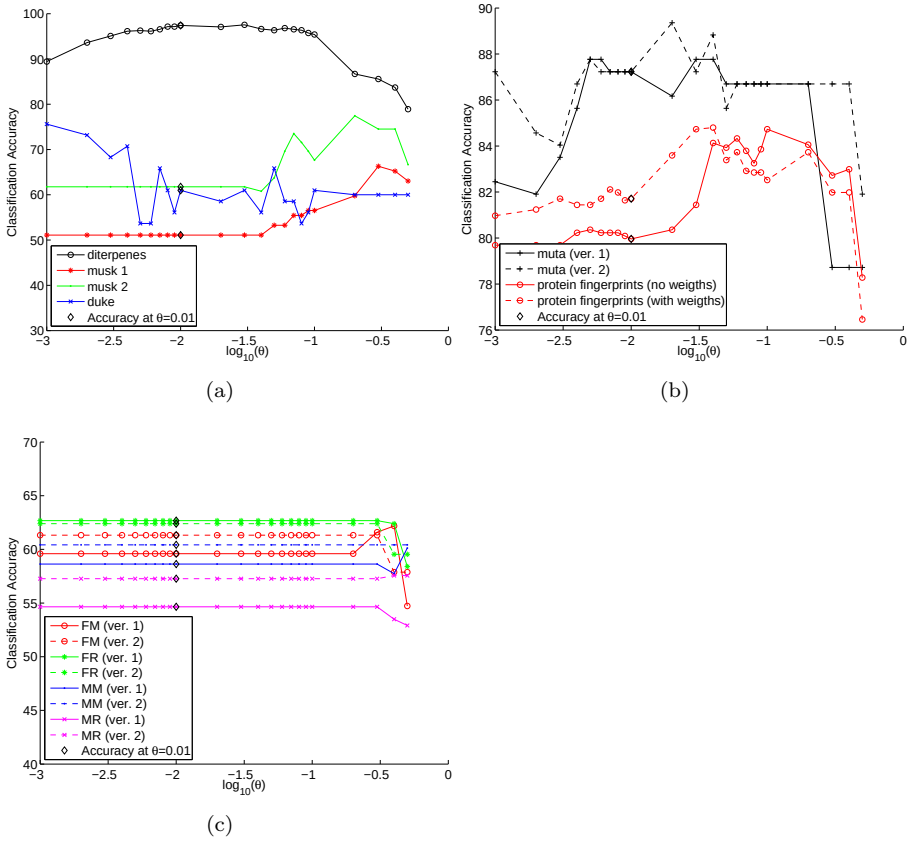


Figure 3.6: Behavior of the Tanimoto distance with respect to its θ parameter. Chart (a) presents datasets where instances are sets of vectors, chart (b) gives performance for the mutagenesis and protein fingerprints datasets, and chart (c) presents results for the carcinogenicity datasets.

which it has no significant difference (Table 3.8). However, for the two versions of musk this is not the case. Better performance can be achieved if one carefully tunes the parameter value, this could be done using internal cross-validation for parameter tuning.

Examining Figure 3.6 we can see that for different datasets there are different behaviors of the Tanimoto distance with respect to its θ parameter. For example, in the mass spectrometry problem (duke) the performance starts with high values and then it quickly falls, while in the carcinogenicity datasets the performance stays constant for values of θ in the range between 0.001 and approximately 0.2 (for larger θ the performance changes). The behavior of d_T in duke agrees with the technical characteristics of mass spectrometry, where the devices exploited to produce the samples have the measurement error which is

dataset	best θ	best accuracy	default θ accuracy	sig
diterpenes	0.03	97.54	97.41	=
musk (ver. 1)	0.3	66.30	51.09	+
musk (ver. 2)	0.2	77.45	61.76	+
duke	0.001	75.61	60.98	=
muta (ver. 1)	0.006,0.03	87.77	87.23	=
muta (ver. 2)	0.02	89.36	87.23	=
FM (ver. 1)	0.4	62.18	59.31	=
FM (ver. 2)	All except: 0.4,0.5	61.32	61.32	=
FR (ver. 1)	All except: 0.4,0.5	62.68	62.68	=
FR (ver. 2)	All except: 0.4,0.5	62.39	62.39	=
MM (ver. 1)	All except: 0.4,0.5	60.12	58.63	=
MM (ver. 2)	All except: 0.4,0.5	60.42	60.42	=
MR (ver. 1)	All except: 0.4,0.5	54.94	54.94	=
MR (ver. 2)	All except: 0.4,0.5	57.27	57.27	=
PFP not-weighted	0.1	84.73	79.96	+
PFP weighted	0.04	84.80	81.71	+

Table 3.8: Effect of parameter tuning on classification performance of the Tanimoto distance. The sign in the "sig" column indicates whether the performance of the best threshold value was significantly better than that of the default (the + sign) or there was no significant difference (the = sign).

approximately 0.001 (Prados et al., 2004). For the other datasets we can observe a very rough pattern: classification performance improves as we move away from very low values of θ, it then reaches some kind of plateau in which it gets its best values, and deteriorates as we move to higher values of θ. The position and size of the plateau depends on the classification problem. This behavior is somehow similar to the one observed in carcinogenicity where the plateau starts directly from very low values of θ and deteriorates for $\theta \geqslant 0.2$. This kind of pattern is quite reasonable if one thinks how the Tanimoto distance works. For very low values of θ few elements of the two sets, if any, will be matched. As the value increases more and more elements are matched to reach at some point an optimal matching level that depends on the problem at hand. As θ continues to increase even more elements are matched; at $\theta = 1.0$ all elements are matched (when normalized distances are used), this results in a Tanimoto distance that only depends on the cardinality of the two sets being compared, $d_T = \frac{|A|+|B|-2min\{|A|,|B|\}}{|A|+|B|-min\{|A|,|B|\}}$ (in this case $|A \cap B| = min\{|A|, |B|\}$, since we allow each element to be matched only once). If we assume that set A is the minimum cardinality set then $d_T = \frac{|B|-|A|}{|B|}$, which simply describes how larger is the cardinality of $|B|$ compared to the cardinality of $|A|$.

3.6 Related Work

In this section we will describe some of the distances defined in the literature which are most relevant to our work. More precisely, we report previous work on distances defined over sets, lists, trees and general complex structures. We conclude by commenting on kernel induced distances; kernels over composite objects will be the main focus in Chapter 4.

Distances on Sets

The central idea in set distance measures we have considered is the definition of a mapping of elements of one set to elements of the other set such that the final distance is determined on the basis of specific pairs of elements from the two corresponding sets. Different types of mappings correspond to distances that have different semantics. It should be noted that the other widely used approach for computing distances is based on averaging. For example, (Tatti, 2007) proposed a general set distance measure which is based on comparison of summary statistics (e.g. a proportion of a specific element) computed from the corresponding sets. The other class of distance measures based on averaging can be obtained from applying different divergence measures between the probability density functions (PDF) fitted to the elements of the corresponding two sets. In this framework we can use among others the Kullback-Leibler divergence, J-coefficient, χ^2-divergence, Hellinger coefficient and the family of L_p distances. If the data is assumed to be drawn from normal distributions with equal covariance matrices, the standard Mahalanobis distance between their means can be used (Pekalska and Duin, 2005). For an overview of different divergence measures the reader is referred to (Taneja, 1989; Pekalska and Duin, 2005). The main disadvantage with the above set distance measures is that they might be inappropriate for applications where only some elements from the two sets determine the overall similarity (e.g. multiple-instance learning). Finally, we mention that some affinity measures between distributions (Kondor and Jebara, 2003; Lyu, 2005a) have the property that the resulting similarities between sets are positive semi-definite (see Section 4.1) and as such they can be directly used in the context of kernel-based learning; these approaches will be discussed in some detail in Section 4.5.

Traditionally, the most widely used set distance measure based on mapping between sets of objects is the Hausdorff metric (Klein and Thompson, 1984) and its many variants (Huttenlocher et al., 1993; Zhao et al., 2005; Yang et al., 2007; Baudrier et al., 2004; Jesorsky et al., 2001), mainly used in the image processing and computer vision communities. In particular, some of the related distance measures presented by Dubuisson and Jain (1994) amount in fact to the d_{SMD} and d_{AL} distance measures; the others are generalized d_{SL} and d_{CL} where the corresponding k smallest (of largest) distances are selected. The other widely used set distance measures based on mappings are single linkage, complete linkage and average linkage. These distance measures, together with centroid linkage, median linkage and Ward's linkage are extensively used in computing set

distances in clustering.

Similarly to the above method, the evaluation of the inter-population dissimilarity could also rely on describing each (parametric) distribution as a point in a Riemann space with the coordinates specified by the distribution's parameters (Pekalska and Duin, 2005). Similar populations, i.e. populations which give rise to distributions with similar parameters, will be mapped into neighboring points in this space. This method can be further extended by defining the coordinates in the feature by partial derivatives of the log-likelihood of the distribution with respect to the model parameters. This construction of the gradient space, F, is in the core of the Fisher kernel (Jaakkola and Haussler, 1999) which is defined as the inner product in F (possibly normalized by the Fisher information matrix).

Distances on Lists

The idea of using the edit distance (Section 3.2.4) for general lists was proposed by Horváth et al. (2001). However, Horváth et al. (2001) applied this procedure only for lists over a symbolic alphabet whereas we apply it to the problem where elements of lists are general complex objects. The other difference is in the normalization method since Horváth et al. (2001) normalizes the distance by the maximum lengths of the two lists. In our case this distance measure is normalized by the number of elementary operations which transform one list into another. Nevertheless, in both cases the resulting distance measure in not a metric (see Example 3.5) since the triangle inequality does not hold.

Apart from the edit distance and its variants (Durbin et al., 1999) a number of other distances for strings have been proposed in the literature. For an overview of different string distances and string matching algorithms the reader is referred to (Gusfield, 1997; Stephen, 1994).

Another class of distance measure over binary sequences is based on the notion of *Kolmogorov complexity* of a string, s, which is defined as the length (in bits) of the shortest program (in a fixed computing system) that produces s (Bennett et al., 1998). Since the Kolmogorov complexity is not computable, an approximate distance measure based on a compressing program was proposed (Cilibrasi and Vitányi, 2005). Since any data objects can be represented as binary strings (after proper encoding) a Kolmogorov complexity-based distance measure can be applied for general complex objects. So far it has been applied to DNA sequences (Li et al., 2003), music pieces in MIDI format (Cilibrasi and Vitányi, 2005) and various time series datasets (Keogh et al., 2004).

Distances on Trees

The most widely used distance on labeled (both ordered and unordered) trees is the tree edit distance (Zhang and Shasha, 1989; Klein, 1998; Bille, 2005), which can be seen as an extension of the basic string edit distance from Section 3.2.4. The edit distance of two trees is the cost of the lowest cost sequence of such operations that turns the first tree into the second one. There are currently two main

algorithms for solving this problem for ordered trees: Zhang-Shasha (Zhang and Shasha, 1989) and Klein (Klein, 1998), both based on the dynamic programming. On the other hand for unordered trees the problem turns out to be NP-hard (Bille, 2005) and except for some restricted cases no efficient algorithms exist.

The main difference between the tree distance proposed in Section 3.3 and tree edit distance lies in the fact that the former tackles the problem using a local approach. When computing our distance the relevant information is considered in a recursive manner according to the levels of given trees. At each level only information from the next deeper level is used so that at the end all the relevant information is accounted for, but it does not take into account the structural properties of learning instances. On the other hand, the tree edit distance explicitly focuses the structural properties training objects. Of course, no general statement can be done about the superiority of one distance measure over another – it is application depend and should be guided by domain knowledge. The other difference is that the edit distance can be only applied on labeled trees which are ordered or unordered and it is not clear how to extend this distance to the generic trees we considered here. Finally, the computational complexity of the edit distance depends on the actual type of a tree (and it is different for ordered and unordered) whereas in our case the complexity is constant, even for trees which consist of both ordered and unordered parts. To our best knowledge no edit distance exists for "mixed" trees.

Distances on General Structures

A number of systems that perform distance based learning over general complex representations have appeared in the literature. The most directly related with our work are *KBG* (Bisson, 1992), *RIBL2.0* (Horváth et al., 2001), *FORC* and *RDBC* (Kirsten et al., 2001) with the latter two being based on the relational distance of *RIBL2.0*. In all of them the representation language is that of first order logic and the distance is computed recursively by taking into account all the information directly or indirectly related with two given instances and information that is found in deeper levels of the recursion has a lesser impact on the computation of the final distance. *RIBL2.0* is much closer to our system since it is used in a classification context the other three systems are clustering systems.

As in our system, *RIBL2.0* can handle terms and lists. However, unlike *RIBL2.0* our system is declarative in nature, and hence we are not constrained to a homogeneous distance measure but we can induce a variety of relational distance measures depending on the distance measures that are declared for different data types. We can even use different distance operators even for objects of the same general type. The above characteristics provide much greater flexibility since it is generally accepted that there is no distance measure that is adequate for all kinds of applications, a fact which also came clear out in the experimental comparison. One can now choose among a number of distance measures or even plug-in easily the one that is most appropriate for the application

at hand. Moreover, in what can be seen as model selection, the most appropriate distance measure can be selected via the use of internal cross-validation. In Chapter 5 we will also propose more elaborate methods, which are based on combination of a number of predefined distance measures. Finally, *RIBL2.0*'s distance measure is a dissimilarity measure while the properties of the finally induced relational distance of our system depend directly, as we have shown, on the distance measure that is used.

Other related work on distances in relational domains include the work of Hutchinson (1997) who defines pseudo-metrics between ground terms based on their distance from their least general generalization. These are then coupled with the Hausdorff distance in order to compute distances between clauses, which are nothing more than disjunctions of positive or negative atoms. Nienhuys-Cheng (1997) proposed a metric on ground terms computed recursively on the different levels of the structure of the terms. Sub-terms that are found deeper in the structure have a lower influence on the distance between the terms. Combining this metric with the Hausdorff distance she produced a metric over sets of ground terms, i.e. interpretations, which can be used to represent the learning examples. Both of the above do not account for variables or identifiers (identifiers correspond to foreign key relations in the context of relational algebra).

Ramon and Bruynooghe (1998) extended the above approaches based on interpretations so that they are able to handle variables and identifiers and proposed a general framework for the definition of distances between first order logic objects at three levels. At the first level a distance between atoms is required, then exploiting that distance a distance between models or clauses can be defined (models are sets of atoms and clauses sets of literals, i.e. positive or negative atoms). At the last level one has to account for the common terms, i.e. variables or identifiers, that appear within each set and appropriately adjust the final distance. This is done by selecting over all the computed distances between the two sets of atoms or literals under all possible renaming substitutions[8] of the variables and identifiers the one that is minimal. The main problem here comes from the computational complexity of the renaming substitutions which is exponential in the number of the variables or identifiers that are present within the sets.

More concretely, if learning examples are represented via sets of ground atoms, the distance between these sets will be computed using some set distance measure over the distance defined on atoms. In order to take into account the presence of identifiers this distance should be computed for each of the possible renaming substitutions of the identifiers. The final distance of two learning examples will be the minimal distance over all the renaming substitutions of their identifiers. To place the interpretation approach in the context of relational algebra in order to construct the interpretation of a given relational instance we would have to flatten the tree structure of the relational instance to its corresponding set of atoms, each tuple of a relation present within the tree structure

[8]The renaming substitution amounts to trying to find the best match between the variables and identifiers of the two sets.

of the relational instance will result in one atom of the interpretation. The main difference between the recursive computation of the relational distance as it is done in *RIBL2.0* and our system, and the approach of Ramon and Bruynooghe (1998) is that in the latter all atoms of an interpretation contribute equally to the final distance between two relational examples while in the two former the contribution of a given tuple-atom reduces with the depth in which it is found in the recursion. The recursive computation of the distance is an approximate computation whose quality increases with the depth of the recursion while the approach of (Ramon and Bruynooghe, 1998) is an exact computation; however the cost that one has to pay is the computation of the optimal matching of identifiers which is exponential to the number of identifiers. In fact, because of that one has often to resolve to an approximate computation as it is done by Ramon (2002, chapter 5) for the mutagenesis problem.

Finally, Sebag (1997) proposes a system called *DISTILL*, where a set of disjunctive hypotheses, possibly redundant, are constructed using disjunctive version spaces. These hypotheses are used in a subsequent step to map examples onto a new space where each dimension of that space corresponds to a given disjunctive hypothesis and its value is the number of disjunctions that subsume a given example. The distance between two relational instances is simply the euclidean distance in that space. The induced distance is a pseudo-metric. This approach has some commonalities with the kernel approaches that we are going to see in Chapter 4, there the examples are also mapped in a new space in which a semi-metric can be induced on the basis of the kernel.

Probably the most important difference from all of the previous work on complex distances is that we are not limited to a single distance measure. All previous systems implement a single relational distance measure. Instead our system offers a variety of distance measures among which the user can choose the one that better matches the problem at hand. In the case that such knowledge is not available distance selection based on cross validation can be employed. Furthermore the system is implemented in such a way that it is straightforward to incorporate new distances. Last but not least the presence of different distances allows for their simultaneous use in a single relational problem. For example, the user can specify that for one set of tuples a given set distance measure should be used, while for other sets of tuples a different distance should be used. This results in a final relational distance that is heterogeneous and matches better the problem requirements; none of the previous work offers that flexibility.

Kernel-Based Distances

Another way of defining distances is by exploiting kernels over structured domains (kernels over complex objects will be presented in Chapter 4). A kernel over a structured domain can induce a distance measure which is actually a pseudo-metric. There is a growing literature on kernels defined over structured objects such as sets, lists and general complex objects (for an overview see Section 4.5). Most of the work in that area is based on a decomposition of the structured objects to sets of their subparts (Haussler, 1999), the computation

of kernels on these subparts, and the combination of these kernels in a way that the resulting function is still a valid kernel, i.e., positive semi-definite. This combination (and hence a kernel over sets) is defined by taking the sum of kernels over *all* the possible pairs of subparts of the two structured objects. Compared to distances over sets one limitation of kernels on sets is that they consider all pairs of elements (in order to guarantee positive definiteness), while as we have seen in distances over sets it is possible to define mappings of elements resulting in greater flexibility. Nevertheless, the advantage of kernels is that they can be used within a support vector machine (Section 4.4.1) which can potentially result in higher classification accuracy and speed compared to the nearest neighbor classifier; the latter due to the small number of support vectors that will be established resulting in a small number of kernel computations. The limitation of existing kernels between sets will be addressed in Section 4.2.4 where a new class of kernels based on specific elements from the two sets is considered.

3.7 Conclusions

In this chapter we proposed various distance measures defined over different building blocks (i.e. primitive attributes, tuples, sets and lists) of composite objects represented using the relational formalism. The distances over general composite objects are based on the decomposition of the objects to simpler parts of various types and the combination of distances on these simpler parts. Our system is not constrained to a specific distance measure over the learning instances. The practitioner has freedom in assigning a distance measure, selected from a set of available distances, to a particular data type. Moreover, it is possible to use different distance operators even for objects of the same general type.

From the experimental evidence it is clear that classification performance critically depends on the choice of the distance measure, which should be guided by domain knowledge. The right choice of distance measure gives encouraging classification results that in many cases compare favorably with those of other relational learners reported in literature. Moreover, by exploiting a cross-validation based distance selection, in what amounts to model selection, we obtain a relational learner whose performance is comparable to the performance of the a-posteriori selected best distance measure.

The central idea in set distance measures we have considered is the definition of a mapping of elements of one set to elements of the other set such that the final distance is determined on the basis of specific pairs of elements from the two corresponding sets. Different types of mappings correspond to distances that have different semantics. The characterization of the various instantiations of the set distance measures was supported by the empirical results, i.e. set distances that were semantically similar were in general also similar in terms of their relative performance on the relational benchmarks examined.

The various distance measures proposed in this chapter form a basis for the material in the remaining part of this study. In particular the various mappings

of elements exploited by different set distance measures will be used to define novel kernels over sets (Chapter 4). Moreover, in Chapter 5 we will propose a method for learning how to combine a set of predefined distance measures.

Chapter 4

Kernels

In this chapter we consider kernel functions, the second class of operators defined over the relational representation. Kernels are fundamental ingredients of kernel-based methods (Shawe-Taylor and Cristianini, 2004; Schölkopf and Smola, 2001; Cristianini and Shawe-Taylor, 2000), the most famous and successful example of which is the Support Vector Machine (SVM). Traditionally, kernel-based methods have been applied on vectorial data, however, starting from the seminal work of Haussler (1999) it became clear that this class of algorithms has the potential to support input spaces whose representation is more general than attribute-value. The examples of existing kernels defined on composite data include kernels on sets (Kondor and Jebara, 2003; Woźnica et al., 2006a), sequences (Leslie et al., 2003), trees (Collins and Duffy, 2002), labeled graphs (Gärtner et al., 2003; Kashima et al., 2003; Ramon and Gärtner, 2003; Menchetti et al., 2005) and general structured data (Gärtner et al., 2004).

For good generalization abilities of kernel-based algorithms it is necessary that the kernel function reflects the underlying semantics of the data and incorporates problem-specific a priori knowledge, if such exists. However, this has been shown to be difficult in practice and in particular defining kernels over composite representations remains a challenging task. To facilitate the design of composite kernels we propose a general method for constructing kernel operators following the syntactic structure of the complex learning instances as defined by their relational representations. Similarly to composite distances defined in the previous chapter the proposed kernels are based on decompositions of complex structures into subparts of various types that are compared via appropriate kernels. The practitioner has freedom in declaring which specific kernel, selected from a set of available kernels, should be used for a particular data type. The final kernel over the complex learning instances is given as a recursive combination of kernel functions assigned to the sub-structures which constitute the learning instances. Ideally, this combination should be defined in a way that the resulting kernel is positive semi-definite such that the data can be mapped to a (pre-)Hilbert feature space where all the kernel-based algorithm is implicitly applied. The above mentioned modular nature of the proposed kernels provides

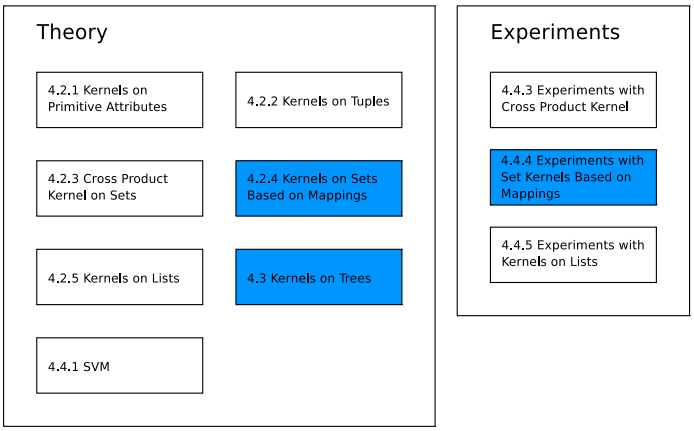


Figure 4.1: Schematic structure of Chapter 4. The sections with our main contributions are highlighted.

with the analyst the maximum flexibility in defining a kernel which reflects the application requirements.

Many of the kernels defined over structured data such as sets, graphs and sequences are based on (usually implicit) decompositions of these complex structures into substructures of different types. The final kernel that is applied on the resulting decompositions is the cross product kernel computing the average of similarities given by the sub-kernels applied on every pair of elements from the two decompositions. This feature might be inappropriate in cases where only specific elements of decompositions are important for a problem at hand. To address the above problem we propose a family of set kernels which are not based on averaging; instead they take into account the specific pairs of elements. We examine the formal properties of the proposed kernels and evaluate them empirically. To the best of our knowledge it is the first time that such general kernels over complex data are considered.

The remaining part of this chapter is organized as follows. In Section 4.1 we review some of the terminology, definitions and theorems used to characterize the kernels explored in this work. In Sections 4.2.1–4.2.5 we formally define different kernels over the building blocks of relational instances. In particular, in Section 4.2.1 we present kernels over elementary objects and in Section 4.2.2 we define various kernels on tuples. In Section 4.2.3 we review and analyze the cross product kernel which is probably the most widely used kernel on sets, while in Section 4.2.4 we propose three families of set kernels which are based on specific pairs elements from the two sets. Finally, in Section 4.2.5 we define kernels over lists. In Section 4.3 we move to composite objects and describe kernels on trees. In the experiments (Section 4.4) we exploit the Support Vector Machines (Section 4.4.1) to empirically analyze the proposed kernels. The description of experiments is divided into three parts: in Section 4.4.3 we examine the

performance of the standard cross product kernel, in Section 4.4.4 we analyze the performance of other, more flexible set kernels, and finally in Section 4.4.5 we report the experimental results where we exploit various kernels on lists. In Section 4.5 we place our complex kernels in the context of related work. Finally, we conclude with Section 4.6.

The most important of our theoretical contributions from this chapter are the theoretical presentation of the flexible set kernels based on mappings (Section 4.2.4) and the kernel on trees (Section 4.3). The most important experimental results are presented in Section 4.4.4; we argue there that when dealing with set problems the standard approaches based on averaging do not necessarily provide the best performance. The organization of this chapter is schematically presented in Figure 4.1.

4.1 Preliminaries

In this section we review some of the terminology, definitions and basic theorems which are relevant to different kernel functions used in this study. We start with the definition of a *kernel* function.

Definition 4.1 (Kernel)**.** A *kernel* is a symmetric function $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$, i.e.

$$\forall x, y \in \mathcal{X} \ : \ k(x, y) = k(y, x)$$

Most of this chapter will be devoted to kernel functions which are *positive semi-definite*.

Definition 4.2 (Positive Semi-Definite Kernel)**.** We call a kernel $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ *positive semi-definite* (PSD) iff for all $x, y \in \mathcal{X}$

$$k(x, y) = \langle \phi(x), \phi(y) \rangle_{\Phi}$$

where ϕ is a mapping from $\mathcal{X}$ to a feature space Φ embedded with an inner product $\langle \cdot, \cdot \rangle_{\Phi}$, i.e. a *pre-Hilbert* space.

Sometimes we will use the term *valid* kernel to denote a PSD kernel. A kernel which is not PSD will be also referred to as an *indefinite* kernel. It should be stressed that the definition of a PSD kernel does not require that the input space $\mathcal{X}$ is a vectorial space – it can be any set which we can embed in the feature space Φ via a PSD kernel. The attractiveness of kernels lies in the fact that one does not need to explicitly compute the mappings $\phi(x)$ in order to compute the inner products in the feature space. It is easy to show the following two theorems (Shawe-Taylor and Cristianini, 2004).

Theorem 4.1. *A function $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is a positive semi-definite kernel iff for any $x_1, \ldots, x_n \in \mathcal{X}$ the induced kernel matrix $\boldsymbol{K} = (k(x_i, x_j))_{i,j}^n$ is positive semi-definite according to Definition 3.12 in Section 3.2.2.*

Theorem 4.2. *A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is positive semi-definite iff for any $x_1, \ldots, x_n \in \mathcal{X}$ the induced kernel matrix $\boldsymbol{K} = (k(x_i, x_j))_{i,j}^n$ has non-negative eigenvalues.*

The kernel matrix is sometimes referred to as the *Gram matrix*. In the remaining part of this chapter we will mainly use kernels which are *normalized*. The most popular way to normalize a valid kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is given by

$$k(x, y) := \frac{k(x, y)}{\sqrt{k(x, x)k(y, y)}} \tag{4.1}$$

Obviously, for the above kernel $k(x, x) = 1$ holds. Assuming that ϕ is a mapping to a feature space Φ associated with kernel k, it is straightforward to find a transformation for the normalized kernel of Equation 4.1 (Shawe-Taylor and Cristianini, 2004).

Proposition 4.1. *Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a valid kernel function that corresponds to the feature mapping ϕ to a feature space Φ. Then the* normalized kernel *defined in Equation 4.1 corresponds to the following feature map*

$$\phi(x) := \frac{\phi(x)}{\|\phi(x)\|_\Phi} \tag{4.2}$$

where $\|\cdot\|_\Phi$ denotes the norm in Φ.

The above normalization procedure is general and can be applied to any kernel. It can be shown that this normalization method computes the cosine of the angle between the two corresponding vectors in the feature space. Other normalization procedures are possible, however they depend on the particular kernel and will be introduced in the following sections. In any case, in order to render kernels defined over various composite objects directly comparable, we require that $\forall x, y \in \mathcal{X}\ k(x, y) \leqslant 1$. Moreover, it is desirable that $k(x, x) = 1$.

We also note that for any PSD kernel defined on $\mathcal{X}$, a pseudo-metric $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_0^+$ can be induced by

$$\begin{aligned} d^2(x, y) &= \|\phi(x) - \phi(y)\|_\Phi^2 \\ &= \langle \phi(x) - \phi(y), \phi(x) - \phi(y) \rangle_\Phi \\ &= \langle \phi(x), \phi(x) \rangle_\Phi - 2\langle \phi(x), \phi(y) \rangle_\Phi + \langle \phi(y), \phi(y) \rangle_\Phi \\ &= k(x, x) - 2k(x, y) + k(y, y) \end{aligned} \tag{4.3}$$

Additionally d is a metric if ϕ is injective (Schölkopf and Smola, 2001).

The above discussion suggests that apart from considering kernels as inner products in some feature space Φ, these functions can be also regarded as measures of similarity, in the sense that $\forall x, y \in \mathcal{X}\ k(x, y)$ is "large" when x and y are "similar" (Vert and Schölkopf, 2004). This can be seen from Equation 4.3 which shows that the kernel $k(x, y)$ measures similarity between x and y as the opposite of the square distance $d^2(x, y)$ between their images in the feature space (up to the terms $k(x, x) = \|\phi(x)\|_\Phi^2$ and $k(y, y) = \|\phi(y)\|_\Phi^2$). In particular if $\forall x \in \mathcal{X}\ k(x, x) = constant$ then the kernel is simply a decreasing measure of the distance in the feature space (Vert and Schölkopf, 2004).

4.2 Kernels over Relational Building Blocks

After defining the basic concepts we move to definitions of various kernel operators on building blocks of relational instances. In particular, in Section 4.2.1 we present kernels over primitive data types, while in Section 4.2.2 we define various kernels over tuples. Different set kernels will be considered in Sections 4.2.3 and 4.2.4. Finally, in Section 4.2.5 we focus on kernels over lists.

4.2.1 Kernels on Primitive Attributes

In this section we review kernels for the most basic objects, i.e. numerical and symbolic data types.

We start by considering numerical data types. Let x, y be objects of numeric data type in range $[a, b]$ $(a < b)$, i.e. $type(x) = type(y) = numeric(a, b)$. The kernel defined over numerical data types we used in this work is defined as follows.

Definition 4.3. A kernel between two numerical values $x, y \in \mathbb{R}$ is defined as

$$k_{num}(x, y) = x \cdot y \tag{4.4}$$

Let $\mathcal{X}$ be a finite set of symbolic values and let x, y be objects of symbolic data type, i.e. $type(x) = type(y) = symbolic(S)$ for some finite set of identifiers S. The most widely used kernel over symbolic values, and the one used in this study, is the *δ kernel*.

Definition 4.4 (δ kernel)**.** For two objects $x, y \in \mathcal{X}$ we define the *δ kernel* $k_\delta : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ as

$$k_\delta(x, y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise.} \end{cases} \tag{4.5}$$

The above kernel is associated with the $\ell^2(\mathcal{X})$ Hilbert space indexed by all the elements of $\mathcal{X}$ (assuming that $\mathcal{X}$ is countable) (Shawe-Taylor and Cristianini, 2004).

4.2.2 Kernels on Tuples

In this section we will first consider the *Direct Sum Kernel* and the *Tensor Product Kernel* (Schölkopf and Smola, 2001) which are defined over tuples where elements are general structured objects. Then, we will briefly review some of the well-known kernels defined over vectors in the $\mathbb{R}^n$ Euclidean space. In comparison with kernels over general tuples, the existing vector kernels give more freedom in handling non-linearities in the data.

Let $\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_n$, not necessary vectorial spaces, be sets on which the corresponding kernel functions $k_1, k_2, \ldots, k_n$ are defined, i.e. $k_i : \mathcal{X}_i \times \mathcal{X}_i \to \mathbb{R}$ for $i = 1, \ldots, n$. Moreover, let $\boldsymbol{\mathcal{X}} = \mathcal{X}_1 \times \mathcal{X}_2 \times \cdots \times \mathcal{X}_n$ be the set of all tuples $(x_1, x_2, \ldots, x_n)$ such that $x_i \in \mathcal{X}_i$. The first kernel we define on $\boldsymbol{\mathcal{X}}$ is the *Direct Sum Kernel*.

Definition 4.5 (Direct Sum Kernel)**.** Let $\boldsymbol{x}, \boldsymbol{y} \in \boldsymbol{\mathcal{X}}$. The kernel function $k_\Sigma : \boldsymbol{\mathcal{X}} \times \boldsymbol{\mathcal{X}} \to \mathbb{R}$ defined as

$$k_\Sigma(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{n} k_i(x_i, y_i) \tag{4.6}$$

is called the *Direct Sum Kernel*.

It is straightforward to show that the direct sum Kernel is a valid kernel (Cristianini and Shawe-Taylor, 2000).

Proposition 4.2. *The direct sum kernel is a valid kernel over* $\boldsymbol{\mathcal{X}} = \mathcal{X}_1 \times \mathcal{X}_2 \times \cdots \times \mathcal{X}_n$ *provided that all kernels* k_i *are valid, for* $i = 1, \ldots, n$.

It is obvious that k_Σ is affected by the number of the constituent elements of tuples. In order to factor out this effect we use a normalized version of k_Σ defined as

$$k_\Sigma(\boldsymbol{x}, \boldsymbol{y}) := \frac{k_\Sigma(\boldsymbol{x}, \boldsymbol{y})}{n} \tag{4.7}$$

The other considered kernel over tuples is the *Tensor Product Kernel*.

Definition 4.6 (Tensor Product Kernel)**.** Let $\boldsymbol{x}, \boldsymbol{y} \in \boldsymbol{\mathcal{X}}$. The kernel function $k_\Pi : \boldsymbol{\mathcal{X}} \times \boldsymbol{\mathcal{X}} \to \mathbb{R}$ is defined as

$$k_\Pi(\boldsymbol{x}, \boldsymbol{y}) = \prod_{i=1}^{n} k_i(x_i, y_i) \tag{4.8}$$

is called the *Tensor Product Kernel*.

Similarly to k_Σ, the tensor product kernel is PSD provided all the sub-kernels are PSD (Cristianini and Shawe-Taylor, 2000).

Proposition 4.3. *The tensor product kernel is a valid kernel over* $\boldsymbol{\mathcal{X}} = \mathcal{X}_1 \times \mathcal{X}_2 \times \cdots \times \mathcal{X}_n$ *provided that all kernels* k_i *are valid, for* $i = 1, \ldots, n$.

Again, it is obvious that the value of k_Π is affected by the number of constituent elements. We opted for the feature space normalization from Equation 4.1 resulting in the following kernel

$$k_\Pi(\boldsymbol{x}, \boldsymbol{y}) := \frac{k_\Pi(\boldsymbol{x}, \boldsymbol{y})}{\sqrt{k_\Pi(\boldsymbol{x}, \boldsymbol{x}) k_\Pi(\boldsymbol{y}, \boldsymbol{y})}} \tag{4.9}$$

In order to get an insight into the behavior of k_Σ and k_Π we analyze the feature space associated with these two kernels. Let ϕ_Σ (ϕ_Π) be an embedding function into a feature space Φ_Σ (Φ_Π) for the kernel k_Σ (k_Π). Let also $\phi_1, \ldots, \phi_n$ be embedding functions into feature spaces $\Phi_1, \ldots, \Phi_n$ of the kernels $k_1, \ldots, k_n$ which constitute the two kernels k_Σ and k_Π. It is easy to show that $\Phi_\Sigma = \Phi_1 \oplus \ldots \oplus \Phi_n$ and $\Phi_\Pi = \Phi_1 \otimes \ldots \otimes \Phi_n$ where $\oplus$ denotes the direct sum and $\otimes$ denotes the tensor product of vector spaces (Schölkopf and Smola, 2001). In

other words, ϕ_Π is constructed by computing all the possible products of all the dimensions of its constituent spaces, where each product becomes a new dimension of Φ_Π. In contrast Φ_Σ is constructed by a simple concatenation of the dimensions of its constituent spaces. In case the spaces Φ_i $(i = 1, \ldots, n)$ have a finite number of dimensions (denoted by $dim(\Phi_i)$), it is obvious that $dim(\Phi_\Sigma) = \sum_{i=1}^{n} dim(\Phi_i)$ and $dim(\Phi_\Pi) = \prod_{i=1}^{n} dim(\Phi_i)$. In order to get the explicit feature space representation induced be the above kernels one has to recursively combine the feature spaces induced by a kernel on tuples' subparts (which are possibly complex themselves).

An important result of the above discussion is that the feature space induced by the tensor product kernel, Φ_Π, is more expressive since it accounts for feature interactions by means of the corresponding products. From the practical point of view the larger feature space induced by k_Π might be beneficial for "complex" problems, while "simple" problems can be best solved in the smaller feature space generated by direct sum version, k_Σ. Additionally, distance-based learning in the feature space induced by k_Π is expected to be more difficult than in the one induced by k_Σ. This is because the dimensionality of Φ_Π is much higher than Φ_Σ[1], and hence distance-based methods can be affected by the "curse of dimensionality" (Hastie et al., 2001) and susceptible to over-fitting. On the other hand, algorithms based on the concept of large margin regularize the solution and are potentially less sensitive to the high dimensionality of Φ_Π.

It should be noted that the two above kernels on tuples are specific applications of the $\Re$-Convolution kernel of Haussler (1999), which is one of the best known kernels defined over non-vectorial data. The main idea in the $\Re$-Convolution kernels is that composite objects consist of simpler parts that are related with the objects via a relation $\Re$. In other words the relation $\Re$ defines how subparts are related to each other and form the composite objects. Kernels on composite objects can be then computed by combining kernels defined on their constituent parts. More formally let $x \in \mathcal{X}$ be a composite object and $\vec{\boldsymbol{x}} = (x_1, \ldots, x_D) \in X_1 \times \ldots \times X_D$ its particular decomposition into constituent parts. Then we can represent the relation "$\vec{\boldsymbol{x}}$ are the parts of x" by the relation $\Re$ defined over the set $X_1 \times X_2 \times \ldots \times X_D \times \mathcal{X}$ where $\Re(\vec{\boldsymbol{x}}, x)$ is true iff $\boldsymbol{x}$ are the parts of x. Let $\Re^{-1}(x) = \{\vec{\boldsymbol{x}} : \Re(\vec{\boldsymbol{x}}, x)\}$ and we assume that a composite object can have more than one decomposing possibilities. Then the $\Re$-Convolution kernel is defined as

$$k_\Re(x, y) = \sum_{\boldsymbol{x} \in \Re^{-1}(x), \boldsymbol{y} \in \Re^{-1}(y)} \prod_{d=1}^{D} k_d(x_d, y_d) \tag{4.10}$$

where $k_d : X_d \times X_d \to \mathbb{R}$ $(d = 1, \ldots, D)$ is a kernel function defined on complex objects' subparts. It should be noted that the product operator in the above equation, which is in fact the tensor product of kernels, can be replaced by other operators that need to be closed with respect to kernel positive

[1]This holds if the elementary kernels induce a feature space of finite dimensionality, otherwise they are both of infinite dimension.

semi-definiteness. An example of such other operator is a direct sum operator resulting in $k_{\Re}(x, y) = \sum_{\boldsymbol{x} \in \Re^{-1}(x), \boldsymbol{y} \in \Re^{-1}(y)} \sum_{d=1}^{D} k_d(x_d, y_d)$. The $k_{\Re}$ kernel can be applied to tuples: since there is only one way to decompose a tuple, the sum in the Equation 4.10 vanishes and we obtain the product of kernels defined over elements of tuples, hence the tensor product kernel. By using the direct sum operator we obtain the direct sum kernel.

The main advantage of the $\Re$-Convolution kernel is that it is very general and as such can be applied for various complex data. On the other hand it is not always easy to adapt this kernel to a problem at hand (Gärtner et al., 2002).

Kernel over Vectors in Euclidean Spaces

In this section we will briefly review some of the kernels defined over the $\mathbb{R}^n$ Euclidean space. As already mentioned, the reason we consider vector kernels as a special case of kernels over general tuples, is that the former allow for treating the data non-linearities in a more principled way. Probably the simplest kernel defined over $\mathbb{R}^n$ is the *Linear Kernel*.

Definition 4.7 (Linear Kernel)**.** Let $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$. The linear kernel is defined as

$$k_{lin}(\boldsymbol{x}, \boldsymbol{y}) = \langle \boldsymbol{x}, \boldsymbol{y} \rangle \tag{4.11}$$

The above kernel can be easily derived by applying the direct sum kernel over general tuples from Equation 4.6 where all the corresponding sub-kernels are k_{num} from Equation 4.4. In such case the linear kernel can be equivalently written as

$$k_{lin}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{n} k_{num}(x_i, y_i) \tag{4.12}$$

The other widely used kernel over $\mathbb{R}^n$ is the *Polynomial Kernel*.

Definition 4.8 (Polynomial Kernel)**.** Let $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$. Moreover, let $l \in \mathbb{R}$ and $p \in \mathbb{N}$ ($p \geqslant 2$) be two parameters. The *Polynomial Kernel* $k_{poly} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is defined as

$$k_{poly}(\boldsymbol{x}, \boldsymbol{y}) = (\langle \boldsymbol{x}, \boldsymbol{y} \rangle + l)^p \tag{4.13}$$

The feature space corresponding to this kernel is constructed by taking products of the original features. More precisely for $l = 0$ the feature space has dimensions indexed by all monomials (of input features) of degree p; the corresponding features do not receive equal weights in this embedding (Cristianini and Shawe-Taylor, 2000). In case $l > 0$ the features are monomials up to and including degree p. Hence, the l parameter can be considered a bias towards lower-order monomials. Additionally, the relative weighting of the higher order polynomials is decreased as l increases. The following proposition determines the dimensionality of the feature space induced by the polynomial kernel (Shawe-Taylor and Cristianini, 2004).

Proposition 4.4. *The dimensionality of the feature space for the polynomial kernel* $k_{poly}(\boldsymbol{x}, \boldsymbol{y}) = (\langle \boldsymbol{x}, \boldsymbol{y} \rangle + l)^p$ *for* $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ *and* $l > 0$ *is* $\begin{pmatrix} n+d \\ d \end{pmatrix}$.

Finally, we consider the *Gaussian RBF Kernel*.

Definition 4.9 (Gaussian RBF Kernel)**.** Let $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$. Given the parameter $\gamma \in \mathbb{R}$ the *Gaussian RBF* kernel $k_{RBF} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is defined as

$$k_{RBF}(\boldsymbol{x}, \boldsymbol{y}) = e^{-\gamma \|\boldsymbol{x}-\boldsymbol{y}\|^2} \tag{4.14}$$

The corresponding embedding $\phi_{k_{RBF}}$ into the feature space $\Phi_{k_{RBF}}$ maps the points onto a surface of a hyperball. The feature space $\Phi_{k_{RBF}}$ has an infinite number of dimensions since any two images are orthogonal and any set of images is linearly independent (Schölkopf and Smola, 2001). The Gaussian RBF kernel can be also considered as a polynomial kernel of infinite degree where the features in $\Phi_{k_{RBF}}$ are all possible monomials without restriction on the degrees. By the Taylor expansion of the exponential function $e^x = \sum_{i=0}^{\infty} \frac{1}{i!} x^i$ it is clear that the importance of individual monomials in k_{RBF} decreases as $i!$ with increasing degree (Shawe-Taylor and Cristianini, 2004).

The parameter γ acts in a similar way to the p parameter in the polynomial kernel. In particular for large values of γ, which correspond to large values of p, the resulting kernel matrix becomes similar to identity matrix. In such cases a kernel-based classifier can fit any labels, and hence the generalization is harmed (Schölkopf et al., 2002). On the other hand, for small values of γ the kernel becomes a constant function, making it difficult to learn (Shawe-Taylor and Cristianini, 2004).

4.2.3 Cross Product Kernel on Sets

In this section we describe different variants of the *Cross Product (CP) Kernel* which is one of the simplest, yet widely used kernels defined over (multi-)sets. This kernel can be considered as a direct application of the $\Re$-Convolution kernel of Equation 4.10. The CP kernel (and the $\Re$-Convolution kernel) plays a fundamental role in the definition of kernels for structured domains. This is a result of the fact that an integral part of many kernels for complex objects is the *decomposition* of these structures into multi-sets of their parts (e.g. sub-sequences in case of strings; walks, sub-trees, etc. in case of graphs), and the final kernel is defined precisely as the CP kernel between the corresponding multi-sets of decompositions (Haussler, 1999; Horváth et al., 2004).

Definition 4.10 (Cross Product Kernel)**.** Let $A = \{a\} \subseteq \mathcal{X}$ and $B = \{b\} \subseteq \mathcal{X}$ be two non-empty and finite sets and let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a (valid) kernel. The *Cross Product (CP) Kernel* between sets is defined as

$$k_{CP}(A, B) = \sum_{a \in A, b \in B} k(a, b) \tag{4.15}$$

It should be noted that in case the arguments to the above kernel are multi-sets, the summation on the right hand of the above equation takes into account the sets' multiplicity. It is straightforward to show the following proposition.

Proposition 4.5 (PSD-ness of the Cross Product Kernel)**.** *The cross product kernel is a valid kernel function over* $2^{\mathcal{X}} \times 2^{\mathcal{X}}$ *provided* $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ *is a valid kernel over* $\mathcal{X} \times \mathcal{X}$*.*

Proof. This kernel is an instantiation of the $\Re$-Convolution kernel from Equation 4.10. More precisely by defining the relation $\Re$ as a set-membership, i.e. $\boldsymbol{x} \in \Re^{-1}(x) \Leftrightarrow \boldsymbol{x} \in x$ and by setting $D = 1$ we obtain the desired result. □

The procedure of computing the CP kernel is sensitive to cardinality variations; sets with larger cardinality will dominate the overall solution. This leads us to the issue of normalization of the CP kernel, so that we obtain

$$k_{CP}(A, B) := \frac{k_{CP}(A, B)}{f_{norm}(A) f_{norm}(B)} \tag{4.16}$$

where $f_{norm}(\cdot)$ is a normalization function which is non-negative and takes non-zero values. Different choices of $f_{norm}(\cdot)$ give rise to different normalization methods (Gärtner et al., 2002). By putting $f_{norm}(\cdot) = |\cdot|$ we obtain the *Averaging* normalization method. By defining $f_{norm}(\cdot) = \sqrt{k(\cdot, \cdot)}$ we get the normalization in the feature space. This amounts to the normalized kernel from Equation 4.1. The two above kernels are valid since for both of them the explicit representation of the feature space can be constructed.

Similarly to the analysis performed in Section 4.2.2 we will examine the feature space associated with the normalized CP kernel from Equation 4.16. We assume ϕ_k is an embedding function into a feature space Φ_k for the kernel k on the right hand of Equation 4.15 so that $\forall a, b \in \mathcal{X}\ k(a, b) = \langle \phi_k(a), \phi_k(b) \rangle_{\Phi_k}$. Consider the following embedding function for a finite set A

$$\phi_{set}(A) = \sum_{a \in A} \phi_k(a) \in \Phi_{set} \tag{4.17}$$

It is easy to show that ϕ_{set} is an embedding function into the feature space Φ_{set} for the CP kernel (Shawe-Taylor and Cristianini, 2004). Similarly, the feature space induced by the kernel from Equation 4.16 where $f_{norm}(A) = |A|$ is given by $\phi_{set}(A) = \frac{\sum_{a \in A} \phi_k(a)}{|A|}$. In other words this normalization method amounts to computing the inner product, in the feature space induced by the kernel k, between the two centroids of the corresponding sets. Similar to Equation 4.2, in the case $f_{norm}(A) = \sqrt{k(A, A)}$, the feature space is given by

$$\phi_{set}(A) = \frac{\sum_{a \in A} \phi_k(a)}{\| \sum_{a \in A} \phi_k(a) \|} \tag{4.18}$$

The computational complexity of the CP kernel between two finite sets A and B is proportional to $O(|A||B|)$. If we assume that the elementary kernel is a polynomial kernel with the exponent p (with the bias towards lower order monomial) and input space is $\mathbb{R}^n$, then this complexity is proportional to $O(|A||B|(n + p))$. It is interesting to compare this computational complexity

with that of the inner product computed directly in the feature space, which is proportional to $O\{2\binom{p+n}{p}(|A| + |B|)\}$, i.e. each point has to be mapped to $\binom{p+n}{p}$ – dimensional feature space $(|A| + |B|)$ times and the computation of the inner product in the feature space is again proportional to $\binom{p+n}{p}$. For example, if we put $n = 10$, $p = 2$ and $|A| = |B| = 100$ then the computation of the cross product kernel requires approximately five times more operations than the inner product in the feature space. As a result, we can see that in some cases it is more efficient to explicitly map the instances to the feature space and compute the inner product.

The CP kernel of Equation 4.16 is simple to compute and proved to be quite useful in practice. However, its main disadvantage is that it takes into account *all* the elements of the two sets. In fact, this kernel can be shown to compute an inner product between two means of the corresponding probability density functions in the feature space induced by the kernel over the sets' elements. This feature might be inappropriate for applications where only some elements from the two sets determine the overall similarity. In Section 4.2.4 we will address this problem by proposing kernels for sets which are not based on averaging, instead they only take into account similarities between specific pairs of elements from the two sets.

4.2.4 Kernels on Sets Based on Mappings

As already mentioned the set kernels which are based on averaging might not be suitable for some learning problems. In this section we will propose three new and flexible families of kernels over sets, where the overall similarity is based only on specific elements of the two sets. The proposed kernels are based on the various families of mapping introduced in Section 3.2.3. Depending on the chosen family of mappings we get set kernels with different semantics which can be then used within the regularization framework possibly increasing the predictive performance over methods where distances are used in a standard way (e.g. kNN). These set kernels can be divided into three groups: (i) kernels defined as the sum of elementary kernels between specific pairs of elements, (ii) set distance substitution kernels, where the set distances are substituted using the Gaussian RBF similarity measure (Haasdonk and Bahlmann, 2004), and (iii) kernels in the proximity space induced by set distances where the mapping is defined by a given representation set (Pekalska and Duin, 2005). While the kernels in the first group exploit the families of mapping in a direct way, the kernels in the second and third group are based on set distance measures and exploit the various mappings indirectly.

We should mention that although we use the distance substitution kernels and kernels in proximity spaces to define kernels over sets, these kernels are not limited to be only used with set distance measures. In fact, we can exploit virtually any distance measure to obtain kernels over general structured domains.

The main problem with kernels of the first group and the distance substitution kernels is that they are not PSD in general, however, encouraged by

recent experimental and theoretical results on the application of Support Vector Machines with non-PSD kernels, we are able to use such kernels with the corresponding theoretical framework (see Section 4.4.1). In Section 4.4.4 where we report experimental results we will see that the performance of the SVM algorithm with kernels based on specific pairs of elements compares favorably to the SVM with kernels based on averaging.

The remaining part of this section is organized as follows. In Section 4.2.4 we defined kernels directly based on mappings, in Section 4.2.4 the distance substitution kernels and in Section 4.2.4 we defined the kernels in the proximity space.

Kernels on Sets Based on Mappings

The first class of kernels based on specific pairs of elements from the two sets is characterized by the fact that the mapping between sets is defined in the feature space associated with kernels on sets' elements. The considered mappings are the same as the ones used to define set distance measures in Section 3.2.3. Unfortunately, all the kernels defined in this section are not valid, i.e. they are not positive semi-definite.

Let $A = \{a\} \subseteq \mathcal{X}$ and $B = \{b\} \subseteq \mathcal{X}$ be two non-empty and finite sets. Let also $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ be a (valid) kernel. The set kernel k_{set} over $2^{\mathcal{X}}$ is defined as

$$k_{set} : 2^{\mathcal{X}} \times 2^{\mathcal{X}} \rightarrow \mathbb{R},\ k_{set}(A, B) = f(\{k(a, b) | (a, b) \in A \times B\}).$$

where k is a kernel on $\mathcal{X}$. The above general form of the set kernel is similar to the one of the set distance measure from Equation 3.9 defined as some function of the pairwise distances, $d(a, b)$, of the set of all pairs $(a, b) \in A \times B$; here the set kernel is defined as a function, f, of the set of pairwise kernels, $k(a, b)$. The actual functions f have the same semantics to the ones used to compute set distance measures. Finally, all the kernels defined in this section are normalized by the number of matched elements defined by the mapping.

Definition 4.11 (Single Linkage Kernel)**.** The *Single Linkage Kernel* is defined as the maximum kernel of all pairwise kernels

$$k_{SL}(A, B) = \max_{(a,b) \in A \times B} \{k(a, b)\} \tag{4.19}$$

Example 4.1. A counter example for the PSD-ness is to consider $A = \{a, b\}, B = \{c, d\}, C = \{a, c\}$ with k being the δ kernel from Equation 4.5. We have $k_{SL}(A, A) = k_{SL}(B, B) = k_{SL}(C, C) = 1$, $k_{SL}(A, B) = 0$, $k_{SL}(A, C) = 1$ and $k_{SL}(B, C) = 1$. The corresponding Gram matrix is the following

$$\boldsymbol{K}_{SL} = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

However, it eigenvalues are $\lambda_1 \approx -0.41, \lambda_2 = 1, \lambda_3 \approx 2.41$, hence the k_{SL} kernel is not PSD.

Definition 4.12 (Complete Linkage Kernel)**.** The *Complete Linkage Kernel* is defined as the minimum kernel of all pairwise kernels

$$k_{CL}(A, B) = \min_{(a,b)\in A\times B}\{k(a, b)\} \tag{4.20}$$

Example 4.2. A counter example for the PSD-ness is to consider $A = \{a, b\}$, $B = \{a\}$ with k being the δ kernel from Equation 4.5. We have $k_{CL}(A, A) = 0$, $k_{CL}(B, B) = 1$ and $k_{CL}(A, B) = 1$. The corresponding Gram matrix is the following

$$\boldsymbol{K}_{CL} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$$

but its eigenvalues are $\lambda_1 \approx -0.61, \lambda_2 \approx 1.62$, hence k_{CL} is not PSD.

Definition 4.13 (Sum of Maximum Kernels)**.** The *Sum of Maximum Kernels* set kernel is defined as the sum of the maximum kernels of the elements of the first set to the elements of the second set and vice versa:

$$k_{SMD}(A, B) = \frac{\sum_{a\in A}\max_{b\in B}\{k(a, b)\} + \sum_{b\in B}\max_{a\in A}\{k(b, a)\}}{|A| + |B|} \tag{4.21}$$

Example 4.3. A counter example for the PSD-ness of k_{SMD} is taken from (Lyu, 2004). Consider $A = \{a, b\}$, $B = \{c, d\}$ and $C = \{e, f\}$. We assume that the kernel on set $\mathcal{X} = \{a, b, c, d, e, f\}$ computed by some PSD kernel k is given by the following Gram matrix

$$\boldsymbol{K} = \begin{pmatrix} 127 & 127 & 141 & 60 & 159 & 128 \\ 127 & 287 & 215 & 127 & 236 & 135 \\ 141 & 215 & 206 & 101 & 223 & 157 \\ 60 & 127 & 101 & 73 & 134 & 79 \\ 159 & 236 & 223 & 134 & 281 & 191 \\ 128 & 135 & 157 & 79 & 191 & 160 \end{pmatrix} \tag{4.22}$$

The above matrix is PSD (i.e. has non-negative eigenvalues) and hence by Theorem 4.1 the kernel k is a valid kernel over $\mathcal{X}$. From $\boldsymbol{K}$, the k_{SMD} kernel can be computed as $k_{SMD}(A, A) = 828$, $k_{SMD}(A, A) = 614$, $k_{SMD}(C, C) = 944$, $k_{SMD}(A, B) = 698$, $k_{SMD}(A, C) = 766$ and $k_{SMD}(B, C) = 737$. However, the resulting Gram matrix

$$\boldsymbol{K}_{SMD} = \begin{pmatrix} 828 & 698 & 766 \\ 698 & 614 & 737 \\ 766 & 737 & 944 \end{pmatrix}$$

is not PSD since its lowest eigenvalue is $\lambda_1 \approx -0.0071$.

Definition 4.14 (Hausdorff Kernel)**.** The *Hausdorff set kernel* is defined as

$$k_H(A, B) = \min\left(\min_{a\in A}\{\max_{b\in B}\{d(a, b)\}\}, \min_{b\in B}\{\max_{a\in A}\{d(b, a)\}\}\right) \tag{4.23}$$

Example 4.4. Similar to Example 4.3 consider $A = \{a, b\}$, $B = \{c, d\}$ and $C = \{e, f\}$. Let also the matrix from Equation 4.22 be a Gram matrix defined on $\mathcal{X} = \{a, b, c, d, e, f\}$. The Gram matrix for k_H is the following:

$$\boldsymbol{K}_H = \begin{pmatrix} 127 & 127 & 135 \\ 127 & 101 & 134 \\ 135 & 134 & 191 \end{pmatrix}$$

However, it has the following eigenvalues: $\lambda_1 \approx -14.9$, $\lambda_2 \approx 25.81$ and $\lambda_3 \approx 408.1$, and hence k_H is not PSD.

Definition 4.15 (RIBL Kernel)**.** The *RIBL* set kernel is given as the sum of the maximum kernels of the elements of the smaller set to the elements of the larger set, normalized by the cardinality of the smaller set

$$k_{RIBL}(A, B) = \begin{cases} \frac{\sum_{a \in A} \max_{b \in B}\{k(a,b)\}}{|A|} & \text{if } |A| < |B| \\ \frac{\sum_{b \in B} \max_{a \in A}\{k(a,b)\}}{|B|} & \text{otherwise} \end{cases} \tag{4.24}$$

Since it is not symmetric is has to be symmetrized e.g. as $k_{RIBL}(A, B) := \frac{1}{2}(k_{RIBL}(A, B) + k_{RIBL}(B, A))$. It is easy to see that this kernel is equivalent (up to a normalization terms) to k_{SMD}, and hence in the following we will focus only on the latter. Moreover, it is not PSD – the counterexample is the same as in Example 4.3.

We also define set kernels which are based on relations $R = \{R_i | R_i \subseteq A \times B\}$ between the two sets. The computation of the set kernel will be based on the $R_i \in R$ that maximizes a sum of kernels computed on the elements that are part of the relation R_i (or equivalently minimizes the sum of inverse kernels). The relations R considered in this work are surjections, fair surjections, linkings and matchings.

For the R being a set of surjections we can define *Surjections Kernel*, k_S, as

$$k_S(A, B) = \frac{\max_{R_i \in R} \sum_{(a_i, b_j) \in R_i} k(a_i, b_j)}{|R_i|}.$$

Similarly, for the R being a set of fair surjections we can define *Fair Surjections Kernel*, k_{FS}, as

$$k_{FS}(A, B) = \frac{\max_{R_i \in R} \sum_{(a_i, b_j) \in R_i} k(a_i, b_j)}{|R_i|}.$$

When R is a set of linkings *Linkings Kernel*, k_L. is defined as

$$k_L(A, B) = \frac{\max_{R_i \in R} \sum_{(a_i, b_j) \in R_i} k(a_i, b_j)}{|R_i|}.$$

Finally, the corresponding set kernel derived from the Matchings family of mappings is defined as

$$k_M(A, B) = \frac{\max_{R_i \in R}(\sum_{(a_i, b_j) \in R_i} k(a_i, b_j) + (|B - R_i(A)| + |A - R_i^{-1}(B)|) \times \frac{M}{2})}{|A| + |B|}$$

where R is a set of matchings.

We also consider a kernel which measures degree of overlapping between the two sets. In this sense this kernel is related to the Tanimoto distance of Definition 3.25.

Definition 4.16 (Intersection Kernel)**.** For two sets A, B the (normalized) *Intersection Kernel* is defined as

$$k_{\cap}(A, B) = \frac{|A \cap B|}{|A| + |B|} \tag{4.25}$$

The above kernel (defined over symbolic sets) is a valid kernel since it is a special case of the CP Kernel where the kernel over sets' elements is the matching kernel of Equation 4.5. Similar to the Tanimoto distance we will extend this kernel to be able to deal with graded similarities. More precisely, we will consider two $a \in A$ and $b \in B$ identical if $k(a, b) \geqslant 1 - \theta$ where θ is a user specified threshold parameter and $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ a valid, normalized kernel on sets' elements. Unfortunately, for such loose definition of identity the resulting kernel is not PSD. We could not find any simple counterexample, however, for the datasets we considered $k_{\cap}$ gives rise to non-PSD Gram matrices (see Section 4.4.4)[2].

It should be noted that the kernel exploiting the d_{AL} set distance measure amounts to the well known cross product kernel kernel on sets discussed in Section 4.2.3. Finally, we mention that the computational complexity of the kernels defined in this section depends on the complexity of computing the corresponding mappings and are the same as presented in in the context of set distances in Table 3.2. In the next two sections we will propose a class of set kernels where distance on sets are used explicitly. We start by defining the *Distance Substitution Kernel*.

Distance Substitution Kernels

Distance Substitution Kernels have been echoing in the literature for some time (Chapelle et al., 1999; Bahlmann et al., 2002; Haasdonk and Keysers, 2002; Moreno et al., 2004; Belongie et al., 2002b; Desobry et al., 2005; Chen, 2004; Hein et al., 2004; Chapelle and Zien, 2005) but only recently were they introduced in a general form in (Haasdonk and Bahlmann, 2004; Haasdonk, 2005a).

Definition 4.17 (Distance Substitution Kernel)**.** Let $k : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ be a valid kernel that can be written in the form of $k(\|\boldsymbol{x} - \boldsymbol{y}\|)$ and $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_0^+$ is a distance measure which is at least a distance function, i.e. it is nonnegative, zero-diagonal and symmetric. Then for any $x, y \in \mathcal{X}$ the *Distance Substitution (DS) Kernel* is defined as

$$k_{DS}(x, y) = k(d(x, y)).$$

[2]The same comment applies to the kernels based on relations, i.e. k_S, k_{FS}, k_L and k_M.

The fact that we do not place any constraints of the underlying distance measure (except that it should be a distance function) makes the DS kernels widely applicable. In particular, the DS kernels are of direct use in these areas of data mining/machine learning where a wide range of different distance measures already exist.

Such a general family of kernels naturally recovers most of the standard vectorial kernels such as the Gaussian RBF and the negative-distance kernels (Berg et al., 1984). It should be also mentioned that similar to the DS kernel it is possible to generalize any kernel of the form $k(\boldsymbol{x}, \boldsymbol{y}) = k(\langle \boldsymbol{x}, \boldsymbol{y} \rangle)$ for $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ such that we obtain a class of kernels based on inner-products[3]. Examples of such kernels are the standard linear and polynomial kernels.

In this work we focus on the generalized Gaussian RBF kernel, which has been widely used and proved to be quite effective in various structured pattern recognition problems (see e.g. Chapelle et al., 1999; Chen, 2004). More precisely, the generalized Gaussian RBF kernel is defined as

$$k_{DS}(x, y) = k_{RBF}(d(x, y)) = e^{-\gamma d(x,y)^2} \tag{4.26}$$

for any $x, y \in \mathcal{X}$, $\gamma \in \mathbb{R}^+$ and where d is at least a distance function. The intuition beyond the above kernel is that it is expected to reflect the same behavior as the standard Gaussian RBF kernel, such as nonlinearity, etc. Moreover, this kernel takes values between $[0, 1]$ and hence it does not have to be normalized.

Some statements about the PSD-ness of the kernel from Equation 4.26 can be made. In particular, the following proposition holds (Haasdonk and Bahlmann, 2004).

Proposition 4.6. *For any distance measure $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_0^+$ which is at least a distance function, the kernel from Equation 4.26 is PSD for any $\gamma \in \mathbb{R}^+$ iff d is isometric to an L_2-norm (Definition 3.8).*

An intermediate result of the above proposition is the following conjecture.

Conjecture 4.1. *For $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_0^+$ which is not a metric (L_2-norms are in particular metrics) the kernel from the Equation 4.26 is not PSD.*

In particular, from the results presented in Table 3.1, we conclude that for d_{AL} (after making it reflexive), d_{SL}, d_{CL}, d_{SMD}, d_{RIBL} (after converting it to be symmetric), d_T, d_S, d_L and d_{FS} set distance measures, the resulting kernel $k_{DS}(x, y)$ is not PSD. For d_H and d_M, even though these distance measures are metrics, the resulting kernels are not PSD since counterexamples can be found[4].

[3] In fact, the distance substitution and inner-product-based kernels are related since any symmetric distance measure d together with the choice of an origin $O \in \mathcal{X}$ induces a generalized inner product by $\langle x, y \rangle_d^O := -\frac{1}{2}(d(x, y)^2 - d(x, O)^2 - d(y, O)^2)$ for $x, y \in \mathcal{X}$. In particular, for d being a Hilbertian metric (Equation 3.8), $\langle x, y \rangle_d^O$ corresponds to the inner product in the corresponding space with respect to the origin O (Haasdonk, 2005a).

[4] In the datasets we considered, and for d_H and d_M the DS kernel results in non-PSD Gram matrices (see Section 4.4.4).

Kernels in Proximity Spaces

In the last method we consider the learning instances are represented in the so called proximity space (Pekalska and Duin, 2005; Pekalska et al., 2001; Graepel et al., 1999). This space is defined by a given distance measure and a representation set of learning instances (i.e. set of prototypes). More precisely, given a representation set $S = \{s_1, \ldots, s_n\} \subseteq \mathcal{X}$ and a distance measure $d : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}_0^+$ we define a mapping $\boldsymbol{d}(z, S) : \mathcal{X} \rightarrow \mathbb{R}^n$ as

$$\boldsymbol{d}(z, S) = [d(z, s_1), \ldots, d(z, s_n)]^T$$

where $z \in \mathcal{X}$. Since distance measures are non-negative, all the data examples are projected to a non-negative orthotope of the $\mathbb{R}^n$ vector space. The dimensionality of this space is controlled by the size of the set S (usually the full training set).

The construction of the proximity space is justified by the fact that for an object s_i belonging to the same class as z, $d(z, s_i)$ should be small while for an object s_j of different classes $d(z, s_j)$ should be large, resulting in a set of features with possibly high discrimination power (Pekalska et al., 2001). On the other hand, if s_i is a characteristic object of a particular class, then the feature $d(x, s_i)$ has a large discrimination power, while for s_j being an outlier, $d(x, s_j)$ may discriminate poorly. It should be noted that the mapping $d(z, S)$ transforms instances to a vector space where any traditional machine learning algorithm such as kNN and SVM may be used.

A direct approach exploiting the dissimilarity information given by by d leads to the k-Nearest Neighbor method (Section 3.5.1). For a given test instance z this rule is applied to $d(z, s_i)$ ($s_i \in S$) such that z is classified to the most frequent class occurring among the k neighbors in S. As already mentioned in Section 3.5.1, the main disadvantages of this algorithm are large storage and computational requirements as well as sensitivity to outliers. Moreover, if the data is sparsely sampled and the underlying distance measure is not a metric, the classification performance of the kNN algorithm may significantly differ from its asymptotic behavior. It has been argued (Pekalska and Duin, 2005) that the above problems could be alleviated precisely by representing the data in the proximity space. As a result, the classification performance in the proximity space is expected to be higher than the one in the "initial" space of input objects.

The definition of a kernel in the proximity space amounts to choosing a distance measure d and a vectorial kernel k in the induced space. The resulting Gram matrix of the kernel k_P consists of the elements

$$(\boldsymbol{K}_{P,d})_{ij} = k(\boldsymbol{d}(x_i, S), \boldsymbol{d}(x_j, S)) \tag{4.27}$$

It should be stressed that the above kernel is PSD iff k is PSD, independently of the characteristics of the corresponding set distance measure. We also mention that in practice we require that k is normalized.

The kernel of Equation 4.27 is similar to the kernels based on similarity measures between distributions and linear subspaces defined over sets (Lyu,

2005a; Wolf and Shashua, 2003; Kondor and Jebara, 2003), in the sense that there too sets are first transformed to some other spaces. For kernels based on distributions the "elementary" kernel is defined in a space of (parametric) distributions whereas for the latter all the operations take place in a space of linear subspaces. In our case the sets are mapped to a vectorial space whose dimensionality is given by the cardinality of the representation set.

In the proximity space the final decision rules are functions of dissimilarities of *all* the elements of a given representation set S, usually the full training set, as a result the final classification models are not sparse. The above issue can be cast as a problem of prototype selection (Wilson and Martinez, 2000) which in our context amounts to applying a feature selection method which returns a set of optimal features (prototypes) according to some class separability measure. We will see in Section 5.4 that it is indeed possible to reduce the size of the representation set without harming the performance of our kernels.

Finally, the computational complexity of the kernels defined in this section depends on the complexity of computing the corresponding set distances (Table 3.2) multiplied by the the size of the representation set S.

4.2.5 Kernels on Lists

In this section we define two kernels over lists of general complex objects that will be used in the rest of this chapter. A natural kernel to consider at this point would be an extension of the edit distance between lists. However, this kernel has already been considered in (Jean-Philippe Vert, 2004), where it was empirically observed it is not PSD in general. The first kernel we define is the *Contiguous Sublist Kernel*.

Definition 4.18 (Contiguous Sublist Kernel)**.** Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a kernel defined over $\mathcal{X}$. Let $\lambda \in \mathbb{R}^+$, $\lambda < 1$ be a parameter. Then the *Contiguous Sublist Kernel* over all lists where elements are from $\mathcal{X}$, i.e. $\mathcal{L}(\mathcal{X} \cup -)$, is defined as

$$k_{CS}(\ell_1, \ell_2) = \sum_{\boldsymbol{i},\boldsymbol{j},l(\boldsymbol{i})=l(\boldsymbol{j})} \lambda^{l(\boldsymbol{i})} \sum_{s=1,\ldots,l(\boldsymbol{i})} k(\ell_1[\boldsymbol{i}_s], \ell_2[\boldsymbol{j}_s]) \tag{4.28}$$

where the subsequences $\boldsymbol{i}$ and $\boldsymbol{j}$ are assumed to be *contiguous* and $l(\boldsymbol{i})$ denotes the length of $\boldsymbol{i}$.

This kernel is a modified version of the *Contiguous Subtree Kernel* from (Zelenko et al., 2003). It works by first enumerating all contiguous sublists of equal length and aggregating their similarities which are calculated by adding the similarities of the elements of the sublists. The similarities between sublists decrease by the factor $\lambda^{l(\boldsymbol{i})}$, i.e. the role of the λ parameter is to penalize longer sublists. Finally, the similarity of two lists is the sum of all possible contiguous sublists of the two input lists.

A slightly more general kernel is proved to be a valid kernel (Zelenko et al., 2003). Moreover, the computational complexity of this kernel is $O(mn)$ where m and n are the lengths of lists ℓ_1 and ℓ_2, respectively.

The other kernel on lists we experimented with is a specialized version of the kernel over basic terms from (Gärtner et al., 2004) which we call the *Longest Common Sublist Kernel*.

Definition 4.19 (Longest Common Sublist Kernel)**.** Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a kernel defined over $\mathcal{X}$. Then the *Longest Common Sublist Kernel* over all lists where elements are of type $\mathcal{X}$, i.e. $\mathcal{L}(\mathcal{X} \cup -)$, is defined as

$$k_{LCS}(\ell_1, \ell_2) = m + \sum_{s=1}^{m} k(\ell_1[s], \ell_2[s]) + n \tag{4.29}$$

where $m = min(l(\ell_1), l(\ell_2))$, i.e. the length of the shortest list, and $n = 1$ if the lists are of the same length and 0 otherwise.

This kernel is proved to be a valid kernel (Gärtner et al., 2004). Its computational complexity is $O(m)$.

The underlying notion of similarities of k_{CS} and k_{LCS} is different. In the former the overall similarity is measured by sum of the similarities of all the (consecutive) sublists of the same length. The similarity between the sublists are computed by means of kernels defined over the elements of the lists. On the other hand, k_{LCS} takes only the longest common contiguous sublist at the start of the two input lists into account. Then a penalty term is added if the input lists have different lengths. In this sense k_{CS} takes a more "global" view and it is expected to perform better in practice. In order to use the above kernels we normalize them using feature space normalization (Equation 4.1).

4.3 Kernels on Trees and Tree-like Structures

In the previous section we proposed various kernel operators defined over the different building blocks of our relational objects. Similar to Section 3.3 we will now define kernels, $k_{tree}(T_i, T_j)$, over two trees (or tree-like structures), $T_i, T_j \in \mathcal{T}$, defined in Section 2.3, which are constructed as a particular combination of the basic structures. The reason we consider kernels over trees as a separate case of kernels over general composite objects is that the former structures will be widely exploited to represent labeled graphs. The main difference from kernels over general composite objects and k_{tree} is that in the latter we put some additional constraints on the actual kernel operators applied over basic structures. Finally, we focus here on unordered trees where the labels, $\mathcal{L}_{\mathcal{V}}$ and $\mathcal{L}_{\mathcal{E}}$, are vectors in an Euclidean space. The extension of the presented ideas to ordered trees (and to other, more general trees) is straightforward.

Let x, y, be two elements of the two trees found at the same height h. These elements are of the same type and are either two vertices, u, v, or two edges e_i, e_j. We also assume that the kernel k_{tuple} defined over tuples is given and is either the normalized direct sum kernel k_Σ from Equation 4.7 or the normalized tensor product kernel k_Π (Equation 4.9). Then the kernel between x and y is defined as

$$k(x, y) = k_{tuple}\left((lab(x), \delta(x)), (lab(y), \delta(y))\right) \tag{4.30}$$

where $(lab(x), \delta(x))$ and $(lab(y), \delta(y))$ are tuples consisting of two elements: the first is a label of a vertex (or an edge) and the second is defined through the $\delta(x)$ neighborhood function, described in Section 2.3, that returns either: the set of edges to which a vertex connects to as a starting vertex, if x is a vertex, or the vertex to which an edge arrives, if x is an edge.

The above kernel requires the definition of a kernel on the labels of x, y and on the sets $\delta(x), \delta(y)$. In the former case we exploit one of the k_{vector} kernels defined in Section 4.2.2. To compute the kernel over sets we use the cross product kernel of Equation 4.16 (normalized using either of the methods), however other set kernels can be exploited in this context. When x or y are leafs the $\delta(\cdot)$ function will return an empty set. k_{CP} should be possible to apply when one or both $\delta(x), \delta(y)$ are empty. We assume that k_{CP} applied on two empty sets returns a maximum value (i.e. 1), when only one of the sets is empty k_{CP} returns 0. To summarize, the definition of k applied on two elements x, y requires the definition of a kernel over vectors, k_{vector}, the normalization of k_{CP}, and finally the definition of a kernel defined over tuples, k_{tuple}. Different choices of the above kernels will give rise to different kernels defined over x and y.

k is a recursive kernel requiring the computation of set kernels between the elements of the trees that are associated to x, y, at the $h + 1$ height. The final kernel between two recursive structures, T_i, T_j, is given by

$$k_{tree}(T_i, T_j) = k(root(T_i), root(T_j)) \tag{4.31}$$

From the above discussion it is obvious that the computation of k_{tree} requires recursive and alternating computations of kernels between nodes and edges.

In order to show the computational complexity of the proposed tree kernel we use BF to denote the maximal out-degree of the considered trees, i.e. $\max_{v \in T(\mathcal{V}), T \in \mathcal{T}}\{|\delta(v)|\}$. It is easy to show that the computation of the relational kernel between two trees of height h is proportional to $O((BF^2)^{h-1}) = O(BF^{2(h-1)})$ (here we assume that the root of a tree is at level 1). This is a result of the fact that at level k we have to compute at most BF^{k-1} cross product kernels, each of which has complexity $O(n^2)$ (n is a cardinality of the corresponding sets). This is the pessimistic estimate of the time complexity a more accurate estimate would be acquired if the average branching factors were used.

4.4 Experiments

In the experiments we will compare different instantiations of the complex kernels defined in the previous sections on a number of relational problems. This section is divided into five parts. First, in Section 4.4.1 we briefly describe the Support Vector Machines classification method that we will use in the experiments. Then, in Section 4.4.2 we describe the experimental setup. Section 4.4.3 examines the performance of the standard cross product kernel, k_{CP}. In Section 4.4.4 we analyze the performance of the more flexible set kernels proposed in Section 4.2.4 which are based on specific pairs of elements from the two sets.

Finally, in Section 4.4.5 we report the experimental results for the problem of protein fingerprints classification where we exploit various kernels on lists.

4.4.1 Support Vector Machines

Probably the most prominent and widely used class of kernel-based methods in supervised settings are the Support Vector Machines (SVMs). In the remainder of this section we assume that the training data is in the form $(\mathcal{X}, \mathcal{Y}) = \{(x_i, y_i)\}_{i=1}^n$ where $\mathcal{X}$ denotes a set of input individuals and $\mathcal{Y}$ denotes a set of output labels. For the moment we only consider binary problems where the learning instances are either positive or negative, i.e. $\mathcal{Y} = \{-1, 1\}$. Assuming that $\boldsymbol{x}_i \in \mathbb{R}^m$ for some $m \in \mathbb{N}$ $(i = 1, \ldots, n)$, SVMs learn a linear decision boundary to discriminate between the two classes. This decision boundary is then used to classify new test examples. In such case the support vector machine is the hyperplane $f(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b$ maximizing the margin $\frac{1}{\langle \boldsymbol{w}, \boldsymbol{w} \rangle}$ between two separable classes (or equivalently minimizing $\langle \boldsymbol{w}, \boldsymbol{w} \rangle$) where $\boldsymbol{w} \in \mathbb{R}^m$ is a normal vector to the hyperplane and $b \in \mathbb{R}$ is the *bias*. A new point $\boldsymbol{x} \in \mathbb{R}^m$ is classified as positive (negative) if $f(\boldsymbol{x}) > 0$ $(f(\boldsymbol{x}) < 0)$.

In case the corresponding sets cannot be correctly separated by a linear hyperplane, the above problem is modified such that we obtain the following quadratic optimization problem (Cristianini and Shawe-Taylor, 2000)

$$\begin{aligned} \min_{\boldsymbol{w}, b, \xi_1, \ldots, \xi_n} \quad & \langle \boldsymbol{w}, \boldsymbol{w} \rangle + C \sum_{i=1}^n \xi_i \\ \text{subject to} \quad & y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b) \geqslant 1 - \xi_i, \\ & \xi_i \geqslant 0, i = 1, \ldots, n \end{aligned} \tag{4.32}$$

where the parameter $C > 0$ controls the trade-off between empirical risk minimization and margin.

Following the standard way of formulating a dual form for the Problem 4.32 we obtain (Boyd and Vandenberghe, 2004; Cristianini and Shawe-Taylor, 2000)

$$\begin{aligned} \operatorname*{argmin}_{\boldsymbol{\alpha} \in \mathbb{R}^n} \quad & \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n y_i y_j \alpha_i \alpha_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle \\ \text{subject to} \quad & \sum_{i=1}^n y_i \alpha_i = 0, \\ & C \geqslant \alpha_i \geqslant 0, i = 1, \ldots, n \end{aligned} \tag{4.33}$$

From the above optimization problem we see that learning a linear classifier only involves the points in the training set through their inner products. By replacing the inner products with a valid kernel function, the SVM can be implicitly applied in the feature space induced by the kernel. Hence, the task from Equation 4.33 can reformulated as the following optimization problem,

represented in the matrix notation

$$\begin{aligned} \underset{\boldsymbol{\alpha}\in\mathbb{R}^n}{\operatorname{argmin}} \quad & \mathbf{1}^T\boldsymbol{\alpha} - \frac{1}{2}\boldsymbol{\alpha}^T(\boldsymbol{y}^T\boldsymbol{K}\boldsymbol{y})\boldsymbol{\alpha} \\ \text{subject to} \quad & \boldsymbol{y}^T\boldsymbol{\alpha} = 0, \\ & C\mathbf{1} \geqslant \boldsymbol{\alpha} \geqslant 0, i = 1, \ldots, n \end{aligned} \tag{4.34}$$

where $\boldsymbol{K}_{ij} = k(x_i, x_j)$ and $\boldsymbol{y} = (y_1, \ldots, y_n)^T$ and $\mathbf{1}$ denotes the vector of length n consisting of the all 1's. In the above case $x_i, x_j \in \mathcal{X}$ can be general complex objects as long as a proper kernel function k is defined.

Multi-class SVMs

The above formulation of SVM can only handle binary classification problems. In order to deal with multi-class learning problems we reformulate the learning task as a number of binary classification problems, and solve these problems with binary SVM. The resulting SVMs are then combined to form a multi-class prediction algorithm. Several different combinations schemes have been proposed in the literature (Weston and Watkins, 1999); among them the best known are *one-against-rest* and *one-against-one*. In this work we used the latter scheme where one classifier is learned for each possible pair of classes (Friedman, 1996). More precisely, assuming c is the number of classes, for each of the $\frac{c(c-1)}{2}$ two-class-combinations, a binary SVM is trained, and then during classification each SVM is used to vote for a particular class. A class with maximum number of votes is selected. The main advantage is that the resulting binary problems are more balanced than in the *one-against-rest* scheme. The main disadvantage of this method is that the number of binary classifiers increases quadratically with the number of classes. Moreover, depending on the actual distribution of classes the learning tasks can be still unbalanced, and hence difficult to solve.

SVMs with Indefinite Kernels

Traditionally the SVM algorithm was applied only if the underlying Gram matrix (i.e. matrix $\boldsymbol{K}$ in Equation 4.34) is PSD, resulting in a convex, local-optimum free optimization problem which is amenable to various efficient optimization algorithms. However, recent experimental (Haasdonk and Bahlmann, 2004; Lin and Lin, 2003; Bahlmann et al., 2002; Haasdonk and Keysers, 2002; Chapelle et al., 1999) and theoretical (Haasdonk, 2005b; Ong et al., 2004; Lin and Lin, 2003) results state that even kernels which are not PSD can be used within the SVMs. The theoretical arguments supporting the use of indefinite kernels in SVMs are three-fold. First, the indefinite kernels can be interpreted as inner products in pseudo-Euclidean (Haasdonk, 2005b; Pekalska et al., 2001) or more general Krein spaces (Ong et al., 2004). Second, SVMs with indefinite kernels can be interpreted as optimal hyperplanes in these indefinite spaces (Haasdonk, 2005b). Third, the uniqueness of the solution of the SVMs can be guaranteed (Haasdonk, 2005b).

Several criteria can be used to determine how suitable a given non-PSD kernel is for SVMs (Haasdonk, 2005b). One of the widely used tests of how difficult it is to obtain a suitable solution with SVMs is to examine the spectrum of a non-PSD kernel matrix. The more negative eigenvalues it has, the more difficult it is to obtain good generalization with SVMs. This criterion can be formalized by defining the ratio r of the negative to the positive eigenvalue sum

$$r(\boldsymbol{K}) = \frac{\sum_{\lambda_i<0} |\lambda_i|}{\sum_{\lambda_i>0} \lambda_i} \in [0, \infty) \tag{4.35}$$

where $\lambda_1, \ldots, \lambda_n$ are eigenvalues of a Gram matrix $\boldsymbol{K}$. Ideally, this quantity is 0 for a positive definite matrix, whereas it goes to infinity, if a matrix has only negative eigenvalues. We will use this test in experiments (Section 4.4.4) to characterize the indefinite kernels from Sections 4.2.4 and 4.2.4. It should be mentioned that other indicators can be computed; more details can be found in (Haasdonk, 2005b).

The other approach to dealing with non-PSD kernels is to regularize the kernel matrix by eliminating its negative eigenvalues. Possible approaches to address this problem include: (i) an uniform shifting of the spectrum of the eigenvalues (without changing the eigenvectors), (ii) removing negative eigenvalues by thresholding, (iii) reflecting negative eigenvalues by taking their absolute values or (iv) adding a suitable constant to the off-diagonal elements of the induced (squared) distance matrix (Haasdonk and Bahlmann, 2004). The main problem with the above procedures it that the kernel function is no longer given in analytic form and the testing data should be known beforehand so that it can used for computing the modified kernel matrix.

4.4.2 Experimental Setup

The learning problems used in these experiments are the same as the ones used in the context of distance-based learning (Section 3.5). More precisely, the learning problems of the first group (i.e. diterpenes, two versions of musk and duke) are characterized by the fact that the learning instances are represented as sets of vectors requiring no further recursion. The learning problems of the second group (mutagenicity and four versions of carcinogenicity) are graph classification problems where the learning instances can be represented in many different ways. The representations we used here are based on graph decompositions into trees and tree-like structures (see Section 3.5.3). The last learning problem is that of protein fingerprint classification. For this problem we used two representations where motifs are represented as sets and lists. We experimented only with the weighted version of the problem. The details on the datasets and on the different representations used are given in Appendix A and in Section 3.5.3.

The learning task is always classification. We use both SVM (Section 4.4.1) and kNN (Section 3.5.1) classification rules to decide the classification of a given instance. We estimate accuracy using ten-fold stratified cross-validation and control for the statistical significance of observed differences for all pairs of

distance measures using McNemar's test (McNemar, 1947), significance level of 0.05. Similarly to the experimental setup described in Section 3.5.2 we establish a ranking schema of the different kernels, based on their relative performance as determined by the results of the significance tests, as follows (Kalousis and Theoharis, 1999): in a given dataset if a kernel k_1 is significantly better than k_2 then k_1 is credited with one point and k_2 with zero points; if there is no significant difference then both are credited with half point.

4.4.3 Experiments with Cross Product Kernel

In the experiments with the cross product (CP) kernel (using both SVM and kNN) we want to perform several comparisons of different composite kernels and examine the influence of their parameter settings to the final results. First, we want to explore the effect of different vector kernels (i.e. k_{lin}, k_{poly} and k_{RBF}). Second, we want to see how the different kernel set normalizations, i.e. averaging ($f_{norm}(\cdot) = |\cdot|$) and feature space normalization ($f_{norm}(\cdot) = \sqrt{k(\cdot,\cdot)}$), influence the performance of the SVM and kNN classifiers. This comparison will be performed only in the first group of datasets (diterpenes, both versions of musk and duke datasets) and we report results where the kernels on tuples is fixed to the direct sum kernel. Third, we want to see how the performance of the direct sum kernel (k_Σ) compares with that of tensor product kernel (k_Π). This comparison will be performed on the graph and protein fingerprints datasets. In graph learning problems the building blocks of the kernel on trees (or tree-like structures) from Equation 4.31 are the CP kernel, different kernels on vectors and two kernels over tuples, i.e. k_Σ and k_Π. The normalization scheme in the CP kernel is fixed to averaging and was used: (i) to compute the set kernel between two sets of decompositions and (ii) inside the tree kernels. Finally, we are going to examine how the SVM algorithm compares with the kNN algorithm which uses distances induced from the corresponding kernels (Equation 4.3). By doing this we establish the influence of the biases introduced by SVM and kNN on the predictive performance.

In SVMs the regularization parameter C was optimized in an inner 10-fold cross validation loop over the set $C = \{0.1, 1, 10, 50\}$ whereas in kNN the selection of the k parameter was performed over set $k = \{1, 3, 9\}$. In protein fingerprints for kNN applied on an independent test set the k parameter was fixed to 9, as this value was most frequently selected by the corresponding cross-validation. The vector kernels in all the experiments are the linear (Equation 4.11), polynomial (Equation 4.13) and Gaussian RBF (Equation 4.9) kernels. In polynomial kernel we report results for $p = \{2, 3\}, l = 1$ whereas in Gaussian RBF kernel we experimented with $\gamma = \{0.1, 1, 10\}$.

Classification Performance

The results for SVM are presented in Tables 4.1 and 4.2 while the results where kNN is used are given in Tables 4.3 and 4.4. In the above tables the top ranked kernels (according to the ranking schema) are emphasized. In Tables 4.3 and 4.4

Kernel	diterpenes	musk (ver. 1)	musk (ver. 2)	duke
	$f_{norm}(\cdot) = \lvert\cdot\rvert$			
k_{lin}	75.12 (2.5)	*83.70 (6.5)*	84.31 (6.0)	*58.54 (5.5)*
$k_{poly,p=2,l=1}$	77.98 (4.0)	*89.13 (6.5)*	82.35 (5.5)	*58.54 (5.5)*
$k_{poly,p=3,l=1}$	79.77 (5.5)	*84.78 (6.5)*	87.25 (6.0)	*58.54 (5.5)*
$k_{RBF,\gamma=0.1}$	71.66 (0.5)	*86.96 (6.5)*	*94.12 (9.0)*	*58.54 (5.5)*
$k_{RBF,\gamma=1}$	81.30 (7.5)	*89.13 (6.5)*	90.20 (6.5)	*58.54 (5.5)*
$k_{RBF,\gamma=10}$	87.56 (10.0)	51.09 (0.5)	61.76 (0.5)	*58.54 (5.5)*
	$f_{norm}(\cdot) = \sqrt{k(\cdot,\cdot)}$			
k_{lin}	75.78 (2.5)	*86.96 (6.5)*	82.35 (5.5)	*58.54 (5.5)*
$k_{poly,p=2,l=1}$	79.71 (5.5)	*88.04 (6.5)*	84.31 (6.0)	*58.54 (5.5)*
$k_{poly,p=3,l=1}$	81.44 (7.5)	*84.78 (6.5)*	90.20 (7.5)	*58.54 (5.5)*
$k_{RBF,\gamma=0.1}$	71.59 (0.5)	*85.87 (6.5)*	91.18 (6.5)	*58.54 (5.5)*
$k_{RBF,\gamma=1}$	83.43 (9.0)	*85.87 (6.5)*	87.25 (6.5)	*58.54 (5.5)*
$k_{RBF,\gamma=10}$	*90.22 (11.0)*	51.09 (0.5)	61.76 (0.5)	*58.54 (5.5)*

Table 4.1: Accuracy and rank results (SVM) for k_{CP} on the datasets where instances are sets of vectors.

the sign in the second parenthesis indicates whether the performance of kNN with the corresponding kernel is significantly better (the + sign) or worse (the - sign) than that of SVM or there is no significant difference (the = sign). The detailed significance results are given in Appendix B.2.

Comparison of Different Vector Kernels In order to compare the different vector kernels we average over the datasets the ranks of k_{poly} (including k_{lin}) and k_{RBF}, ignoring their parameter settings. For SVM there is a slight advantage of the polynomial over the Gaussian RBF kernels; the average rank of polynomial vector kernels is 5.68 (for Gaussian RBF kernels 5.32). For kNN the two kernels perform similarly and the average ranks for the polynomial and Gaussian RBF kernels are 5.53 and 5.47, respectively. We also observed that the polynomial kernel is not very sensitive to the parameter settings (i.e. the p parameter) and thus no extensive search is required over the parameter space. The Gaussian RBF kernel is less stable and the wrong selection of the γ parameter might degrade the performance dramatically (see e.g. the results of $k_{RBF,\gamma=10}$ for version 1 of musk). This observation agrees with the discussion at the end of Section 4.2.2 where the stability of the Gaussian RBF kernel with respect to the γ parameter is considered. For SVM the average ranks for the linear and two polynomial kernels were 5.28, 5.78 and 5.98, respectively. On the other hand, the average ranks for the Gaussian RBF are 4.72, 6.08 and 5.15, respectively. For kNN the corresponding values for the linear and polynomial kernels are 5.18, 5.73 and 5.68; for Gaussian RBF the average ranks are 5.47, 5.7 and 5.23. The stability among different parameters of the polynomial kernel might be a result of the fact that the normalization of the CP kernel plays an important role in the construction of our structured kernel. This normalization can factor

Kernel	muta (ver. 1)	muta (ver. 2)	FM (ver. 1)	FM (ver. 2)
		k_Σ		
k_{lin}	80.32 (4.0)	76.60 (5.5)	*62.18 (5.5)*	61.32 (5.5)
$k_{poly,p=2,l=1}$	80.32 (4.0)	76.60 (5.5)	*62.18 (5.5)*	61.32 (5.5)
$k_{poly,p=3,l=1}$	79.79 (3.5)	76.60 (5.5)	*62.46 (5.5)*	61.32 (5.5)
$k_{RBF,\gamma=0.1}$	75.53 (1.5)	75.00 (4.5)	*61.32 (5.5)*	61.32 (5.5)
$k_{RBF,\gamma=1}$	78.72 (3.0)	77.66 (5.5)	*62.46 (5.5)*	60.74 (5.0)
$k_{RBF,\gamma=10}$	81.38 (4.5)	76.60 (5.0)	*62.46 (5.5)*	61.32 (5.5)
		k_Π		
k_{lin}	*87.23 (9.0)*	80.32 (5.5)	*61.60 (5.5)*	62.18 (5.5)
$k_{poly,p=2,l=1}$	*87.23 (9.0)*	80.32 (5.5)	*62.18 (5.5)*	62.18 (5.5)
$k_{poly,p=3,l=1}$	*87.23 (9.0)*	79.79 (5.5)	*61.89 (5.5)*	62.18 (5.5)
$k_{RBF,\gamma=0.1}$	77.66 (3.0)	77.13 (5.5)	*61.32 (5.5)*	61.03 (5.5)
$k_{RBF,\gamma=1}$	86.17 (8.5)	81.91 (6.0)	*62.18 (5.5)*	62.18 (5.5)
$k_{RBF,\gamma=10}$	84.57 (7.0)	*82.45 (6.5)*	*61.60 (5.5)*	*62.46 (6.0)*
	FR (ver. 1)	FR (ver. 2)	MM (ver. 1)	MM (ver. 2)
		k_Σ		
k_{lin}	*68.38 (5.5)*	*66.95 (6.5)*	*64.58 (5.5)*	*63.39 (5.5)*
$k_{poly,p=2,l=1}$	*68.09 (5.5)*	*67.24 (6.5)*	*64.58 (5.5)*	*63.69 (5.5)*
$k_{poly,p=3,l=1}$	*68.38 (5.5)*	*67.52 (6.5)*	*64.58 (5.5)*	*63.10 (5.5)*
$k_{RBF,\gamma=0.1}$	*68.09 (5.5)*	65.24 (0.5)	*64.58 (5.5)*	*62.80 (5.5)*
$k_{RBF,\gamma=1}$	*68.09 (5.5)*	*67.81 (6.5)*	*64.58 (5.5)*	*63.69 (5.5)*
$k_{RBF,\gamma=10}$	*68.09 (5.5)*	*68.09 (6.5)*	*64.58 (5.5)*	*63.69 (5.5)*
		k_Π		
k_{lin}	*67.52 (5.5)*	*68.09 (6.5)*	*63.10 (5.5)*	*63.39 (5.5)*
$k_{poly,p=2,l=1}$	*67.81 (5.5)*	*67.81 (6.5)*	*63.99 (5.5)*	*63.10 (5.5)*
$k_{poly,p=3,l=1}$	*67.81 (5.5)*	*68.09 (6.5)*	*63.99 (5.5)*	*63.99 (5.5)*
$k_{RBF,\gamma=0.1}$	*68.09 (5.5)*	65.24 (0.5)	*63.10 (5.5)*	*63.10 (5.5)*
$k_{RBF,\gamma=1}$	*67.52 (5.5)*	*68.09 (6.5)*	*63.10 (5.5)*	*63.99 (5.5)*
$k_{RBF,\gamma=10}$	*67.81 (5.5)*	*68.09 (6.5)*	*63.39 (5.5)*	*63.39 (5.5)*
	MR (ver. 1)	MR (ver. 2)	Protein Fingerprints	
			10-fold CV	test set
		k_Σ		
k_{lin}	*58.72 (5.5)*	57.56 (6.0)	83.12 (1.5)	77.75
$k_{poly,p=2,l=1}$	*58.72 (5.5)*	58.43 (6.0)	85.61 (7.0)	83.94
$k_{poly,p=3,l=1}$	*58.72 (5.5)*	*58.72 (6.5)*	86.08 (7.0)	83.66
$k_{RBF,\gamma=0.1}$	*58.43 (5.5)*	55.52 (1.5)	85.68 (7.0)	81.97
$k_{RBF,\gamma=1}$	*58.72 (5.5)*	*58.72 (6.5)*	*86.08 (7.5)*	83.66
$k_{RBF,\gamma=10}$	*58.72 (5.5)*	*58.72 (6.5)*	84.87 (5.5)	76.90
		k_Π		
k_{lin}	*59.59 (5.5)*	58.43 (6.0)	83.25 (1.5)	77.74
$k_{poly,p=2,l=1}$	*60.17 (5.5)*	57.85 (5.5)	85.94 (7.0)	83.94
$k_{poly,p=3,l=1}$	*60.17 (5.5)*	57.85 (5.5)	*86.21 (7.5)*	83.66
$k_{RBF,\gamma=0.1}$	*57.85 (5.5)*	56.98 (4.0)	85.74 (7.0)	81.97
$k_{RBF,\gamma=1}$	*61.34 (5.5)*	58.14 (6.0)	*86.35 (7.5)*	83.66
$k_{RBF,\gamma=10}$	*60.76 (5.5)*	58.14 (6.0)	80.63 (0.0)	76.90

Table 4.2: Accuracy and rank results (SVM) for k_{CP} on the graph and protein fingerprints datasets.

Kernel	diterpenes	musk (ver. 1)	musk (ver. 2)	duke
	$f_{norm}(\cdot) = \lvert\cdot\rvert$			
k_{lin}	89.22 (2.0)(+)	*85.87 (7.0)*(=)	66.67 (5.5)(-)	*58.54 (5.5)*(=)
$k_{poly,p=2}$	91.08 (5.5)(+)	*85.87 (7.0)*(=)	68.63 (5.5)(-)	*46.34 (5.5)*(=)
$k_{poly,p=3}$	91.62 (6.0)(+)	*88.04 (7.0)*(=)	67.65 (5.0)(-)	*41.46 (5.5)*(=)
$k_{RBF,\gamma=0.1}$	90.02 (2.5)(+)	*83.70 (7.0)*(=)	*74.51 (6.5)*(-)	*46.34 (5.5)*(=)
$k_{RBF,\gamma=1}$	92.28 (8.0)(+)	57.61 (1.5)(-)	*78.43 (6.5)*(-)	*39.02 (5.5)*(=)
$k_{RBF,\gamma=10}$	*94.81 (10.5)*(+)	48.91 (0.5)(=)	61.76 (5.0)(=)	*41.46 (5.5)*(=)
	$f_{norm}(\cdot) = \sqrt{k(\cdot,\cdot)}$			
k_{lin}	88.16 (0.0)(+)	*83.70 (7.0)*(=)	67.65 (5.5)(-)	*58.54 (5.5)*(=)
$k_{poly,p=2}$	90.82 (4.0)(+)	*85.87 (7.0)*(=)	63.73 (4.0)(-)	*41.46 (5.5)*(=)
$k_{poly,p=3}$	91.68 (7.0)(+)	*88.04 (7.0)*(=)	68.63 (5.5)(-)	*41.46 (5.5)*(=)
$k_{RBF,\gamma=0.1}$	90.09 (2.5)(+)	*85.87 (7.0)*(=)	71.57 (5.5)(-)	*51.22 (5.5)*(=)
$k_{RBF,\gamma=1}$	92.28 (7.5)(+)	*88.04 (7.0)*(=)	73.53 (6.0)(-)	*46.34 (5.5)*(=)
$k_{RBF,\gamma=10}$	*94.88 (10.5)*(+)	58.70 (1.0)(=)	72.55 (5.5)(=)	*48.78 (5.5)*(=)

Table 4.3: Accuracy and rank results (kNN) for k_{CP} on the datasets where instances are sets of vectors.

out the effect of possible outliers and different cardinalities of sets. In the case of k_{RBF} with large values of γ the corresponding Gram matrix is almost an identity matrix. The kernel matrix of k_{CP} has similar behavior, independently of the normalization. In addition, for the graph datasets, this stability could be an indication that the structural properties of the relational instances are more important for the classification than the properties of constituent atoms and bonds. The last finding is that the performance of the linear kernel, k_{lin}, is usually inferior to the one of k_{poly} and k_{RBF}, provided that for the latter kernels careful parameter selection is performed. On the other hand the advantage of k_{lin} is that it does not require any parameter selection.

Normalization of the CP kernel The different normalization methods for the CP kernel do not appear to have a big influence on the final results. For SVM in diterpenes dataset averaging led to an average rank of 5 over the different elementary kernels, and feature space normalization to an average rank of 6. In musk 1 (musk 2) the corresponding figures were 5.5 and 5.5 (5.58 and 5.42). In duke the two normalization schemes had the same average rank of 5.5. For kNN similar trends hold: the average ranks for averaging (features space) normalization in diterpenes, musk (ver. 1), musk (ver. 2) and duke are 5.75 (5.25), 5 (6), 5.67 (5.33) and 5.5 (5.5), respectively. One explanation for this might be that the two denominators in the explicit feature space representations for the two kernels (Equations 4.17 and 4.18) are correlated, which makes sense since sets of higher cardinality will probably have a higher $\|\sum_{a\in A}\phi_k(a)\|$, at least for the datasets we examined. However, this depends on the problem at hand, there could other datasets where this correlation does not hold.

Kernel	muta (ver. 1)	muta (ver. 2)	FM (ver. 1)	FM (ver. 2)
			k_Σ	
k_{lin}	84.04 (5.5)(=)	*85.64 (5.5)*(+)	*62.75 (6.5)*(=)	60.17 (5.5)(=)
$k_{poly,p=2,l=1}$	82.45 (5.0)(=)	*87.77 (5.5)*(+)	*62.46 (6.5)*(=)	58.45 (5.5)(=)
$k_{poly,p=3,l=1}$	83.51 (5.5)(=)	*86.70 (5.5)*(+)	62.18 (6.0)(=)	*59.31 (6.0)*(=)
$k_{RBF,\gamma=0.1}$	82.98 (5.5)(+)	*87.77 (5.5)*(+)	61.89 (6.0)(=)	57.59 (5.5)(=)
$k_{RBF,\gamma=1}$	84.57 (5.5)(=)	*87.23 (5.5)*(+)	61.89 (6.0)(=)	57.31 (5.5)(=)
$k_{RBF,\gamma=10}$	84.57 (5.5)(=)	*86.17 (5.5)*(+)	59.89 (5.5)(=)	56.16 (5.0)(=)
			k_Π	
k_{lin}	85.11 (5.5)(=)	*85.11 (5.5)*(=)	57.02 (5.5)(=)	59.31 (5.5)(=)
$k_{poly,p=2,l=1}$	84.57 (5.0)(=)	*86.17 (5.5)*(=)	57.31 (5.5)(=)	59.89 (5.5)(=)
$k_{poly,p=3,l=1}$	86.70 (5.5)(=)	*87.23 (5.5)*(+)	57.02 (4.5)(=)	59.31 (5.5)(=)
$k_{RBF,\gamma=0.1}$	85.64 (5.5)(+)	*86.17 (5.5)*(+)	59.03 (5.5)(=)	57.31 (5.5)(=)
$k_{RBF,\gamma=1}$	*87.77 (6.5)*(=)	*88.83 (5.5)*(+)	55.01 (3.0)(-)	57.59 (5.5)(=)
$k_{RBF,\gamma=10}$	86.70 (5.5)(=)	*87.77 (5.5)*(+)	57.31 (5.5)(=)	56.45 (5.5)(=)
	FR (ver. 1)	FR (ver. 2)	MM (ver. 1)	MM (ver. 2)
			k_Σ	
k_{lin}	*67.24 (5.5)*(=)	62.39 (5.5)(=)	*63.39 (6.0)*(=)	62.50 (5.5)(=)
$k_{poly,p=2,l=1}$	*66.95 (5.5)*(=)	62.96 (5.5)(=)	*63.69 (6.0)*(=)	*66.07 (9.0)*(=)
$k_{poly,p=3,l=1}$	*67.52 (5.5)*(=)	62.96 (5.5)(-)	61.90 (5.5)(=)	63.69 (6.0)(=)
$k_{RBF,\gamma=0.1}$	*67.24 (5.5)*(=)	62.68 (5.5)(=)	62.80 (5.5)(=)	64.58 (7.5)(=)
$k_{RBF,\gamma=1}$	*67.52 (5.5)*(=)	63.82 (5.5)(=)	61.31 (5.5)(=)	59.52 (4.0)(=)
$k_{RBF,\gamma=10}$	*68.09 (5.5)*(=)	62.39 (5.5)(-)	61.90 (5.5)(=)	60.71 (4.5)(=)
			k_Π	
k_{lin}	*66.38 (5.5)*(=)	63.82 (5.5)(=)	58.04 (4.5)(=)	61.31 (5.5)(=)
$k_{poly,p=2,l=1}$	*67.52 (5.5)*(=)	65.24 (5.5)(=)	58.33 (5.5)(=)	60.12 (5.0)(=)
$k_{poly,p=3,l=1}$	*67.81 (5.5)*(=)	*66.38 (6.0)*(=)	59.52 (5.5)(=)	59.23 (4.5)(=)
$k_{RBF,\gamma=0.1}$	*65.81 (5.5)*(=)	63.25 (5.5)(=)	60.12 (5.5)(=)	60.71 (5.0)(=)
$k_{RBF,\gamma=1}$	*66.10 (5.5)*(=)	64.67 (5.5)(=)	58.93 (5.5)(=)	59.52 (4.5)(=)
$k_{RBF,\gamma=10}$	*65.81 (5.5)*(=)	62.39 (5.0)(-)	63.10 (5.5)(=)	59.52 (5.0)(=)
	MR (ver. 1)	MR (ver. 2)	Protein Fingerprints	
			10-fold CV	test set
			k_Σ	
k_{lin}	52.33 (5.0)(=)	53.49 (6.0)(=)	82.99 (2.5) (=)	80.84
$k_{poly,p=2,l=1}$	53.20 (5.5)(=)	50.58 (3.5)(-)	84.73 (6.5) (=)	83.94
$k_{poly,p=3,l=1}$	52.62 (5.0)(=)	51.16 (4.0)(-)	84.80 (6.5) (=)	83.66
$k_{RBF,\gamma=0.1}$	52.62 (5.0)(=)	49.42 (2.0)(=)	84.33 (5.5) (=)	81.97
$k_{RBF,\gamma=1}$	53.78 (5.5)(=)	50.29 (3.5)(-)	84.87 (6.5) (=)	83.66
$k_{RBF,\gamma=10}$	52.03 (5.0)(=)	50.29 (3.5)(-)	83.52 (4.0) (=)	76.90
			k_Π	
k_{lin}	57.27 (5.5)(=)	*56.69 (8.0)*(=)	82.04 (2.0) (=)	77.74
$k_{poly,p=2,l=1}$	57.56 (5.5)(=)	*56.98 (8.0)*(=)	85.34 (7.5) (=)	83.94
$k_{poly,p=3,l=1}$	55.23 (5.5)(=)	55.52 (6.0)(=)	85.14 (7.0) (=)	83.66
$k_{RBF,\gamma=0.1}$	57.27 (5.5)(=)	54.94 (6.0)(=)	85.74 (8.0) (=)	81.97
$k_{RBF,\gamma=1}$	56.69 (5.5)(=)	*57.56 (8.0)*(=)	*86.15 (10.0)* (=)	83.66
$k_{RBF,\gamma=10}$	*59.30 (7.5)*(=)	56.40 (7.5)(=)	56.22 (0.0) (-)	76.90

Table 4.4: Accuracy and rank results (kNN) for k_{CP} on the graph and protein fingerprints datasets.

Comparison of k_Σ vs. k_Π The next dimension of comparison is the relative performance of k_Σ and k_Π. There is a slight advantage of k_Π over k_Σ: for kNN the averaged ranks (over all the graph datasets) of k_Σ and k_Π are 5.4 and 5.6, respectively. This is a rather surprising fact since as we have seen before, the instance-based learning in the space induced by the tensor product kernel should be harder than in the space induced by the direct sum kernel. However, k_Π is more expressive than k_Σ because it accounts for feature interactions. The trade-off between hardness of learning in a space of higher dimensionality and the higher expressiveness might explain the better performance of k_Π than that of k_Σ. For SVM the advantage of k_Π over k_Σ is even bigger: the averaged ranks of k_Σ is 5.32 while for k_Π the corresponding value is 5.78. The better performance of k_Π in SVM could be a result of the fact that in general large margin classifiers regularize the solutions, and hence they handle redundant or irrelevant features in a more efficient way (Schölkopf and Smola, 2001).

Comparison of SVM vs. kNN We also compared the performances of the SVM and kNN for different structured kernels (the results are reported in the second parenthesis in Tables 4.3 and 4.4). In general there is an advantage of using SVM over kNN; the highest is in the musk (version 2) where SVM is significantly better than kNN in 10 out of 12 kernels. For other datasets (with the exception of the diterpenes and muta datasets) SVM wins more often. The poor performance of SVM in diterpenes datasets might be explained by the fact this is a 23-class problem, and hence most of the binary classifiers which are combined in multi-class predictions are unbalanced (see Section 4.4.1).

Summary of the Results To summarize, in our experiments we observed several findings: (i) our complex kernels are in general stable to the parameter settings of the vectorial kernels (polynomial kernels are more stable than the Gaussian RBF kernels) – this is probably due to the normalization of k_{CP} or the importance of the structural features, (ii) the performance of the linear kernel is usually inferior to the one of polynomial or Gaussian RBF kernels – the nonlinearity induced by the latter kernels is important, (iii) different normalizations of k_{CP} as well as different choices of kernels on tuples do not have a big impact on the classification accuracy and (iv) in general there is an advantage of using SVM over kNN (with the exception of the learning problems with large number of classes).

Comparison with other systems Finally, we note that for many datasets the results obtained using our best kernel (for SVM) compare favorably with the results obtained using other relational kernel-based systems and the best complex distances from Chapter 3. These results are reported in Table 4.5 under the columns SVM_{best}, "Kernel-based" and kNN_{Best}, respectively. The references of the specific relational kernel-based systems are given in Section 3.5.6. The entries in the "SVM_{best} from Section 4.4.4" column will be discussed in the next

SVM_{best}	SVM_{best} from Section 4.4.4	Kernel-based	kNN_{Best}
		diterpenes	
90.22	94.54	KeS 94.70	97.41
		musk (ver. 1)	
89.13	94.74	KeS 81.00 MIK 91.60 mi-SVM 87.40 MI-SVM 77.90	84.78
		musk (ver. 2)	
94.12	92.24	KeS 85.50 MIK 88.00 mi-SVM 83.60 MI-SVM 84.30	77.45
		duke	
58.54	70.73		78.05
		mutagenesis	
87.23	85.64	MIK 93.00 K1 85.1 (loo) K2 91.0 (loo) K3 91.5 (loo) RK 85.4	87.23
		FM	
62.46	67.05	K1 63.4 (loo) K3 64.5 (loo)	63.32
		FR	
68.38	67.81	K1 66.1 (loo) K3 66.9 (loo)	66.67
		MM	
64.58	66.67	K1 64.3 (loo) K3 66.4 (loo)	65.18
		MR	
61.34	64.53	K1 58.4 (loo) K3 65.7 (loo)	61.63

Table 4.5: Accuracy results of the best kernel from this section (SVM_{best}) together with the best distance (kNN_{Best}) and other related relational kernels ("Kernel-based"). Additionally, we provide the best results obtained using the kernels from Section 4.4.4. The references of the specific systems ("Kernel-based" column) are given in Section 3.5.6.

section. We mention that the results in columns "Kernel-based" and kNN_{Best} are the same as reported in Table 3.7.

The best result in version 2 of musk (94.12 %) obtained for $k_{RBF,\gamma=0.1}$ is

significantly better than the one of the best distance measure and is the best result reported in the literature so far. In FR the best result (68.38 %) is better than both best distance and other related kernels, however, the difference in performance with the former is not statistically meaningful. In version 1 of musk the result for our kernel is better than the best performance obtained using distances (this difference is also not statistically significant), however, it is worse than the performance of the MIK kernel of Gärtner et al. (2002). The best kernel in mutagenesis has the same performance as the best distance. Finally, the results for diterpenes, duke, FM, MR and MM are worse than the best accuracies from the previous chapter, however the differences in performances are statistically meaningful only in diterpenes.

4.4.4 Experiments with Set Kernels Based on Mappings

In these experiments we want to perform several comparisons of the SVM and kNN algorithms with different set kernels based on specific pairs, and distances over sets. First, for the various distance measures we want to explore the relative performance of: (i) the kernels based on mappings (Section 4.2.4), (ii) the DS kernels from Equation 4.26 and (iii) the linear kernel defined in the corresponding proximity spaces. These kernels will be used with the SVM method, resulting in the SVM_{SP} , SVM_{DS} and SVM_{P} algorithms, respectively. Second, we want to see how the performance of SVM with the above kernels compares with SVM with the following three kernels based on averaging: (i) Bhattacharrya kernel (Bhatta) (Kondor and Jebara, 2003) with the linear kernel as the elementary kernel (ii) the cross product kernel k_{CP} with the linear kernel and (iii) the linear kernel in the proximity space induced by the d_{AL} distance measure; kernels based on averaging are a standard way of tackling classification problems where instances are represented as sets. Third, we are going to examine how the SVM_{P} algorithm compares with the kNN algorithm where these distances are used directly. By doing this we establish whether SVM_{P} indeed provides an improvement over the simple kNN. Finally, we will try to gain more insight into the proximity space by examining the relative performance of the SVM_{P} and the kNN algorithm operating in the same feature space (denoted as kNN_{P}). The reason we use the linear kernel in the experimental setup is to make a fair comparison between the algorithms and to avoid the situation where an implicit mapping given by a nonlinear kernel will influence the results.

The results are not reported for the protein fingerprints dataset. For graph datasets the results are only reported where the first representation of the learning instances is used, i.e. graphs are represented as sets of trees (see Section 3.5.3). The representation of learning objects for other datasets is the same as in the previous section. We also note that we do not report results using the mapping defined by d_{RIBL} since this is closely related with the mapping of d_{SMD} (see the remark after Definition 4.15). Finally, the kernel corresponding to d_T and used within SVM_{SP} is in fact the intersection kernel of Definition 4.16. For SVM_{P}, SVM_{SP}, and for kernels based on averaging the regularization parameter C was optimized in an inner 10-fold cross validation loop over the set

$C = \{0.1, 1, 10, 50\}$. For SVM_{DS} the same procedure was used to optimize the width γ and the C parameter over the grid of $\gamma = \{0.1, 1, 5, 10, 20, 30, 40, 50, 60\}$ and $C = \{0.1, 1, 10, 50\}$. In all the kNN algorithms the number of nearest neighbors was optimized over the set $k = \{1, 3, 9\}$.

Classification Performance

The classification and rank results (with the significance test results in parenthesis) are presented in Tables 4.6, 4.7 and 4.8. The first parenthesis contains rank results comparing SVM_{P}, SVM_{DS} and SVM_{SP} (for a given distances). The sign in the second parenthesis corresponds to comparison of SVM_{P} vs. Bhattacharrya kernel from (Kondor and Jebara, 2003) and the third to SVM_{P} vs. SVM with cross product (CP) kernel and the last one to SVM_{P} vs. SVM_{P} where d_{AL} is used. In all the cases + stands for a significant win of the first algorithm in the pair, - for a significant loss and = for no significant difference.

Comparison of SVM_{P}, SVM_{DS} and SVM_{SP} In order to compare SVM_{P} with SVM_{DS} and SVM_{SP} we fix a dataset and for each distance measure we compute ranks of different kernels. The comparison results of the three kernels provide strong evidence that in terms of predictive accuracy there is an advantage of the linear kernels in the proximity spaces over the set distance substitution kernels, and the kernels which are directly based on specific mappings. Indeed, SVM_{P} was significantly worse than SVM_{DS} only once and it was significantly better in 30 cases. In the remaining 50 cases there was no significant difference. The advantage of SVM_{P} in comparison with SVM_{SP} is even stronger: SVM_{P} was significantly better in 38 cases, there was no significant difference in 41 cases and SVM_{P} was significantly worse only in 1 case. The average ranks for SVM_{P} (averaged over different mappings and different datasets) was 1.41. The average ranks for SVM_{DS} and SVM_{SP} were 0.97 and 0.64, respectively. The better performance of SVM_{P} could be explained by the fact that the set distance substitution kernels and kernels used within SVM_{SP} for our mappings are not PSD, which means that it is harder for SVM to find an optimal solution.

To further investigate the above issue we examined the spectra of the kernel matrices corresponding to the different set kernels presented in Section 4.2.4 which are directly based on the different mapping between the sets. The results for the considered datasets are visualized in Figure 4.2. In each of the graphs the x-axis corresponds to different set kernels. For each set kernel we provide the r ratio (solid lines) from Equation 4.35 which characterizes the "non-positive definiteness" of the corresponding kernel matrices. Additionally, we present the estimated accuracies of SVM_{SP} for each of the kernel (the bars). The analysis of the spectra reveals that for all the datasets (except for diterpenes) for the mappings corresponding to the k_{SL}, k_{CL}, k_{SMD}, k_H, $k_\cap$, k_S and k_L set kernels, the proportion of sums of negative to the sums of positive eigenvalues as given by the r ratio is relatively small. This is in contrast with the remaining set kernels, i.e. k_{FS} and k_M, for which the r ratio is larger, and in many datasets it takes values which are close to (or even larger than) one. This could indicate

	diterpenes		
	SVM_P	SVM_{DS}	SVM_{SP}
d_{SL}	82.50 (2.0)(=)(+)(-)	29.81 (0.5)	29.81 (0.5)
d_{CL}	52.03 (2.0)(-)(-)(-)	29.81 (0.0)	37.13 (1.0)
d_{SMD}	94.54 (2.0)(+)(+)(+)	90.22 (1.0)	49.77 (0.0)
d_H	73.85 (1.5)(-)(=)(-)	73.85 (1.5)	29.81 (0.0)
d_T	94.21 (2.0)(+)(+)(+)	90.22 (1.0)	68.00 (0.0)
d_S	93.68 (2.0)(+)(+)(+)	82.30 (1.0)	29.81 (0.0)
d_{FS}	93.68 (2.0)(+)(+)(+)	82.24 (1.0)	29.81 (0.0)
d_L	93.48 (2.0)(+)(+)(+)	60.21 (1.0)	29.81 (0.0)
d_M	94.48 (2.0)(+)(+)(+)	89.02 (1.0)	29.81 (0.0)
Bhatta	83.03		
k_{CP}	75.12		
d_{AL}	91.62		
	musk (ver. 1)		
	SVM_P	SVM_{DS}	SVM_{SP}
d_{SL}	79.35 (1.0)(=)(=)(=)	81.52 (1.5)	70.65 (0.5)
d_{CL}	79.35 (1.5)(=)(=)(=)	66.30 (0.0)	78.26 (1.5)
d_{SMD}	94.74 (2.0)(+)(+)(+)	85.87 (1.0)	67.39 (0.0)
d_H	83.70 (1.5)(=)(=)(=)	83.70 (1.5)	67.39 (0.0)
d_T	51.09 (1.0)(-)(-)(-)	51.09 (1.0)	53.26 (1.0)
d_S	79.35 (1.5)(=)(=)(=)	79.35 (1.5)	51.09 (0.0)
d_{FS}	90.22 (2.0)(+)(+)(+)	82.61 (1.0)	53.26 (0.0)
d_L	84.78 (2.0)(=)(=)(=)	68.48 (1.0)	51.09 (0.0)
d_M	84.78 (2.0)(=)(=)(=)	53.26 (0.5)	51.09 (0.5)
Bhatta	80.43		
k_{CP}	83.70		
d_{AL}	83.70		
	musk (ver. 2)		
	SVM_P	SVM_{DS}	SVM_{SP}
d_{SL}	82.35 (1.5)(+)(=)(=)	72.55 (1.5)	58.82 (1.5)
d_{CL}	80.39 (2.0)(=)(=)(=)	62.75 (0.5)	66.67 (0.5)
d_{SMD}	92.24 (2.0)(+)(+)(+)	78.43 (1.0)	56.86 (0.0)
d_H	91.27 (1.5)(+)(+)(+)	88.24 (1.5)	78.43 (0.0)
d_T	61.76 (0.5)(=)(-)(-)	61.76 (0.5)	72.55 (2.0)
d_S	83.33 (1.5)(+)(=)(=)	76.47 (1.5)	61.76 (0.0)
d_{FS}	75.49 (1.5)(=)(=)(=)	62.75 (1.5)	61.76 (0.0)
d_L	88.24 (2.0)(+)(=)(=)	75.49 (1.0)	61.76 (0.0)
d_M	78.43 (2.0)(=)(=)(=)	60.78 (0.5)	55.88 (0.5)
Bhatta	69.61		
k_{CP}	84.31		
d_{AL}	82.35		

Table 4.6: Accuracy, ranks and significance test results for set kernels in diterpenes and both versions of the musk dataset. The description of the notation is given in the text.

	duke		
	SVM_P	SVM_{DS}	SVM_{SP}
d_{SL}	68.29 (1.0)(=)(=)(=)	58.54 (1.0)	58.54 (1.0)
d_{CL}	68.29 (1.0)(=)(=)(=)	58.54 (1.0)	58.54 (1.0)
d_{SMD}	68.29 (1.0)(=)(=)(=)	58.54 (1.0)	58.54 (1.0)
d_H	70.73 (1.0)(=)(=)(=)	70.73 (1.0)	58.54 (1.0)
d_T	60.98 (1.0)(=)(=)(=)	65.85 (1.0)	58.54 (1.0)
d_S	65.85 (1.0)(=)(=)(=)	58.54 (1.0)	58.54 (1.0)
d_{FS}	58.54 (1.0)(=)(=)(=)	58.54 (1.0)	58.24 (1.0)
d_L	60.98 (1.0)(=)(=)(=)	58.54 (1.0)	58.54 (1.0)
d_M	68.29 (1.0)(=)(=)(=)	58.54 (1.0)	53.66 (1.0)
$Bhatta$	58.54		
k_{CP}	58.54		
d_{AL}	63.41		
	muta (ver. 1)		
	SVM_P	SVM_{DS}	SVM_{SP}
d_{SL}	79.79 (2.0)(+)(=)(=)	66.49 (0.5)	66.49 (0.5)
d_{CL}	67.02 (1.0)(=)(-)(=)	66.49 (1.0)	70.21 (1.0)
d_{SMD}	78.19 (2.0)(=)(=)(=)	63.83 (0.5)	64.36 (0.5)
d_H	73.94 (0.0)(=)(=)(=)	79.79 (1.5)	79.79 (1.5)
d_T	85.64 (1.5)(+)(=)(+)	85.64 (1.5)	66.49 (0.0)
d_S	81.91 (1.5)(+)(=)(+)	66.49 (0.0)	81.38 (1.5)
d_{FS}	79.79 (2.0)(=)(=)(+)	66.49 (1.0)	40.96 (0.0)
d_L	78.19 (2.0)(=)(=)(=)	66.49 (0.5)	66.49 (0.5)
d_M	83.51 (2.0)(+)(=)(+)	66.49 (0.5)	64.89 (0.5)
$Bhatta$	71.81		
k_{CP}	80.32		
d_{AL}	73.40		
	FM (ver. 1)		
	SVM_P	SVM_{DS}	SVM_{SP}
d_{SL}	58.74 (1.0)(=)(=)(=)	59.03 (1.0)	57.88 (1.0)
d_{CL}	54.73 (1.0)(-)(-)(=)	59.03 (1.0)	60.17 (1.0)
d_{SMD}	62.46 (1.5)(=)(=)(=)	56.73 (0.5)	58.74 (1.0)
d_H	61.03 (1.0)(=)(=)(=)	59.60 (1.0)	59.03 (1.0)
d_T	64.47 (1.0)(=)(=)(=)	67.05 (1.5)	59.03 (0.5)
d_S	62.46 (1.0)(=)(=)(=)	59.03 (1.0)	59.03 (1.0)
d_{FS}	57.02 (1.0)(-)(=)(=)	59.03 (1.5)	51.58 (0.5)
d_L	61.89 (1.5)(=)(=)(=)	57.88 (0.5)	59.03 (1.0)
d_M	62.46 (1.5)(=)(=)(=)	59.03 (1.5)	46.42 (0.0)
$Bhatta$	63.32		
k_{CP}	62.18		
d_{AL}	59.31		

Table 4.7: Accuracy, ranks and significance test results for set kernels in duke, muta and FM datasets. The description of the notation is given in the text.

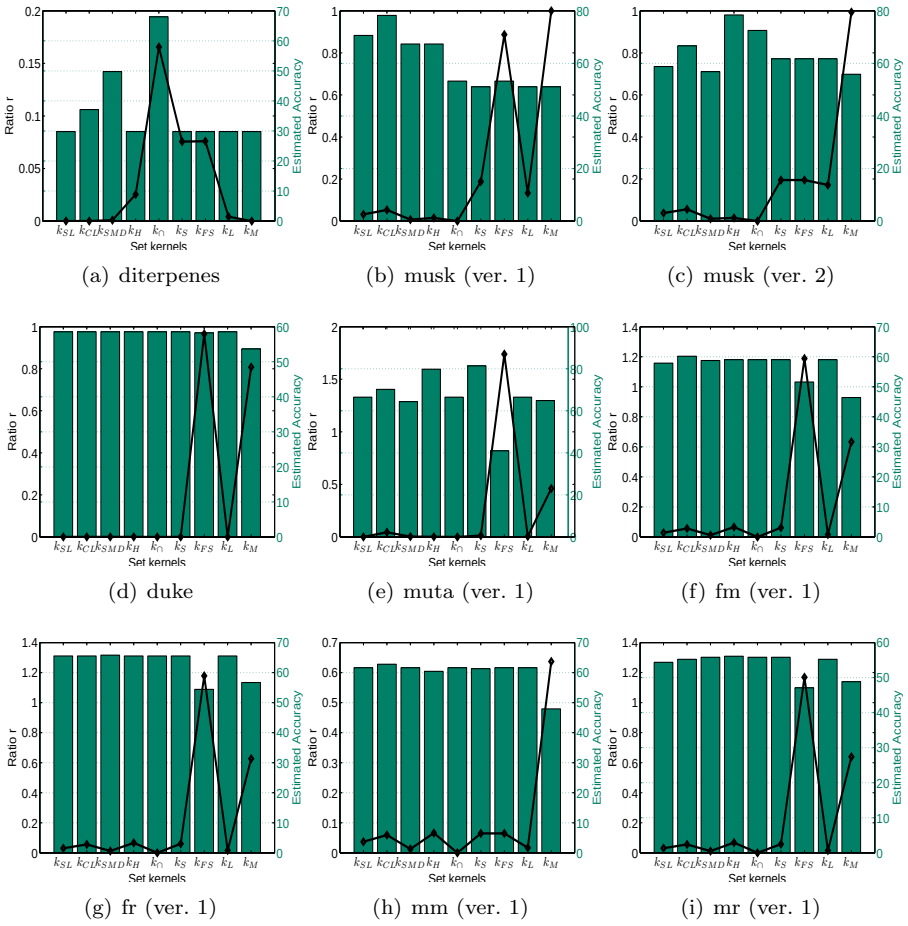
	FR (ver. 1)		
	SVM_P	SVM_{DS}	SVM_{SP}
d_{SL}	64.67 (1.0)(=)(-)(=)	65.53 (1.0)	65.53 (1.0)
d_{CL}	64.39 (1.0)(=)(-)(=)	65.53 (1.0)	65.53 (1.0)
d_{SMD}	67.81 (1.0)(=)(=)(=)	65.24 (1.0)	65.81 (1.0)
d_H	63.53 (1.0)(=)(-)(=)	64.96 (1.0)	65.53 (1.0)
d_T	65.53 (1.0)(=)(-)(=)	66.38 (1.0)	65.53 (1.0)
d_S	67.24 (1.0)(=)(=)(=)	64.67 (1.0)	65.53 (1.0)
d_{FS}	63.82 (1.5)(=)(-)(=)	65.53 (1.5)	54.42 (0.0)
d_L	66.95 (1.5)(=)(=)(=)	64.39 (0.5)	65.53 (1.0)
d_M	67.81 (2.0)(+)(=)(+)	65.53 (1.0)	56.70 (0.0)
Bhatta	63.82		
k_{CP}	68.38		
d_{AL}	65.53		
	MM (ver. 1)		
	SVM_P	SVM_{DS}	SVM_{SP}
d_{SL}	63.99 (1.0)(=)(=)(=)	59.82 (1.0)	61.61 (1.0)
d_{CL}	58.33 (1.0)(=)(-)(-)	61.61 (1.0)	62.80 (1.0)
d_{SMD}	65.48 (1.0)(=)(=)(=)	61.61 (1.0)	61.61 (1.0)
d_H	61.01 (1.0)(=)(=)(=)	61.01 (1.0)	60.42 (1.0)
d_T	66.37 (1.0)(=)(=)(=)	65.77 (1.0)	61.61 (1.0)
d_S	66.67 (2.0)(=)(=)(=)	61.61 (0.5)	61.31 (0.5)
d_{FS}	57.74 (1.0)(-)(-)(-)	61.61 (1.0)	61.61 (1.0)
d_L	66.07 (2.0)(=)(=)(=)	61.61 (0.5)	61.61 (0.5)
d_M	65.48 (1.5)(=)(=)(=)	63.69 (1.5)	47.92 (0.0)
Bhatta	64.29		
k_{CP}	64.58		
d_{AL}	63.99		
	MR (ver. 1)		
	SVM_P	SVM_{DS}	SVM_{SP}
d_{SL}	55.81 (1.0)(-)(=)(=)	55.81 (1.0)	54.36 (1.0)
d_{CL}	53.20 (1.0)(-)(-)(=)	55.81 (1.0)	55.23 (1.0)
d_{SMD}	57.56 (1.0)(=)(=)(=)	55.23 (1.0)	55.81 (1.0)
d_H	53.49 (1.0)(-)(=)(=)	54.94 (1.0)	56.10 (1.0)
d_T	64.53 (1.5)(=)(=)(+)	59.59 (1.0)	55.81 (0.5)
d_S	58.14 (1.0)(=)(=)(=)	56.10 (1.0)	55.81 (1.0)
d_{FS}	54.94 (1.5)(-)(=)(=)	55.81 (1.5)	47.09 (0.0)
d_L	60.47 (1.0)(=)(=)(=)	55.23 (1.0)	55.23 (1.0)
d_M	61.63 (1.5)(=)(=)(=)	55.52 (1.0)	48.84 (0.5)
Bhatta	62.79		
k_{CP}	58.72		
d_{AL}	55.23		

Table 4.8: Accuracy, ranks and significance test results for set kernels in FR, MM and MR datasets. The description of the notation is given in the text.

that for the former set kernels efficient learning with SVM is possible while for k_{FS} and k_M it is much harder for SVM to converge to a global optimum (see discussion in Section 4.4.1). Indeed, in these datasets we observed a correlation between the spectra of a given kernel matrix and the estimated accuracy of the corresponding SVM algorithm, i.e. a high ratio r usually is associated with low accuracy. In diterpenes the r ratio for all the set kernels was smaller that in other datasets, and it was the highest for $k_{\cap}$. In this dataset no clear correlation between the spectra and accuracies can be observed, e.g the highest predictive accuracy is obtained for $k_{\cap}$. As a result we can state that the bad performance of the SVM_{SP} algorithm, relative to SVM_{P}, might be a result of: (i) the lack of PSD-ness of the corresponding kernels or (ii) the fact that the kernels in the proximity space are simply better suited for the datasets we considered.

We also performed the above analysis for the different distance substitution kernels. The results for $\gamma = 1$ are presented in Figure 4.3; the results for $\gamma = 0.1$ and $\gamma = 10$ are presented in Appendix B.3. The results suggest that in comparison with set kernels used within SVM_{SP}, it should be easier to train SVM using the distance substitution kernels since in general the corresponding r ratio takes lower values. This observation agrees with empirical results since SVM_{DS} was significantly better than SVM_{SP} in 27 cases, it was worse in 4 cases and in the remaining cases (50) the differences were not statistically significant.

Comparison of SVM_{P} vs. SVM with Kernels Based on Averaging The next dimension of comparison is the relative performance between the SVM_{P} and SVM with kernels based on averaging (the latter, as already mentioned, is a standard approach to tackle set problems). We experimented with the following kernels based on averaging: the Bhattacharrya kernel (Bhatta) (Kondor and Jebara, 2003), the cross product kernel k_{CP} and the linear kernel in the proximity space induced by the d_{AL} distance measure. The main point of this comparison is to examine whether there are cases in which different ways of matching the elements of two sets can be more beneficial than the standard averaging which matches everything with everything. From the results it is clear that the relative performance of kernels based on specific pairs of elements and kernels based on averaging depends on the actual application. The strongest advantage of the former is in diterpenes, mutagenesis and musk. For carcinogenicity datasets the opposite trend holds. For duke both approaches have similar performance and no conclusions can be drawn. It should be noted that the state-of-the-art Bhattacharrya set kernel from (Kondor and Jebara, 2003) performs poorly for all the examined datasets. Overall the choice of the appropriate way of matching the elements of two sets depends on the application and ideally should be guided by domain knowledge, if such exists. Nevertheless, the relative performance of the different kernels provides valuable information about the type of problem we are facing. For example, by examining mutagenesis and carcinogenicity we see that although they correspond to the same type of classification problem, i.e. classification of graphs, in the latter averaging works better, hinting that the global structure of the molecules is important, whereas in the former averaging

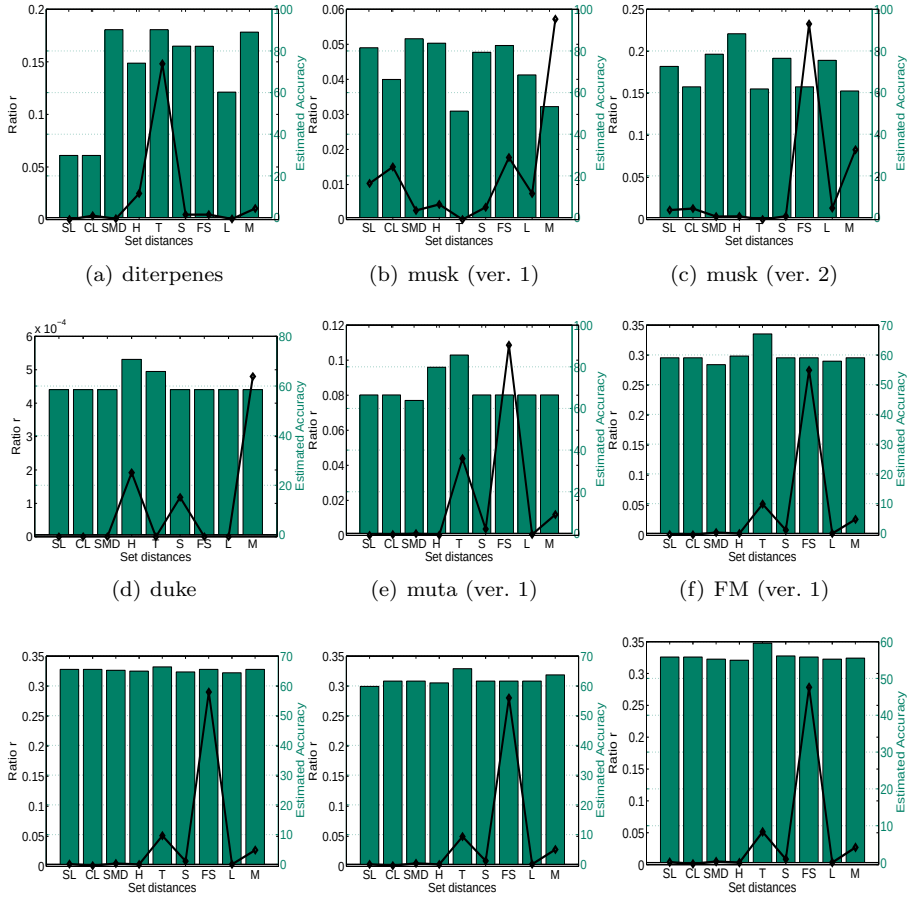


(a) diterpenes (b) musk (ver. 1) (c) musk (ver. 2)

(d) duke (e) muta (ver. 1) (f) fm (ver. 1)

(g) fr (ver. 1) (h) mm (ver. 1) (i) mr (ver. 1)

Figure 4.2: Spectra of Gram matrices of set kernels used within SVM_{SP}. The solid line denotes the r ratio from Equation 4.35 (left y-axis) and bars denote the estimated accuracies (right y-axis).



(a) diterpenes (b) musk (ver. 1) (c) musk (ver. 2)

(d) duke (e) muta (ver. 1) (f) FM (ver. 1)

(g) FR (ver. 1) (h) MM (ver. 1) (i) MR (ver. 1)

Figure 4.3: Spectra of Gram matrices of set distance substitution kernels. The solid line denotes the r ratio from Equation 4.35 (left y-axis) and bars denote the estimated accuracies (right y-axis). The γ parameter is fixed to 1.

performs poorly, indicating that matching specific components of the molecules is more informative. Finally, for the diterpenes and musk (ver. 2) datasets we see that there is an advantage of the kernels based on matchings of specific elements of two sets over kernels that match everything with everything.

Comparison of SVM vs. kNN We also compared the performance of a standard kNN algorithm on the standard set distance measures with that of SVM_P in order to establish whether the latter indeed brings some improvement over the naive way of exploiting distance measures (these results are listed in Table 4.9 [5]). Indeed, kNN was significantly better than SVM_P in only 6 cases and it was significantly worse in 25 cases. For the remaining set distance measures (59) the differences were not statistically significant.

To analyze whether the above improvement simply comes from the use of SVM, we compared SVM_P with kNN applied in the same proximity spaces, kNN_P (these results are also listed in Table 4.9). Overall, the results are similar to ones presented above: SVM_P performed significantly better (worse) in 24 (7) cases; in 59 the differences were not significant. Nevertheless, for some datasets (both versions of musk and FM) the performances of SVM_P and kNN_P are very similar (according to statistical significance results) and the former performed significantly better than standard kNN. For the other datasets we conjecture that the bias introduced by SVM is important.

The better performance of SVM_P and kNN_P in the musk and FM datasets in comparison with the "standard" kNN can be also explained by the fact that working in the proximity space gives a more global view to the data. More precisely, the kNN algorithm in a non proximity space examines a single neighborhood of a given instance which is to be classified. On the other hand SVM_P and kNN_P in the proximity space have access to all the neighborhoods of points in the representation set. These neighborhoods are precisely given by the instances in the new space. In particular, it means that kNN in the proximity space has access to more training points, hence kNN in the initial space would probably achieve better results for higher values of the parameter k. To check the above hypothesis we examined the performance of the standard kNN in the version 1 of musk dataset for two set distance measures d_{RIBL} and d_M where the number of nearest neighbors k was fixed to 9 (these set distance measures achieve the poor accuracy of 50 % for $k = 1$). For $k = 9$ the performance increased by more than 15 % reaching 65.22% and 70.65 %, respectively.

Summary of the Results To sum up, in our experiments we observed several findings: (i) kernels in the proximity space (k_P) outperform distance substitution kernels (k_{DS}) and kernels directly based on specific pairs of elements (k_{SP}) – this is due to the non-PSD-ness of k_{DS} and k_{SP} (or to better suitability of k_P), (ii) as a result of many negative eigenvalues in the corresponding spectra k_{DS} outperforms k_{SP}, (iii) the appropriate way of matching the elements of two sets depends on the actual application (kernels based on specific pairs vs. kernels

[5] In contrast with Tables 4.6, 4.7 and 4.8 we report results also for d_{RIBL}.

	diterpenes		musk (ver. 1)		musk (ver. 2)	
	SVM_P	kNN_P	SVM_P	kNN_P	SVM_P	kNN_P
d_{SL}	82.50 (+)(+)	77.38	79.35 (=)(=)	89.13	82.35 (=)(+)	89.13
d_{CL}	52.03 (-)(+)	70.73	79.35 (=)(=)	80.43	80.39 (=)(+)	76.47
d_{SMD}	94.54 (+)(=)	89.22	94.74 (=)(+)	92.39	92.24 (=)(+)	87.25
d_H	73.85 (-)(-)	85.83	83.70 (=)(=)	85.87	91.27 (+)(+)	84.31
d_{RIBL}	94.88 (+)(=)	89.42	83.70 (=)(+)	82.61	77.45 (=)(+)	74.51
d_T	94.21 (-)(-)	96.21	51.09 (=)(=)	51.09	61.76 (=)(=)	61.76
d_S	93.68 (=)(-)	92.81	79.35 (-)(=)	90.22	83.33 (=)(=)	87.25
d_{FS}	93.68 (=)(-)	92.81	90.22 (-)(+)	96.74	75.49 (=)(=)	72.55
d_L	93.48 (+)(=)	88.96	84.78 (=)(=)	84.78	88.24 (+)(+)	77.45
d_M	93.48 (=)(-)	93.21	84.78 (=)(+)	82.61	78.43 (=)(+)	72.55
	duke		muta (ver. 1)		FM (ver. 1)	
	SVM_P	kNN_P	SVM_P	kNN_P	SVM_P	kNN_P
d_{SL}	68.29 (=)(=)	51.22	79.79 (=)(=)	78.72	58.74 (=)(=)	55.01
d_{CL}	68.29 (=)(+)	53.66	67.02 (-)(-)	76.06	54.73 (-)(=)	61.32
d_{SMD}	68.29 (=)(=)	53.66	78.19 (=)(=)	80.85	62.46 (+)(=)	55.59
d_H	70.73 (=)(=)	68.29	73.94 (=)(=)	77.13	61.03 (=)(=)	55.87
d_{RIBL}	70.73 (=)(=)	58.54	81.91 (=)(=)	85.11	61.32 (+)(=)	52.15
d_T	60.98 (=)(=)	58.54	85.64 (=)(=)	82.98	64.47 (+)(=)	57.88
d_S	65.85 (=)(=)	65.85	81.91 (=)(=)	82.45	62.46 (=)(+)	59.89
d_{FS}	58.54 (=)(=)	58.54	79.79 (=)(=)	85.11	57.02 (=)(=)	55.01
d_L	60.98 (=)(=)	60.98	78.19 (=)(=)	80.85	61.89 (=)(=)	57.31
d_M	68.29 (=)(=)	65.85	83.51 (=)(+)	83.51	62.46 (+)(=)	52.72
	FR (ver. 1)		MM (ver. 1)		MR (ver. 1)	
	SVM_P	kNN_P	SVM_P	kNN_P	SVM_P	kNN_P
d_{SL}	64.67 (+)(=)	55.56	63.99 (=)(=)	59.82	55.81 (=)(=)	57.56
d_{CL}	64.39 (+)(+)	56.13	58.33 (=)(+)	63.99	53.20 (=)(+)	51.16
d_{SMD}	67.81 (+)(+)	54.99	65.48 (=)(=)	60.42	57.56 (=)(=)	59.30
d_H	63.53 (+)(=)	57.83	61.01 (=)(=)	60.12	53.49 (=)(=)	56.69
d_{RIBL}	64.10 (+)(=)	55.56	62.20 (+)(+)	55.65	57.56 (=)(=)	52.33
d_T	65.53 (=)(=)	59.83	66.37 (+)(+)	55.36	64.53 (=)(+)	60.76
d_S	67.24 (+)(=)	49.29	66.67 (+)(+)	59.82	58.14 (=)(=)	54.07
d_{FS}	63.82 (+)(=)	53.56	57.74 (=)(=)	58.33	54.94 (=)(=)	53.78
d_L	66.95 (+)(=)	53.28	66.07 (=)(=)	63.99	60.47 (=)(=)	54.07
d_M	67.81 (+)(=)	52.99	65.48 (+)(=)	57.14	61.63 (+)(+)	53.78

Table 4.9: Accuracy, ranks and significance test results on the considered datasets. The first parenthesis corresponds to the comparison of SVM_P vs. kNN_P and the second compares SVM_P vs. kNN.

based on averaging), (iv) SVM_P (and in some cases kNN_P) performs better than standard kNN (in the proximity space algorithms have a more global view to the data, hence separability is increased).

Comparison with Other Systems Finally, we situate the performance of our relational kernel to other relational learning systems and the best cross

product kernel (k_{CP}) of Section 4.4.3. The results for our kernels are presented in Table 4.5 under the column "SVM_{best} from Section 4.4.4". In the version 1 of musk, for SVM_{P} with d_{SMD}, we obtained 94.54 % of accuracy which the best result reported so far. Moreover, in duke, FM and MM the results are better than both the best results reported in the literature and the results of k_{CP}, however, the difference in accuracy with k_{CP} is statistically significant only in FM. In other datasets the results are worse than either the other relational kernels or the best k_{CP}. From the results reported above we can see that our kernel-based learner compares favorably with the results achieved by special-purpose algorithms applied to structured data. Finally, we mention that except for diterpenes, mutagenesis and duke the performances of our best kernels is better than the performances of best distances (kNN_{best} column).

4.4.5 Experiments with Kernels on Lists

In this section we will perform experiments on the protein fingerprints dataset where we analyze the performance of different kernels defined over lists. More precisely, the considered list kernels are the Contiguous Sublist kernel (k_{CS}) from Equation 4.28 and the Longest Common Sublist kernel (k_{LCS}) from Equation 4.29. We experiment with two different representations of the protein fingerprints dataset. The first approach is identical to the one used in Section 3.5.3 and is based on relating protein fingerprints with a list of motifs. The second representation is based on combination of representations based on sets and lists such that a given fingerprint is associated both with the set and list of motifs. Finally, all the results are reported for the weighted version of the dataset (see description in Section 3.5.3)[6].

In all the experiments we used only the SVM classification rule to decide the classification of a given instance. The performance is measured using both 10-fold cross validation and on an independent test set. The regularization parameter C was optimized in an inner 10-fold cross validation loop over the set $C = \{0.1, 1, 10, 50\}$. Similar to the experimental setup from Section 4.4.3 the exploited vector kernels are the linear, polynomial and Gaussian RBF kernels. In polynomial kernel we report results for $p = \{2, 3\}, l = 1$ whereas in Gaussian RBF kernel we fix $\gamma = \{0.1, 1, 10\}$. In k_{CS} we fixed the λ parameter to 0.5. Finally, we report results only for the tensor product kernel k_{Π}. The results are presented in Table 4.10. In the "lists" column the sign compares the representation based on lists with the representation based on sets from Table 4.2 (with k_{Π}). In the "sets and lists" column the sign compares the representation based on both sets and lists with the representation based only on lists.

[6]In the case where the representation based on both sets and lists is used, both of these complex objects are assigned the same weight.

	lists		sets and lists	
Kernel	10-fold CV	test set	10-fold CV	test set
	k_{CS}			
k_{lin}	83.12 (2.5)(=)	78.31	82.99 (2.5)(=)	83.25
$k_{poly,p=2,l=1}$	86.01 (7.5)(=)	82.82	87.09 (9.5)(+)	87.01
$k_{poly,p=3,l=1}$	86.48 (9.0)(=)	83.63	87.69 (10.5)(+)	87.35
$k_{RBF,\gamma=0.1}$	85.81 (7.5)(=)	82.53	86.15 (7.5)(=)	85.68
$k_{RBF,\gamma=1}$	86.35 (8.5)(=)	83.66	86.15 (7.0)(=)	83.94
$k_{RBF,\gamma=10}$	80.23 (0.5)(=)	78.59	78.55 (0.5)(-)	76.90
	k_{LCS}			
k_{lin}	83.19 (2.5)(=)	78.31	82.78 (2.5)(=)	83.32
$k_{poly,p=2,l=1}$	85.27 (6.5)(=)	82.82	85.34 (6.0)(=)	85.68
$k_{poly,p=3,l=1}$	85.68 (7.5)(=)	83.66	85.88 (7.0)(=)	85.68
$k_{RBF,\gamma=0.1}$	85.34 (7.0)(=)	82.82	85.00 (6.0)(=)	85.73
$k_{RBF,\gamma=1}$	85.54 (6.5)(=)	84.51	85.88 (6.5)(=)	83.66
$k_{RBF,\gamma=10}$	79.69 (0.5)(=)	78.59	79.29 (0.5)(=)	77.18

Table 4.10: Accuracy, rank and significance test results (SVM) on the weighted version of the protein fingerprints dataset.

Classification Performance

The first observation from the results in Table 4.10 is that, in terms of predictive performance, the two representations based on lists and sets[7] are equivalent, i.e. for all the vector kernels the differences in predictive performances estimated using 10-fold CV are not statistically significant. A similar observation holds for the results computed on the independent test set. This is in contrast with the results presented in Section 3.5.4, obtained using kNN, where the representation based on sets showed better performance.

The other observation is that there is an advantage of the contiguous sublist kernel, k_{CS}, over the longest common sublist kernel, k_{LCS}. Indeed, for the representation based on lists the average rank of all the vector kernels used together with k_{CS} is 5.92, while the average rank of kernels combined with k_{LCS} is 5.08. For representation based on both sets and lists the corresponding values are 6.25 and 4.75. This could be a result of the fact that in comparison with k_{LCS}, k_{CS} takes a more "global" view on the compared lists, since it takes into account all (consecutive) sublists of the same length. In contrast, k_{LCS} focuses only on the longest common contiguous sublist at the start of the two lists.

When comparing performances between representations based on lists, and on both sets and lists, we note that except for 3 cases the differences in performances are not statistically significant; in two cases the latter representation was better, in one case it was worse. The best results are obtained for the $k_{poly,p=3,l=1}$ elementary kernel together with the Contiguous Sublist kernel k_{CS} and where the representation of the data based both on sets and lists is used. The estimated cross-validation accuracy is 87.69 % whereas the holdout accu-

[7]Results obtained using the representation based on sets are presented in Table 4.2.

racy is 87.35 %. This represents a statistically significant improvement over the best accuracy presented in Tables 3.6, 4.2 and 4.4. Moreover, this result is significantly better than the one presented in (Hilario et al., 2004) by 1.78 % for cross-validation, and 1.43 % for the holdout test set.

4.5 Related Work

In this section we describe some of the kernels over complex objects which are the most relevant in our context. We also report the previous work on distance substitution kernels and proximity spaces. For an overview of other kernels defined over non-vectorial spaces the reader is refered to (Gärtner, 2003).

Kernels on Sets

The most popular approach for constructing a PSD kernel over sets is based on computing different affinity measures between the Probability Density Functions (PDF) fitted to the elements of the two sets (Kondor and Jebara, 2003; Lyu, 2005a; Moreno et al., 2004; Cuturi et al., 2005; Jebara et al., 2004; Lafferty and Lebanon, 2002). For example, the PDFs in (Kondor and Jebara, 2003) and in (Lyu, 2005a) are Gaussians and mixtures of Gaussians which are compared using the Bhattacharyya affinity and expected likelihood[8], respectively. Non-parametric distribution estimates were considered e.g. in (Hein and Bousquet, 2005) where the distributions are estimated using histograms and the kernels are built from similarity measures on histograms. A more flexible approach was proposed in (Cuturi et al., 2005) where the authors considered a general class of kernels between (molecular) measures or densities defined over the space of elements of the two sets. The final kernel is defined as a measure of dispersion of the sum of the corresponding measures. The authors prove that several functions that quantify the dispersion of measures through their entropy (or through their generalized variance) result in a PSD kernel. The standard cross product (CP) kernel defined in Section 4.2.3, which amounts to computing an inner product between two means of the corresponding PDFs in the feature space, can be seen as a simple example of such kernels. Similar argument holds for a specialized kernel for multiple-instance problems from (Gärtner et al., 2002) which, in case of a Gaussian RBF elementary kernel, amounts to the standard CP kernel.

A somehow related kernel is the Fisher kernel of Jaakkola and Haussler (1999), initially proposed for sequences over finite alphabet, but being general enough to be applied for general objects over which a PDF can be defined. Here a (parametric) distribution is represented as a point in a Riemann space with the coordinates specified by the partial derivatives of the log-likelihood of the distribution with respect to the model parameters. The final kernel is defined as an inner product in this gradient space. This kernel is often referred to as the practical Fisher kernel. The theoretical Fisher kernel is normalized by the

[8]The Bhattacharyya's affinity and the expected likelihood similarity measures are variants of the kernel between distribution from (Jebara et al., 2004).

Fisher information matrix, which is not feasible to compute for most of practical tasks and is usually omitted.

The above approaches for building set kernels suffer from several drawbacks. First, these kernels assume that sets are large enough to accurately estimate the distribution parameters, whereas in real data it is not always the case. In particular, for sets with a small number of elements it is hard to estimate the corresponding PDFs (Duda et al., 2001). Second, if the underlying PDFs can be estimated well enough, a Bayesian framework would be probably more appropriate (Desobry et al., 2005). Last, the "averaging" property might be inadequate for some applications (e.g. multiple-instance problems). The kernels presented in Section 4.2.4 address precisely the last shortcoming of the above kernels. By exploiting the semantics of different set distance measures the proposed kernels are not based on averaging, instead they take into account similarities only between specific pairs of elements from the two sets.

A geometrical approach for building set kernels was presented in (Wolf and Shashua, 2003) where the concept of principal angles between two linear subspaces is exploited. This kernel has a cubic complexity and is only PSD for sets of equal cardinality. A similar idea was developed in (Shashua and Hazan, 2005) where an algebraic kernel based on a group invariant tensor product is used to combine similarities given by local (vector-based) kernels. Again, the above kernels can be seen as averaged similarities of the elements of the two sets.

Kernels over sets which are based on specific pairs of elements were considered mainly in the computer vision community (Wallraven et al., 2003; Lyu, 2005b; Boughorbel et al., 2004). The underlying idea in these kernels is that the actual matching should focus on the most important elements of the sets, neglecting the elements which are likely to introduce noise. The kernel defined in (Wallraven et al., 2003) can be seen as a variant of the Sum of Maximal Kernels from Equation 4.21. Despite the claim of Wallraven et al. (2003), the kernel is not PSD (see Example 4.3). The kernel from (Lyu, 2005b) tries to overcome this limitation by raising the elementary kernel between each pair of elements to a given power, p, such that the final kernel becomes similar to k_{SMD} of Equation 4.21 but it is still PSD. With $p = 1$, the proposed kernel includes the standard CP kernel as a special case. However, as p becomes larger, the more dominant pair is the best matched pair, and in the limit of large p we obtain k_{SMD}. An interesting alternative is presented by Boughorbel et al. (2004) where a greedy algorithm is used to search for the optimal matching between two sets. A similar idea to the above kernels was proposed in the context of kernels over graphs (Fröhlich et al., 2005) where only specific elements of decompositions of graphs into their parts are considered. This kernel has been recently shown to be non-PSD by Vert (2008). The actual mapping considered in this work is such that the sum of similarities (kernels) of the matched elements is maximum. In this sense this kernel is similar to kernels based on relations between two sets (i.e. k_S, k_{FS}, k_L and k_M). In contrast, the kernels considered in this study are more flexible since they allow for more flexible matchings of elements between the two sets.

Recently, an efficient matching-based kernel called the pyramid match kernel was proposed by Grauman and Darrell (2007). It approximates the optimal partial matching by computing a weighted intersection over multi-resolution histograms with different sizes of bins. Two elements from two sets are considered matched if they fall in the same bin. The correspondences between elements are implicit in the sense that matches are counted and weighted according to their strength, but the specific pairwise links between points do not to be individually enumerated. This characteristic renders the complexity of the pyramid match kernel to be linear in the cardinalities of sets.

Kernels on Lists

Although many kernels have been recently proposed for sequences over a finite alphabet (Leslie et al., 2003; Saigo et al., 2004; Lodhi et al., 2002; Viswanathan and Smola, 2003; Tsuda et al., 2002), not much work has been done for defining kernels over lists of complex objects. The exceptions are kernels described in (Zelenko et al., 2003) and (Gärtner et al., 2004), variants of which are used in this study (Section 4.2.5). The main difference between the k_{CS} kernel from Equation 4.28 and the Contiguous Subtree kernel from (Zelenko et al., 2003) is that the latter uses an additional user defined matching function. For the trees we consider it is the syntax of relational object representation that determines whether two nodes are matchable or not. The other difference is that kernels from (Zelenko et al., 2003) are highly specialized for the task of information extraction where the instances are shallow parse tress (i.e. ordered trees where nodes are labeled with vectors) whereas our kernels are not domain-specific and can be used for any classification problem. The kernel from Equation 4.29 is a direct application of the kernel over basic terms defined in (Gärtner et al., 2004). The main difference is that the latter kernel was only used for sequences over a finite alphabet (with matching kernel for sequences' elements) whereas we applied it to more complex structures.

Most of the existing kernels for sequences were developed in the context of computational biology (Leslie et al., 2003; Saigo et al., 2004; Tsuda et al., 2002). The kernel from (Leslie et al., 2003) implicitly induce a feature space indexed by all the l-length sub-sequences from a given string alphabet. If each feature is simply a count of the number of exact occurrences of each of the sub-sequences, we obtain the *Spectrum kernel*. In general, if we allow for some inexact string matching we obtain more general feature maps, such as mismatch, wildcards, etc. The main problem with this kernel is that it is designed only for sequences over finite alphabet and it is not straightforward to extend it so that it can handle general lists. The kernel of Saigo et al. (2004) is based on the detection of local alignment in strings and in that sense is similar to the edit distance measure from Section 3.2.4. The kernel from (Tsuda et al., 2002) is a generalization of the Fisher kernel of Jaakkola and Haussler (1999).

Kernels on Trees

The two kernels most relevant to the kernels over trees from Section 4.3 are the kernels from (Collins and Duffy, 2002) and (Kashima and Koyanagi, 2002). These kernels can be considered as specialized $\Re$-Convolution kernels where instances are labeled, ordered and directed trees. The main idea is based on the notion of counting common subtrees in a tree i.e. the kernel function is the inner product in the space which describes the number of occurrences of all possible subtrees. The main difference between the kernels from (Collins and Duffy, 2002) and (Kashima and Koyanagi, 2002) is that the former is applicable only to trees where no node shares its label with any of its siblings. (Kashima and Koyanagi, 2002) overcomes this limitation by defining the substructures of a tree as a tree such that there is a descendant order preserving mapping from vertices in the substructure to vertices in the tree. There are many differences between our kernel and kernels defined in (Kashima and Koyanagi, 2002). First, the trees considered in (Kashima and Koyanagi, 2002) are labeled trees, i.e. each node is characterized by a symbolic label so two nodes are either the same or different, there is no graded similarity. In our case however nodes can be associated with more general labels, resulting in the definition of a graded similarity. The only restriction is that the comparison is performed only between subtrees rooted at nodes of the same "type" and only descendants of the same type can be compared. Second, the trees in (Kashima and Koyanagi, 2002) are ordered whereas we consider both ordered and unordered trees. The difference in time complexity between our kernel and the kernels of (Collins and Duffy, 2002; Kashima and Koyanagi, 2002) comes mainly from the fact that the input trees for our kernel are ordered and unordered whereas their kernels operate only on ordered trees.

A general class of kernels over trees was defined in (Passerini et al., 2006). More precisely, the considered trees are prolog proof trees, which are obtained by execution traces of a prolog program taking two complex instances as input. The abstraction from the input instances renders the proposed kernel to be in fact applied on general objects represented in the first-order formalism. The actual kernels are the instantiations of the $\Re$-Convolution kernels which work in a recursive manner similar to the kernel from Section 4.3.

Kernels on Graphs

Due to the rich expressiveness of graphs it has been proved that kernels over arbitrarily structured graphs, taking their full structure into account, can be neither computed (Gärtner et al., 2003) nor even approximated efficiently (Ramon and Gärtner, 2003). The most popular way to tackle this problem is based on decompositions of graphs into subgraphs of specific types which are compared via subkernels. The subgraph types mainly considered are walks (Gärtner et al., 2003; Kashima et al., 2003; Mahé et al., 2004; Borgwardt et al., 2005; Ralaivola et al., 2005; Vishwanathan et al., 2007). However, other researchers have experimented with shortest paths (Borgwardt and Kriegel, 2005), subtrees (Ramon

and Gärtner, 2003; Fröhlich et al., 2005), cyclic and tree patterns (Horváth et al., 2004; Horváth, 2005) and limited-size general subgraphs centered at each vertex (Menchetti et al., 2005).

The existing kernels based on decompositions have two main limitations. First, it is difficult in general to specify in advance the appropriate type of substructures for a given problem. The proper representation of learning examples should be determined by domain knowledge and the application requirements, however, in practice we rarely have a solid description of the learning problem. Second, most of the decomposition-based kernels combine substructures using the cross product kernel which accounts for *all* possible substructures of a given decomposition. This might adversely affect the generalization of the final classifier since most of the substructures-subgraphs, and hence attributes in the induced feature space, will be poorly correlated with the actual class variable (Ben-David et al., 2002; Menchetti et al., 2005). The latter limitation of the graph kernels was considered in Section 4.2.4 where we defined a more flexible class of kernels. In Chapter 5 we will show how we can combine a number of existing graph decompositions.

Most of the kernels on graphs use the δ kernel (i.e. $k_\delta(x,y) = 1 \iff x = y, k_\delta(x,y) = 0$ otherwise) on subparts. As a result the ability to find partial similarities is lost and the expressivity of these kernels is reduced (Borgwardt and Kriegel, 2005). A graded similarity on walks was considered in the graph kernel presented in (Borgwardt and Kriegel, 2005), however it suffers from high computational complexity since it requires taking powers of the adjacency matrix of the direct product graph, leading to huge runtime and memory requirements (Borgwardt et al., 2005). Graded similarities on trees were also used in (Fröhlich et al., 2005).

In parallel to kernels based on comparison of particular subgraphs the other line of research focuses on special kinds of graphs such as strings, trees and nodes in graphs (Gärtner et al., 2003). The resulting kernels are efficient, however, they lose most of the modeling power of general graphs. An alternative direction explicitly controls the dimensionality of the feature space by generating sets of connected graphs that occur frequently as subgraphs in the graph database and this frequency is beyond a user defined threshold (Deshpande et al., 2003; Kramer et al., 2001; Horváth et al., 2006; Horváth et al., 2006; Rückert and Kramer, 2004). These kernels form an attractive alternative to kernels based on decompositions since the corresponding feature space is constructed explicitly. On the other hand these methods bear difficulties since their efficiency is threshold-dependent. It should be mentioned it is also possible to automatically extract features which are maximally class-correlated, or maximally diverse (Rückert and Kramer, 2007; Bringmann et al., 2006). Yet another approach was recently proposed in (Mahé et al., 2006) where the kernel takes into account the 3D structures of the molecules. This kernel, although computationally more expensive, achieved good performance on the task of target binding of molecules where it is known that traditional 2D representations are not sufficient.

For an overview of kernels over graphs the reader is referred to (Gärtner

et al., 2007; Wrobel et al., 2006).

Kernels on General Structures

Gärtner et al. (2004) proposed a framework that allows the application of kernel methods to different types of structured data e.g. sets, trees, graphs, lists. The representation formalism used was that of typed λ-calculus. The representation framework allows for the modeling of arbitrary complex objects which however is not at all a trivial task. Individuals are represented as terms of the typed λ-calculus formal logic. The composition of individuals from their parts is expressed in terms of a fixed type structure that is made up of function types, product types and type constructors. Function types are used to represent types corresponding to sets and multisets, product types represent types corresponding to fixed size tuples and type constructors for arbitrary structured objects such as lists, trees, graphs. Each type defines a set of terms that represent instances of that type. Kernels are then defined on each of the functions, product types and type constructors, along with a kernel defined on the data constructors associated with each type constructor.

Distance Substitution Kernels

Several experimental results on applying the distance substitution (DS) kernels (Section 4.2.4) have been presented in the literature. The first attempts were based on the χ^2-distance on histograms (Chapelle et al., 1999), tangent-distance (Haasdonk and Keysers, 2002), dynamic-time-warping (Bahlmann et al., 2002) and Kullback-Leibler divergence between distributions (Moreno et al., 2004). More recent examples include the applications of DS kernels for semi-supervised learning problems (Chapelle and Zien, 2005), segmentation of images by spectral partitioning (Belongie et al., 2002b) and M-estimators (Chen, 2004). DS kernels were also used to define kernels on sets of vectors (Desobry et al., 2005), where set distance measures based on level sets of corresponding PDFs, easier to estimate than the PDFs themselves, are "substituted". However, in this kernel the averaging mechanism is also present (see Section 4.5). Haasdonk and Bahlmann (2004) presents a general framework for DS kernels, examines their formal properties and shows that better performance of SVMs with these kernels over standard kNN algorithms can be achieved.

Proximity Spaces

Proximity space was first proposed by Tsuda (1999). However, in (Tsuda, 1999) the space is induced by means of an asymmetric kernel function. The proximity space defined by dissimilarity measures was considered among others in (Graepel et al., 1999; Pekalska et al., 2001). Several experimental results were reported for algorithms in the proximity space: SVM was considered in (Graepel et al., 1999; Pekalska et al., 2001), LP machines and Fisher Linear Discriminant were examined in (Pekalska et al., 2001). Various methods for prototype selection

which amount to feature selection in the proximity space were examined in (Duin et al., 1999; Pekalska et al., 2006).

Proximity spaces induced by set distances based on specific pairs of elements were considered in (Zhang and Malik, 2003) where the application domain is recognition of handwritten digits, and the shape context distance of Belongie et al. (2002a) was used as the set distance measure. The kernel from (Boughorbel et al., 2005) is similar to the Gaussian RBF kernel in proximity space (Equation 4.27) induced by d_{SL} set distance. However, instead of mapping the data points to the $\mathbb{R}^{|S|}$ where S is the representation set, the instances are mapped to one-dimensional space F and the final kernel over sets is defined as the sum of kernels defined in F.

Proximity spaces were also exploited for classifying sequences of proteins (Liao and Noble, 2002). In this case the actual (dis-)similarities exploited to construct the feature space were the pairwise scores obtained from the Smith-Waterman or BLAST algorithms (Durbin et al., 1999).

4.6 Conclusions

Bringing together kernel methods and structured data is an important direction for practical machine learning and data mining research. In this chapter we proposed kernel operators defined over composite objects that are represented using the relational formalism. The resulting kernels are based on decompositions of complex objects into simpler parts of various types (i.e. primitives, tuples, sets and lists) and the combination of kernels on these simpler parts. The analyst can define a new composite kernel, that reflects as closely as possible the semantics of a problem at hand, by simply declaring which kernel should be used for a given data type.

We extended the flexibility of existing kernels over sets by allowing only specific elements from the two sets to be matched such that the resulting set kernels are not based on averaging. Within this context we proposed three flexible families of set kernels, namely: kernels in proximity space induced by set distances, set distance substitution kernels and kernels directly based on specific mappings. The last class of kernels naturally generalizes the standard cross product (CP) kernel and a number of existing kernels based on explicit correspondences between sets' elements. The semantics of the proposed set kernels are similar to the semantics of various set distances considered in Chapter 3.

In the experiments we examined the performance of the CP set kernel, set kernels based on specific mappings and kernels on lists. The CP kernel was in general stable with respect to the parameter settings of the vectorial kernels, different normalization schemes and various choices of kernels on tuples. Moreover, we observed that in comparison with kNN using kernel induced distances, the bias introduced by SVM was usually beneficial. The experimental results with more flexible set kernels showed that the kernels in the proximity space outperform the set distance substitution kernels and kernels directly based on various mappings. We also argued that when dealing with set problems the standard

approaches based on averaging do not necessarily provide the best performance. We can get significant gains in classification performance by focusing on specific types of matchings between elements of the two sets. Finally, the experiments with lists kernels showed better performance of the contiguous sublist kernel over the longest common sublist kernel. The experiments indicated that on a range of real-world datasets our kernel-based learner compares favorably with other state-of-the-art algorithms.

Chapter 5

Adaptive Approaches

The correct choice of representation for the learning instances and the data mining / machine learning operators to be applied on the chosen representation are two crucial components for the successful application of learning algorithms. Preferably, both of them should be adapted for a given problem in a manner that reflects the important relationships between features in the data; however, this has been proved to be difficult in practice. For example, the experimental evidence from Chapters 3 and 4 clearly indicated that there is no distance (kernel) which is overall better than any other, and in the context of graph datasets the optimal representations of learning instances depends on the problem at hand. As a result, for a given problem there might exist several plausible operators which are defined over different representations and reflect different aspects of the data. An obvious question is how to chose the pair of representation and operator that best fits the requirements of the problem at hand. A simple solution that was examined in the previous chapters is to select these pairs by cross-validation; however, the main drawback of this approach is that only one representation-operator combination per training set is selected, and hence the expressiveness of the resulting method is limited. Additionally, this method requires the use of extra data.

In this chapter we address the above issues and propose a general framework for combining representations of complex data and/or the corresponding operators. Couplings of representations and operators are the building blocks of our method, the relative importance of which is learned directly from the training data. This approach allows for the simultaneous combination of representations and operators, nevertheless one can choose to focus on only one of them, i.e. one can apply a fixed operator on different decompositions, or apply different operators which are defined on the same representation.

We only focus on the distance-based paradigm since, as already mentioned, distances over composite objects are in general more flexible than composite kernels; in the latter case the options are much more limited due to the requirement of positive semi-definiteness. In any case the ideas presented in this chapter can be easily extended to kernels. We assume that learning objects are internally

modeled as tuples where each dimension corresponds to a particular representation of the instances and on each of the representations a given operator is applied. The actual combination of the building blocks is achieved by the use of a Mahalanobis distance measure (parameterized by a positive semi-definite matrix) between the corresponding tuples. In order to learn the parameters of this distance measure we exploit three metric learning methods from (Xing et al., 2003; Globerson and Roweis, 2006; Goldberger et al., 2005), originally developed for propositional data, and adapt them so that they can be used in the context of complex structures. The proposed methods boil down to mathematical optimization problems which are amenable to various optimization techniques. We empirically evaluate our framework for the tasks of combination of set distance measures, and combination of different graph decompositions into sub-graphs of specific types. In the latter case we exploit the adaptive sub-graphs combination to construct a flexible and powerful class of graph kernels which are defined in proximity space (Section 4.2.4).

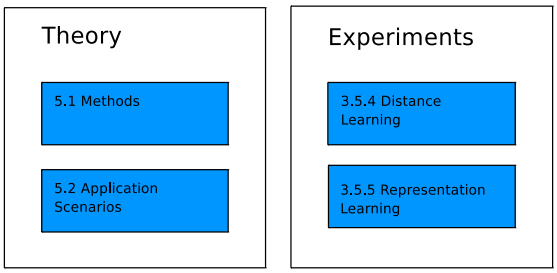


Figure 5.1: Schematic structure of Chapter 5.

This chapter is organized as follows. In Section 5.1 we propose a framework for learning combinations of distance measures and representations on complex objects. In particular, in Section 5.1.1 we review some of the definitions which will be used in the remainder of this chapter, and define the general optimization method. Next, in Sections 5.1.2, 5.1.3 and 5.1.4 we present the three instantiations of this framework. In Section 5.1.5 we address the problem of regularization of the solutions, and in Section 5.1.6 we describe how the corresponding optimization tasks are solved. In Section 5.1.7 we discuss the computational complexity of the proposed techniques. In Section 5.2 we describe the application scenarios of the proposed framework. In particular in Section 5.2.1 we discuss the adaptive graph kernels which are constructed by combining various graph decompositions. In the experiments we present the performance of the proposed framework for the tasks of combination of distances (Section 5.3) and representations of labeled graphs (Section 5.3). In Section 5.5 we present related work and we conclude with Section 5.6.

The organization of this chapter is schematically presented in Figure 5.1.

5.1 Adaptive Methods

We view the problem of representation and/or distance combination as an optimization problem where the two main constituents are the definition of a parameterized cost function that depends on the class labels (or in general on some form of side-information) and an optimization method. In this work we only focus on methods which combine representations (or distance measures) in a *global* manner as opposed to *local* methods which aim to determine a "stretched" neighborhood around each query instance such that class conditional probabilities are likely to be constant (Hastie and Tibshirani, 1996; Domeniconi and Gunopulos, 2002; Yang et al., 2006). Local methods form an interesting alternative and were shown to achieve better performance for data exhibiting "difficult" distributions; however, global methods have the advantage of providing insight into the underlying structure of the data which might be subsequently used e.g. for dimensionality reduction.

In this study, in order to render the optimization task computationally efficient and amenable to theoretical analysis, we take the view that the top level representation of complex objects consists of tuples, the elements of which correspond to m different representations. This may seem too restrictive; however, we do not put any restrictions on the type of structures contained in the tuples and, as we will see in the experiments, this representation allows for the modeling of a wide class of composite objects, or approximations of composite objects. Over the tuples we use the Mahalanobis distance measure defined in Section 3.14 which depends on a number of continuous parameters. In general the number of these parameters scales with $O(m^2)$; however, we can also use a parameterization based on $O(m)$ parameters. Depending on whether the goal is to combine representations or distance measures we can apply a fixed distance measure on different representations, or apply different distance measures which are defined on a fixed representation, or both. The parameters of the Mahalanobis distance measure will reflect the importance of the decompositions and of the distance measures. We mention that the Mahalanobis distance measures can be used to define cost functions that are differentiable with respect to the parameters such that the machinery of the mathematical optimization theory can be exploited (Avriel, 2003). Finally, it is sometimes possible to define these cost functions such that they are convex with respect to the parameters of the Mahalanobis distance measures; in these cases an unique global solution can be guaranteed (Boyd and Vandenberghe, 2004).

5.1.1 General Optimization Method

We begin with a labeled set of n complex objects $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ where $\boldsymbol{x}_i \in \boldsymbol{\mathcal{X}}$ and $y_i \in \{1, 2, \ldots, c\}$. We assume that each complex object $\boldsymbol{x}_i$ is given by a tuple $(x_{i_1}, \ldots, x_{i_m})^T$ where $x_{i_l} \in \mathcal{X}_l$ and $\mathcal{X}_l$ $(l = 1, \ldots, m)$ are different representation spaces of $\boldsymbol{x}_i$. In particular $\boldsymbol{x}_i \in \boldsymbol{\mathcal{X}} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_m$. On each of the $\mathcal{X}_l$ a (possibly different) distance measure $d_l : \mathcal{X}_l \times \mathcal{X}_l \to \mathbb{R}_0^+$ is defined. We can now define two variants of the Mahalanobis distance measure over tuples $\boldsymbol{x}_i, \boldsymbol{x}_j \in \boldsymbol{\mathcal{X}}$ as

follows

$$d^2_{tuple,\boldsymbol{A}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{d}(\boldsymbol{x}_i, \boldsymbol{x}_j)^T \boldsymbol{A}\ \boldsymbol{d}(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{5.1}$$

and

$$d^2_{tuple,\boldsymbol{W}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{d}(\boldsymbol{x}_i, \boldsymbol{x}_j)^T \boldsymbol{W}^T \boldsymbol{W} \boldsymbol{d}(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{5.2}$$

where $\boldsymbol{A}, \boldsymbol{W} \in \mathbb{R}^{m \times m}$, $\boldsymbol{A} = \boldsymbol{W}^T \boldsymbol{W} \succeq 0$, i.e. it is positive semi-definite, and $\boldsymbol{d}(\boldsymbol{x}_i, \boldsymbol{x}_j) = (d_1(x_{i_1}, x_{j_1}), \ldots, d_m(x_{i_m}, x_{j_m}))^T$. We mention, that the distance measures from Equations 5.1 and 5.2 are the same as defined in Section 3.2.2. As already mentioned, the above quadratic forms have the Mahalanobis distance over vectorial data as a special case. In this case the elements of the tuples are simply numbers and the (absolute) difference between these numbers is used as a distance, i.e. $x_{i_l} \in \mathbb{R}$ and $d_l(x_{i_l}, x_{j_l}) = |x_{i_l} - x_{j_l}|$ for $l = 1, \ldots, m$. The fact that we use the Mahalanobis distance measure defined over general tuples allows for adaptive learning on complex data, as long as distance measures over the corresponding representations are defined.

It should be noted that the matrix $\boldsymbol{W}$ (and $\boldsymbol{A}$) implicitly defines a transformation of the data by providing a set of "useful" directions in the input space. More precisely, in the case of Euclidean space (with d_l being the absolute differences between real numbers) the application of the $d_{tuple,\boldsymbol{A}}$ (or $d_{tuple,\boldsymbol{W}}$) distance measure is equivalent to finding an explicit transformation that replaces each point $\boldsymbol{x}_i$ with $\boldsymbol{W}\boldsymbol{x}_i$, and applying a standard Euclidean metric on the rescaled data. In general, this transformation is implicit and depends on the topology induced by the distance measures d_l.

For an optimization problem where the objective function is optimized with respect to matrix $\boldsymbol{A}$ it is necessary to ensure that $\boldsymbol{A} \succeq 0$ such that $d_{tuple,\boldsymbol{A}}$ is a pseudo-metric (if all d_l are pseudo-metrics), or at least $d_{tuple,\boldsymbol{A}}$ is non-negative (see Proposition 3.4). On the other hand, an optimization problem where the objective function is optimized with respect to the matrix $\boldsymbol{W}$ is not constrained by $\boldsymbol{W} \succeq 0$ and thus is easier to solve.

A common approach for learning a metric is to provide information in the form of equivalence relations as pairwise constraints on the data. In the classification framework there is a natural equivalence relation in the above form, namely whether two points are in the same class or not, i.e. $\mathcal{S} = \{(\boldsymbol{x}_i, \boldsymbol{x}_j) \mid y_i = y_j\}$ and $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{x}_j) \mid y_i \neq y_j\}$. The general problem of representation and distance combination in a supervised setting can be now stated as the following optimization problem

$$\min_{\boldsymbol{Z}} \mathcal{F}_{\boldsymbol{Z}}(\mathcal{S}, \mathcal{D}, d_{tuple,\boldsymbol{Z}}) \tag{5.3}$$

where $\mathcal{F}_{\boldsymbol{Z}}$ is a differentiable function, $\boldsymbol{Z} = \boldsymbol{A}$ or $\boldsymbol{Z} = \boldsymbol{W}$, and this optimization is subject to some constraints (e.g. $\boldsymbol{A} \succeq 0$ if $\boldsymbol{Z} = \boldsymbol{A}$). Depending on the actual form of the function $\mathcal{F}_{\boldsymbol{Z}}$ and the optimization technique, different instantiations of the algorithm can be obtained.

In the next three subsections we explore three different instantiations of the above framework which differ with respect to the actual objective function which is being optimized. We based our study on methods originally developed

for vectorial data, namely Xing's method (Xing et al., 2003), Maximally Collapsing Metric Learning (MCML) method (Globerson and Roweis, 2006) and Neighborhood Component Analysis (NCA) method (Goldberger et al., 2005). We adapted these methods so that they can learn the Mahalanobis distance measure of the form given in Equation 5.1 (for Xing's and MCML methods) and Equation 5.2 (for NCA). The methods differ with respect to the assumptions they make for the data distribution. In principle any metric learning method where the objective is a function of pairwise distances could be formulated within this framework.

In the rest of this chapter, to simplify the notation, we will denote $\delta_l = d_l(x_i, x_j)$ for $l = 1, \ldots, m$.

5.1.2 Xing's Method

The first method proposed by Xing et al. (2003) was originally developed for clustering with side-information. Adapting from (Xing et al., 2003) we formulate the problem of distance combination as the following optimization task

$$\begin{aligned} \min_{\boldsymbol{A}} \quad & \sum_{(\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathcal{S}} d^2_{tuple, \boldsymbol{A}}(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ \text{s. t.} \quad & \sum_{(\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathcal{D}} d_{tuple, \boldsymbol{A}}(\boldsymbol{x}_i, \boldsymbol{x}_j) \geqslant 1 \\ & \boldsymbol{A} \succeq 0 \end{aligned} \tag{5.4}$$

The above optimization problem is convex and is equivalent (up to a multiplication of $\boldsymbol{A}$ by a positive constant) to minimizing

$$\mathcal{F}_{\boldsymbol{A}} = \sum_{(\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathcal{S}} d^2_{tuple, \boldsymbol{A}}(\boldsymbol{x}_i, \boldsymbol{x}_j) - \log \left(\sum_{(\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathcal{D}} d_{tuple, \boldsymbol{A}}(\boldsymbol{x}_i, \boldsymbol{x}_j) \right) \tag{5.5}$$

subject to $\boldsymbol{A} \succeq 0$ (Xing et al., 2003). Differentiating $\mathcal{F}_{\boldsymbol{A}}$ with respect to the elements of $\boldsymbol{A}$ yields the following gradient rule which we will use for learning

$$\frac{\partial \mathcal{F}_{\boldsymbol{A}}}{\partial A_{kl}} = \sum_{\mathcal{S}} \delta_k \delta_l + \frac{1}{2 \sum_{\mathcal{D}} d_{tuple, \boldsymbol{A}}(\boldsymbol{x}_i, \boldsymbol{x}_j)} \sum_{\mathcal{D}} \frac{\delta_k \delta_l}{d_{tuple, \boldsymbol{A}}(\boldsymbol{x}_i, \boldsymbol{x}_j)}$$

where A_{kl} denotes the k, l-th element of matrix $\boldsymbol{A}$.

From Equation 5.5 it is clear that more emphasis is placed on minimizing the pairwise distances between *all* examples in the same class. Moreover, this method implicitly assumes that instances from each class form a single compact and connected set. In particular, for binary problems where the negative class contains *any* examples which do not have the property encoded by the positive class, the cost function for Xing's method will be severely penalized. A similar problem occurs for data exhibiting highly multi-modal distributions. The other problem with this method is that the use of the squared distance in the minimization term and the non-squared distance for the constraint term is arbitrary and asymmetric.

5.1.3 MCML

The MCML algorithm is based on the simple geometric intuition that all points of the same class should be mapped onto a single location and far from points of the other classes (Globerson and Roweis, 2006). To learn the metric which would approximate this ideal geometrical setup a conditional distribution is introduced which for each example $\boldsymbol{x}_i$ selects another example $\boldsymbol{x}_j$ as its neighbor with some probability $p_{\boldsymbol{A}}(j|i)$, and $\boldsymbol{x}_i$ inherits its class label from $\boldsymbol{x}_j$. The probability $p_{\boldsymbol{A}}(j|i)$ is based on the soft-max of the $d_{tuple,\boldsymbol{A}}$ distance measure, i.e.

$$p_{\boldsymbol{A}}(j|i) = \frac{e^{(} - d^2_{tuple,\boldsymbol{A})(\boldsymbol{x}_i,\boldsymbol{x}_j)}}{\sum_{k\neq i} e^{-d^2_{tuple,\boldsymbol{A}}(\boldsymbol{x}_i,\boldsymbol{x}_k)}} \ , \ p(i|i) = 0 \tag{5.6}$$

It can be shown (Globerson and Roweis, 2006) that any set of points which has the distribution $p_0(j|i) = 1$ if $(\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathcal{S}$ and $p_0(j|i) = 0$ if $(\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathcal{D}$ exhibits the desired ideal geometry. It is thus natural to seek a matrix $\boldsymbol{A}$ such that $p_{\boldsymbol{A}}(\cdot|i)$ is as close (e.g. in the sense of the Kullback-Leibler divergence, KL) to $p_0(\cdot|i)$. This is equivalent to minimizing

$$\mathcal{F}_{\boldsymbol{A}} = \sum_i KL[p_0(\cdot|i)|p_{\boldsymbol{A}}(\cdot|i)] \tag{5.7}$$

subject to $\boldsymbol{A} \succeq 0$. We can rewrite the above objective function in the form[1]

$$\mathcal{F}_{\boldsymbol{A}} = - \sum_{(\boldsymbol{x}_i,\boldsymbol{x}_j)\in\mathcal{S}} \log(p_{\boldsymbol{A}}(j|i)) \tag{5.8}$$

Taking the first-order derivative of $\mathcal{F}_{\boldsymbol{A}}$ with respect to the elements of $\boldsymbol{A}$ we obtain

$$\frac{\partial \mathcal{F}_{\boldsymbol{A}}}{\partial A_{kl}} = \sum_{(\boldsymbol{x}_i,\boldsymbol{x}_j)\in\mathcal{S}} \left(p_0(j|i) - p_{\boldsymbol{A}}(j|i)\right) \delta_k \delta_l.$$

MCML is similar to the well-known Linear Discriminant Analysis (LDA) (Bishop, 2006) in that it tries to minimize within class distances (variance) and maximize between class distances (variances). The main difference between MCML and LDA is that the latter is a purely second order (i.e. "Gaussian") method, i.e. it depends only on the mean of each class and on covariance of points within the same class. MCML is a generalization of LDA that makes a much weaker assumption, namely that each class is uni-modally distributed. It has been shown in (Globerson and Roweis, 2006) that the sufficient statistics used in MCML are n "spread" matrices centered at each training point. The main difference between MCML and Xing's method is that the former puts more emphasis on the pairs of points which are in different classes.

[1] Up to an additive constant $-\sum_i H(p_0(j|i))$, where $H(\cdot)$ denotes the entropy function.

5.1.4 NCA

The NCA method proposed by Goldberger et al. (2005) attempts to directly optimize a continuous version of the leave-one-out error of the kNN algorithm on the training data. The main difference between NCA and the two previous methods is that optimization in NCA is done with respect to the matrix $\boldsymbol{W}$ of Equation 5.2. The actual cost function used is a differentiable function based on stochastic neighbor assignments in the weighted space, which is based on $p_{\boldsymbol{W}}(j|i)$ of Equation 5.6 where $\boldsymbol{A}$ is replaced with $\boldsymbol{W}$. In the remainder we denote the set of points that share the same class with $\boldsymbol{x}_i$ by $C_i = \{j|(\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathcal{S}\}$. Under this stochastic selection rule the probability $p_{\boldsymbol{W}}(i)$ of correctly classifying $\boldsymbol{x}_i$ is given by $\sum_{j \in C_i} p_{\boldsymbol{W}}(j|i)$. One possible objective to minimize is the negative of the expected number of correctly classified points which is given by $-\sum_i \sum_{j \in C_i} p_{\boldsymbol{W}}(j|i)$. Instead, in this work we use the following objective

$$\mathcal{F}_{\boldsymbol{W}} = -\sum_i \log(\sum_{j \in C_i} p_{\boldsymbol{W}}(j|i)) \tag{5.9}$$

which expresses the probability of obtaining an error free classification on the training set (Goldberger et al., 2005). Differentiating $\mathcal{F}_{\boldsymbol{W}}$ with respect to the elements of $\boldsymbol{W}$ yields the following gradient rule

$$\frac{\partial \mathcal{F}_{\boldsymbol{W}}}{\partial W_{kl}} = 2W_{kl} \sum_i (\sum_j p_{\boldsymbol{W}}(j|i)\delta_k\delta_l - \frac{\sum_{j \in C_i} p_{\boldsymbol{W}}(j|i)\delta_k\delta_l}{\sum_{j \in C_i} p_{\boldsymbol{W}}(j|i)}).$$

The main advantage of the NCA method is that it makes no assumptions about the shape of the class conditional distributions or the corresponding boundaries (Goldberger et al., 2005). In particular, nothing is assumed whether the class conditional distributions are uni- or multi-modal. The main problem with the NCA algorithm is that there is no guarantee that a gradient method will converge to the global optima.

5.1.5 Regularization

One possible problem with the optimization task of Equation 5.3 is that for full matrices $\boldsymbol{Z}$ the number of parameters to estimate is $O(m^2)$ (the number of these parameters is precisely $\frac{m(m-1)}{2}$ since $\boldsymbol{Z}$ is assumed to be symmetric). This could be problematic in cases where m is large with respect to the number of instances in the training set and could lead to over-fitting. One possible way to overcome this problem is to add a soft constraint to the objective function, which results in the following regularized optimization task

$$\min_{\boldsymbol{Z}} \{\mathcal{F}_{\boldsymbol{Z}}(\mathcal{S}, \mathcal{D}, d^2_{tuple,\boldsymbol{Z}}) + \lambda\Omega(\boldsymbol{Z})\} \tag{5.10}$$

where $\Omega(\boldsymbol{Z})$ is a regularization term and $\lambda > 0$ is a regularization parameter. For $d_{tuple,\boldsymbol{A}}$ defined in Equation 5.1 we set $\Omega(\boldsymbol{A}) = Tr(\boldsymbol{A})$ i.e. the trace of $\boldsymbol{A}$, whereas for $d_{tuple,\boldsymbol{W}}$ given in Equation 5.2 we set $\Omega(\boldsymbol{W}) = \|\boldsymbol{W}\|^2_{\mathcal{F}}$ where

$\|\cdot\|_{\mathcal{F}}$ denotes the Frobenius norm applied over matrices. It is easy to show that $Tr(\boldsymbol{A}) = \|\boldsymbol{W}\|^2_{\mathcal{F}}$.

The above regularization can be also regarded as a method for reducing the number of features retained by the matrix $\boldsymbol{A}$ (or $\boldsymbol{W}$), meaning that the rank of these matrices should be kept as small as possible (Kwok and Tsang, 2003). More precisely, assume that the eigen-decomposition of $\boldsymbol{A} = \boldsymbol{W}^T\boldsymbol{W}$ is $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{U}^T$. Then $rank(\boldsymbol{A}) = rank(\boldsymbol{\Sigma}) = \|\boldsymbol{\Sigma}\|_0$ where $\|\cdot\|_0$ denotes the zero norm (i.e. number of non-zero elements). Since in general the direct minimization of the zero norm is difficult it can be approximated by the Euclidean norm as $\|\boldsymbol{\Sigma}\|_0 \simeq \|\boldsymbol{\Sigma}\|_2 = Tr(\boldsymbol{A}) = \|\boldsymbol{W}\|^2_{\mathcal{F}}$.

A complementary solution that we will explore is to restrict matrix $\boldsymbol{A}$ (or $\boldsymbol{W}$) to be diagonal resulting in a weighted combination of distances, i.e.

$$d^2_{tuple,\boldsymbol{Z}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sum_{l=1,\ldots,m} t_{ll} d^2_l(x_{i_l}, x_{j_l})$$

where t_{ll} are the diagonal elements of $\boldsymbol{A}$ or $\boldsymbol{W}^T\boldsymbol{W}$. This restriction can be seen as a simple form of regularization since it reduces the effective number of parameters from $O(m^2)$ to $O(m)$. It should be noted that the regularization technique based on diagonal matrices, although resulting in faster algorithms, is also less expressive since it does not account for interactions between different decompositions. When full matrices are used these interactions are weighted by off diagonal elements of matrix $\boldsymbol{A}$ or $\boldsymbol{W}^T\boldsymbol{W}$, i.e.

$$d^2_{tuple,\boldsymbol{Z}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sum_{l,k=1,\ldots,m} t_{lk} d_l(x_{i_l}, x_{j_l}) d_k(x_{i_k}, x_{j_k}).$$

5.1.6 Solving Optimization Problems

As already mentioned in the Xing and MCML methods the objective functions are convex (Avriel, 2003) and thus the optimization problems are well defined in the sense that there exists a single global optimum. Different initialization and optimization techniques may affect the efficiency of the algorithm but the final solution itself is unique. In order to ensure that $\boldsymbol{A} \succeq 0$ we use the iterative projection approach as proposed in (Xing et al., 2003). First, we calculate the eigen-decomposition of $\boldsymbol{A} = \sum_k \lambda_k \boldsymbol{u}_k \boldsymbol{u}_k^T$, where λ_k for $k = 1, \ldots, m$ are $\boldsymbol{A}$'s eigenvalues, and $\boldsymbol{u}_k$ the corresponding eigenvectors. Subsequently, we set $\boldsymbol{A} = \sum_k \max(\lambda_k, 0)\boldsymbol{u}_k\boldsymbol{u}_k^T$. In the NCA method the optimization is done with respect to matrix $\boldsymbol{W}$ of Equation 5.2 which makes the optimization problem easier to solve, since it is unconstrained. However, the objective function is no longer convex and is thus susceptible to local minima, and sensitive to initial conditions and choice of optimization method.

In order to solve the optimization problems we exploit the conjugate gradient method (Avriel, 2003) where backtracking line search is used to optimize the step-size parameter. For full matrices this method has a complexity of $O(m^2)$ since it requires computing a gradient in the form of an $m \times m$ matrix. In the

case of diagonal matrices the above complexity is reduced to $O(m)$. This method has an advantage over the well-known Newton method since the latter has a complexity of $O(m^3)$ (for diagonal matrices) because it requires the inversion of the $m \times m$ Hessian matrix; for full matrices this complexity is $O(m^6)$.

5.1.7 Computational Complexity

The computational complexity of the adaptive methods depends on the cost of computing the objective function, the gradient, and the number of iterations needed to converge. The cost of computing the objective functions and the gradients depends on the size of the sets $\mathcal{S}$ and $\mathcal{D}$, and the parameter m (i.e. the lengths of the input tuples). In the supervised setting the size of both $\mathcal{S}$ and $\mathcal{D}$ scales in general as $O(n^2)$ where n is the size of the training set. Moreover, as mentioned in the previous section, for full (diagonal) matrices the complexity of computing the objective and gradient scales as $O(m^2)$ ($O(m)$).

In the case of Xing's method the complexity of computing the objective function and gradient is $O(n^2m^2)$ and $O(n^2m)$ depending on whether full or diagonal matrices are used. In MCML and NCA the additional complexity comes from the normalization of $p_{\boldsymbol{A}}(j|i)$ (or $p_{\boldsymbol{W}}(j|i)$) which requires one extra iteration through the training set. As a result, the computational complexity of MCML and NCA is $O(n^3m^2)$ (or $O(n^3m)$ for diagonal matrices). On the other hand, since MCML and NCA do not take implicitly into account the pairs in $\mathcal{D}$, their complexity is reduced and depends only on $|\mathcal{S}|$. The complexities of the proposed algorithms are also influenced by the number of iterations i needed to converge, which in the applications we examined, depending on the method and the choice of the λ parameter, varied between 2 and 60.

5.2 Application Scenarios

The above adaptive framework can be used in various learning tasks involving composite objects. In this section we list the application scenarios that we will examine in the remainder of this chapter (Sections 5.3 and 5.4). More precisely, we show that it is possible to: (i) learn an optimal combination of a set of predefined complex distance measures and (ii) combine a number of representations of composite objects.

In the task of distance combination we assume that an object is represented as $\boldsymbol{x}_i = (x_{i_1}, \ldots, x_{i_m})^T \in \boldsymbol{\mathcal{X}} = \mathcal{X}_1 \times \ldots \times \mathcal{X}_m$ where $x_{i_k} = x_{i_l}$, $\forall k, l = 1, \ldots, m$. Over each of $\mathcal{X}_l$, $l = 1, \ldots, m$ a *different* complex distance measure $d_l : \mathcal{X}_l \times \mathcal{X}_l \rightarrow \mathbb{R}_0^+$ is applied. The goal here is to learn an optimal combination of all d_l.

Our framework can be also used in applications where the "correct" representation of composite objects is not given and it is difficult to specify it a priori. More precisely, we assume that a composite object is represented as $\boldsymbol{x}_i = (x_{i_1}, \ldots, x_{i_m})^T \in \boldsymbol{\mathcal{X}} = \mathcal{X}_1 \times \ldots \times \mathcal{X}_m$ where $x_{i_k} \neq x_{i_l}$, $\forall k, l = 1, \ldots, m$. Over each of $\mathcal{X}_l$, $l = 1, \ldots, m$ a *fixed* distance measure $d : \mathcal{X}_l \times \mathcal{X}_l \rightarrow \mathbb{R}_0^+$ is

applied. The goal here is to learn, according to d, a combination of the different decompositions $\mathcal{X}_l$.

In this study we focus on graph problems and we learn the appropriate combinations of different graph decompositions; there exist several possible decompositions of graphs into specific subparts and it is in general difficult to specify the appropriate decomposition a priori. We decompose graphs into sets of walks and trees of various lengths and heights, respectively (see Section 2.3 for details). Both types of decompositions are obtained from a depth first exploration emanating from each node in a graph and yielding all the walks of length l and all the trees of height h. Additionally, we require that repetitions of the same two-nodes-cycles are not allowed so that we avoid the problem known as *tottering* (Mahé et al., 2004). In particular, for $l = 1$ (and $h = 1$) a graph is represented as a set of vertices. For $l = 2$ a graph is decomposed into a set of two-element tuples where the first element is a vertex, and the second element is one of its adjacent edges. Similarly, for $h = 2$ the corresponding decomposition is into trees with vertices as roots connected to all their adjacent edges.

The adaptive distance measures can be easily turned into kernels, which could be used in large-margin classification, possibly increasing the performance of the adaptive techniques. For this purpose we can use the kernels in the proximity space $k_{P,d}$ defined in Section 4.2.4 which were shown to be quite effective in various domains[2]. The adaptive kernel is induced by the learned combination of representations or distances on a given representation set (usually the full training set). In particular, the distance measures resulting from combinations of various graph decompositions into substructures of specific types can be exploited to construct positive semi-definite (PSD) kernels in proximity space resulting in a flexible and powerful class of graph kernels. In the next section we will show how this kernel relates with the other relevant kernels defined on graphs; the general discussion on the the existing kernels over graphs was already given in Section 4.5. We also discuss the computational complexity of the proposed kernels.

5.2.1 Graph Kernels

As already mentioned, in this work we will experiment with, and learn combinations of, two types of decompositions: walks and trees of various lengths and heights. The former were widely used in the context of graph kernels and shown to be very effective in chemical domains, achieving state-of-the-art results (Gärtner et al., 2003; Kashima et al., 2003; Mahé et al., 2004; Borgwardt et al., 2005; Ralaivola et al., 2005; Vishwanathan et al., 2007). Decomposition kernels based on trees were first proposed by Ramon and Gärtner (2003); however, the authors did not perform experimental evaluation. Graph kernels based on trees were empirically examined in (Fröhlich et al., 2005).

[2] We could also use the distance substitution kernel from Section 4.2.4; however, the experimental results from Section 4.4.4 indicated that kernels in proximity space are a better alternative.

The fact that we combine different graph representations and use flexible set distance measures over the corresponding decompositions, results in a flexible and powerful class of PSD kernels over graphs. In particular, the resulting kernels address some of the limitations of the existing decomposition-based graph kernels.

To the best of our knowledge all existing graph kernels are currently limited to a single type of decomposition which is then, most often, used in the context of the cross product kernel. However, it is difficult in general to specify in advance the appropriate type of substructures for a given problem. Although it is in principle possible to simultaneously exploit kernels defined over different representations, this is usually not done because there is a trade-off between expressivity, achieved by enlarging the kernel-induced feature space, and the increased noise to signal ratio (introduced by irrelevant features). It should be noted that we could directly combine graph kernels using the methods proposed e.g. in (Lanckriet et al., 2004; Ong et al., 2005; Bousquet and Herrmann, 2003). Nevertheless, the problem with learning kernel combinations is that the combined elements should, obviously, be valid kernels on the different types of decompositions. However, as we saw previously this type of kernels are based on the cross product kernel that requires the complete matching of the individual components, raising the problems that were described before. Finally, the existing methods for kernel combination work only in transductive setting, i.e. they complete the labeling of a partially labeled dataset, which limits the application area of such methods (see Section 5.5).

The use of flexible set distance measures results in kernels which do not necessarily require that every subpart of one decomposition is matched against every subpart of the other, or that all elements of the two sets should participate in the computation of the final distance. This is in contrast with most of the graph kernels (or kernels on composite structures) which are based on averaging; for such kernels the generalization of a large margin classifier might be harmed since due to the combinatorial growth of the number of distinct subgraphs most of the features in the feature space will be poorly correlated with the target variable (see the discussion on the set and graph kernels in Section 4.5).

Walks and trees can be easily compared by a graded similarity (e.g. the standard Euclidean distance measure for walks). Nevertheless, most of the kernels on graphs use the δ kernel (Definition 4.4) on subparts, i.e. $k(x,y) = 1 \iff x = y, k(x,y) = 0$ otherwise. As a result, the ability to find partial similarities is lost and the expressivity of these kernels is reduced (Borgwardt et al., 2005). A graded similarity on walks was considered in the graph kernel presented in (Borgwardt et al., 2005); however, it suffers from high computational complexity since it requires taking powers of the adjacency matrix of the direct product graph, leading to huge runtime and memory requirements (Borgwardt and Kriegel, 2005). Graded similarities on trees were also used in (Fröhlich et al., 2005).

The main disadvantage of the graphs kernels defined in the proximity space is that the resulting models are not sparse, i.e. predictions for new inputs depend on all the examples from the representation set. However, as already noted in

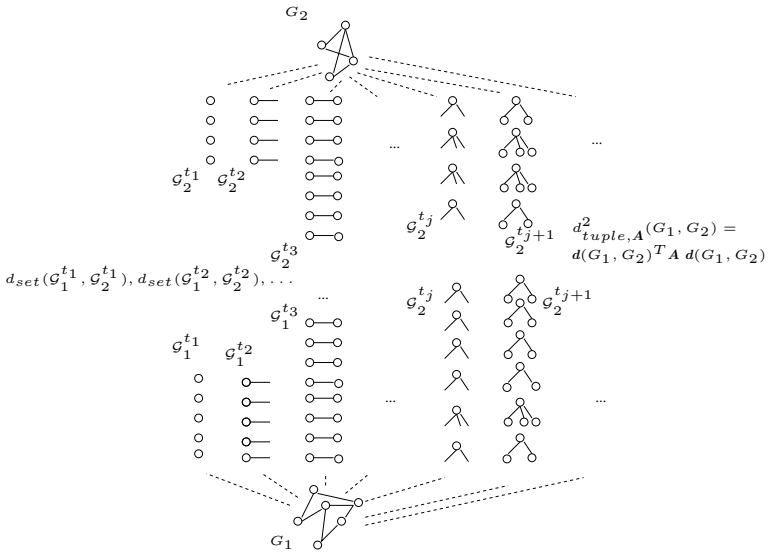


Figure 5.2: A schematic description of the basic building components of the proposed graph kernel. G_1, G_2 are any two graphs, $\mathcal{G}_i^{t_k}$ is the decomposition of the G_i graph to substructures of type t_k, $d_{set}(\mathcal{G}_1^{t_k}, \mathcal{G}_2^{t_k})$ is a set distance between the sets of decompositions of G_1 and G_2 to substructures of type t_k, $d^2_{tuple,\boldsymbol{A}}(G_1, G_2)$ is the final learned weighted distance that combines different types of decompositions.

Section 4.2.4, one way to overcome this limitation is to use a prototype selection method which makes the resulting models sparser. In the experiments reported in Section 5.4.2 we will see that it is indeed possible to reduce the size of the representation set without harming the performance of a large-margin algorithm (SVM).

The idea of our adaptive graph kernels in proximity space $k_{P,d}$ can be illustrated with a simple example given in Figure 5.2. Consider two graphs G_1 and G_2; each graph, G_i, is decomposed into a number of sets of walks of varying lengths, noted in the figure as $\mathcal{G}_i^{t_1}, \mathcal{G}_i^{t_2}, \mathcal{G}_i^{t_3}, \ldots$, and a number of sets of trees of varying height, noted as $\mathcal{G}_i^{t_j}, \mathcal{G}_i^{t_{j+1}}, \ldots$, giving rise to the final representation of each graph as $\boldsymbol{\mathcal{G}}_i = (\mathcal{G}_i^{t_1}, \ldots, \mathcal{G}_i^{t_l})^T$ where l is the total number of different decompositions. In order to compute the final distance between the two graphs we have first to compute their distance with respect to each of the components of $\boldsymbol{\mathcal{G}}_i$ using a set distance, the result is a vector of l set distances, $\boldsymbol{d}(G_1, G_2) = (d_{set}(\mathcal{G}_1^{t_1}, \mathcal{G}_2^{t_1}), d_{set}(\mathcal{G}_1^{t_2}, \mathcal{G}_2^{t_2}), \ldots, d_{set}(\mathcal{G}_1^{t_l}, \mathcal{G}_2^{t_l}))^T$. The next step is to learn the final distance $d^2_{tuple,\boldsymbol{A}}(G_1, G_2)$ as a weighted combination of the component-wise set distances, i.e., $d^2_{tuple,\boldsymbol{A}}(G_1, G_2) = \boldsymbol{d}(G_1, G_2)^T \boldsymbol{A}\, \boldsymbol{d}(G_1, G_2)$. Once the distance $d^2_{tuple,\boldsymbol{A}}$ is learned we can induce the corresponding proximity space, using $d^2_{tuple,\boldsymbol{A}}$ and the representation set, the latter typically being the full training set. It is on that space that the final kernel on graphs is defined.

Complexity Analysis

In this section we provide the analysis of the computational complexity of our adaptive graph kernels. The complexity of general graph kernels based on decompositions depends largely on the cost of constructing and matching subgraphs as well as computing the cross product (CP) kernel. We replace the computation of the CP kernel by a given set distance measure. An additional factor that influences complexity of our kernel is the cost of learning the weights; however, this occurs only during the training phase. We will give the complexity of the adaptive matching based kernels where decompositions are into walks and trees. For decompositions into other types of subgraphs different complexity might be obtained. Finally, we report the complexity assuming that the full matrices are used.

The complexity of computing a set distance measure varies from $O(n^2)$ (for d_{SL}, d_{CL}, d_{AL}, d_{SMD}, d_H and d_{RIBL}) to $O(n^3)$ (for d_T, d_S, d_{FS}, d_L and d_M) where n is the cardinality of the sets (see Section 3.2.3). The complexity of extracting all walks of length up to l_{max} (trees of height up to h_{max}) from a graph G is at most $O(|G|\alpha^{l_{max}})$ (or $O(|G|\alpha^{h_{max}})$) where α is the branching factor which can be upper bounded by a small constant (usually 4). The cost of matching walks of length l using the normalized Euclidean distance (Equation 3.11) is $O(l)$ whereas the complexity of computing distances from Equation 3.22 between trees of height h is $O(\alpha^{3(h-1)})$; here we assume that labels of vertices and edges can be matched in $O(1)$ time. Given the above observations the overall complexity of computing the adaptive matching kernel between two graphs (if the weights of different decompositions are known) is at most $O\{|S|m^2|G|^3(\alpha^{3l_{max}} + \alpha^{3h_{max}})\}$ where $|S|$ is the cardinality of the representation set and $m = l_{max} + h_{max}$ is the number of decompositions.

Depending on the actual application and the characteristics of decompositions different factors will dominate the above complexity. In particular, the size of the representation set can be reduced by using prototype selection methods as described in Section 4.2.4. Moreover, the average branching factor in the datasets we experimented with was close to two (the last column in Table A.3)[3]. As a result, in our experimental setup (Section 3.5), the above complexity reduces to $O(|G|^3)$; or $O(|G|^2)$ if one of the d_{SL}, d_{CL}, d_{AL}, d_{SMD}, D_H, d_{RIBL}, set distance measures is used.

The complexity of learning the weights for MCML and NCA is at most $O(im^2(n^2|G|^3 + n^3))$ where i is the number of iterations needed for the above function to converge (in our case, depending on the set distance measure and the application, i varied between 5 and 40). In the Xing method this complexity is at most $O(im^2n^2|G|^3)$. In Table 5.1 we compare the complexity of our kernel to known decomposition kernels based on walks and trees (only testing phase). We assume that we are dealing with two graphs with n nodes and k bonds.

[3]In organic chemistry in general this average is slightly above two (Ralaivola et al., 2005).

Graph Kernel	Complexity
Exponential and Geometric kernels (Gärtner et al., 2003; Vishwanathan et al., 2007)	$O(n^3)$
Random walks kernels (Kashima et al., 2003; Vishwanathan et al., 2007)	$O(n^3)$
Kernels from (Ralaivola et al., 2005)	$O(nk)$
Assignments kernel (Fröhlich et al., 2005)	$O(n^3)$
Graph kernel from this section	$O(\|S\|m^2n^2)$ or $O(\|S\|m^2n^3)$ plus complexity of learning the weights

Table 5.1: Worst-case complexity of related kernels based on walks and trees.

5.3 Experiments on Distance Measure Learning

In the first part of the experiments, we will perform the evaluation of the methods from Section 5.1 on the task of composite distance combination. We focus on problems where training instances are represented as sets of objects. More precisely, we used a representation where an instance $\boldsymbol{x}_i$ is given as tuple of 11 sets, i.e. $\boldsymbol{x}_i = (x_{i_1}, \ldots, x_{i_{11}})^T$ where $\forall k, l \in \{1, \ldots, 11\}$, $x_{i_k} = x_{i_l}$. On each of the decomposition we apply one of the 11 set distance measures described in Section 3.2.3, i.e. d_{SL}, d_{CL}, d_{AL}, d_{SMD}, d_H, d_{RIBL}, $d_{T,\theta=0.01}$, d_S, d_{FS}, d_L and d_M. The different complex distance combination methods are compared in the context of the kNN algorithm. The goal is to examine whether we can increase the predictive performance of kNN by combining different set distance measures. We denote kNN in which the combined distances are used by kNN_{DC}.

5.3.1 Experimental Setup

We compared two regularized instantiations of the Xing, MCML and NCA methods, over full and diagonal matrices which we will denote respectively $\text{METHOD}_{\text{full}}$ and $\text{METHOD}_{\text{diag}}$, where METHOD is Xing, MCML or NCA. We used two methods as a baseline for comparison. The first one, denoted as kNN_{Best}, is obtained by simply selecting the set distance which gives the best 10-fold CV performance on the full dataset. It should be noted that this performance estimate is optimistically biased. The second baseline method, denoted as kNN_{CV}, is based on an inner cross-validation loop to select the appropriate set distance. More precisely, on each training set an inner 10-fold stratified cross-validation is performed for each set distance in order to select the one with the highest accuracy.

The results are reported for all the datasets, except for the protein fingerprints. For graph datasets the results are only reported when the first representation of the learning instances is used, i.e. graph are represented as sets of trees

	$\text{Xing}_{\text{full}}$	$\text{Xing}_{\text{diag}}$	$\text{MCML}_{\text{full}}$	$\text{MCML}_{\text{diag}}$	NCA_{full}	NCA_{diag}	kNN_{Best}	kNN_{CV}
diterpenes	96.01(-)(-)	51.63(-)(-)	97.54(=)(=)	98.34(+)(+)	98.34(+)(+)	97.74(=)(=)	97.41	97.41
musk (ver. 1)	63.04(-)(-)	75.00(=)(=)	84.78(=)(=)	88.04(=)(+)	85.87(=)(=)	89.13(=)(+)	84.78	80.43
musk (ver. 2)	59.80(-)(=)	57.84(-)(=)	60.78(-)(=)	74.51(=)(=)	79.41(=)(=)	88.24(+)(+)	77.45	70.59
duke	65.85(=)(=)	65.85(=)(=)	70.73(=)(=)	65.85(=)(=)	60.98(=)(=)	63.41(=)(=)	78.05	70.73
muta (ver. 1)	67.02(-)(-)	76.06(-)(-)	82.98(=)(=)	82.45(=)(=)	85.64(=)(=)	86.17(=)(=)	87.23	87.23
FM (ver. 1)	52.15(-)(=)	56.73(=)(=)	58.17(=)(=)	63.90(=)(+)	60.17(=)(=)	62.18(=)(=)	63.32	57.31
FR (ver. 1)	63.52(=)(=)	61.82(=)(=)	62.39(=)(=)	62.11(=)(=)	64.10(=)(=)	64.67(=)(=)	65.53	66.10
MM (ver. 1)	59.52(=)(=)	59.82(=)(=)	61.90(=)(=)	60.42(=)(=)	62.80(=)(=)	59.82(=)(=)	63.39	60.42
MR (ver. 1)	54.94(=)(=)	56.69(=)(=)	56.69(=)(=)	54.36(-)(-)	56.40(=)(=)	54.94(=)(=)	61.63	61.92

Table 5.2: Accuracy and significance test results of the kNN_{DC} algorithm. The + sign stands for a significant win of the first algorithm in the pair, - for a significant loss and = for no significant difference. The signs in the first parenthesis correspond to the comparison of kNN_{DC} vs. kNN where the best distance is used (the kNN_{Best} column) and the second to the comparison of kNN_{DC} vs. kNN with cross-validation (the kNN_{CV} column).

of height 3.

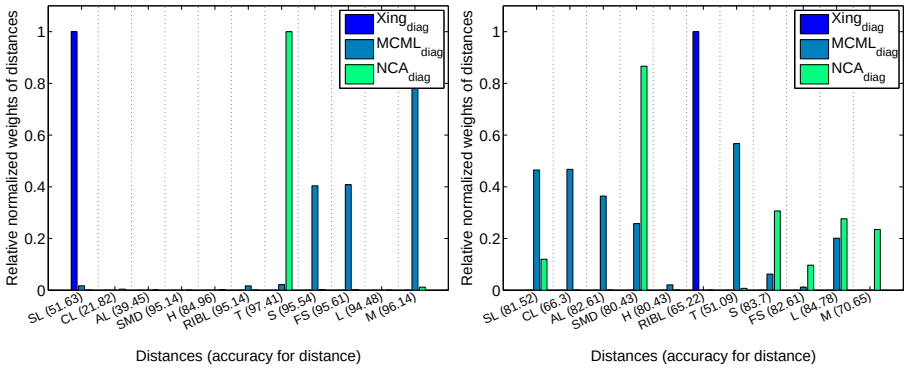
In all the experiments the number of nearest neighbors in kNN was optimized in an inner 10-fold cross validation loop over $k = \{1, 3, 9\}$. Moreover, the the regularization parameter λ in $\text{METHOD}_{\text{full}}$ was cross-validated over $\lambda = \{0, 0.1, 1, 10\}$; in $\text{METHOD}_{\text{diag}}$ the λ parameter was set to 0. We estimate accuracy using stratified ten-fold cross-validation and control for the statistical significance of observed differences using McNemar's test (McNemar, 1947), with significance level of 0.05.

5.3.2 Results and Analysis

The results are presented in Table 5.2. From these results we can see that NCA has an advantage over the considered baseline methods. It is never significantly worse and sometimes it is significantly better than both kNN_{Best} and kNN_{CV}. The MCML method is also sometimes significantly better than the baseline methods; however, in comparison with NCA it does not fare so well, being sometimes significantly worse than both baseline methods. Finally, Xing's method is never significantly better (and in many cases it is significantly worse) than both kNN_{Best} and kNN_{CV}. We note that the performances achieved by $\text{MCML}_{\text{full}}$ and NCA_{diag} in the diterpenes dataset (98.34 %) are the best reported so far in the literature.

The poor performance of Xing's method might be a result of the fact that the objective function in this method is not suitable for data exhibiting a multimodal distribution. On the other hand, the good performance of NCA could be explained by the fact that it makes no assumptions about the shape of the class conditional distributions or the boundaries between different classes. Another possible explanation for the poor performance of Xing's method, especially in musk (ver. 1), musk (ver. 2) and mutagenesis datasets, is that in these datasets the negative class contains *any* examples which do not have the property encoded by the positive class. As already mentioned, in such cases the cost function implemented by Xing's method is severely penalized. The above limitations also hold for MCML; however, as we can see from Table 5.2 MCML has an advantage over the Xing method. This might be a result of the fact that in MCML more importance is put on the pairs of instances form different classes (in Xing's method this contribution is logarithmically down-weighted) which could factor out the effect of the above mentioned multi-modalities.

The instantiations of the Xing, MCML and NCA methods for which optimization is performed over diagonal matrices tend to have an advantage over the corresponding soft-regularized full matrix instantiations, mainly for the datasets with small number of instances (i.e. both versions of the musk datasets and duke). Indeed, in these datasets the accuracy of $\text{METHOD}_{\text{diag}}$ was higher that $\text{METHOD}_{\text{full}}$ in 6 out of 9 cases; however, only in one case was this difference statistically significant (MCML in version 2 of musk); in all other cases the differences were not statistically meaningful. For these datasets, when we work with the full matrices which have $O(m^2)$ parameters to estimate, the small number of training instances leads to under-determined problems. In these cases, in

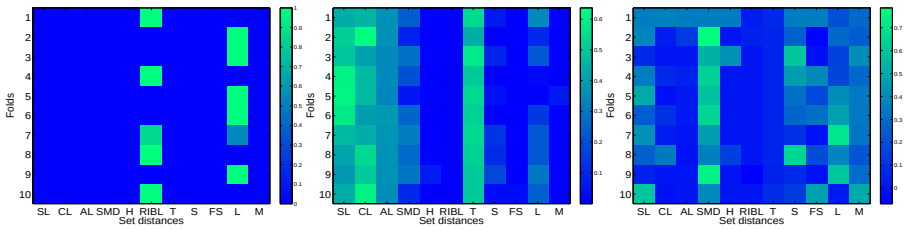


(a) diterpenes (b) musk (ver. 1)

Figure 5.3: Relative importance of the different set distance measures, computed by $\text{METHOD}_{\text{diag}}$ ($\lambda = 0$), for the diterpenes and musk (ver. 1) datasets. The weights are learned on the full training set. For each set distance measure we also provide in parenthesis the corresponding performance of kNN.

comparison with the soft-regularization, the reduction of the number of free parameters to estimate to $O(m)$ tends to be a better means of protection against a possible overfit. For other datasets no clear trends can be observed.

The results of the optimization process, both for full and diagonal matrices, can be visualized to provide insight into the relative importance of the distance measures (and their combinations for $\text{METHOD}_{\text{full}}$), as these are determined by each method. An example of such a visualization for diterpenes and version 1 of musk is given in Figure 5.3, where optimization is performed over diagonal matrices and where the weights are learned on the full training set; the visualization for other datasets is presented in Tables B.3 and B.4 in Appendix B.4. The bars correspond to distance measures for which the corresponding accuracies of kNN, estimated by 10-fold cross-validation, are given in parenthesis. The y-axis represents the weights, normalized by the Frobenius norm of $\boldsymbol{A}$, computed by the three methods. The visualization results are somehow in agreement with results from Table 5.2. For example, in diterpenes $\text{MCML}_{\text{diag}}$ (98.34 % of accuracy) assigns high weights to distance measures which individually exhibit good performance (d_S, d_{FS} and d_M) and neglects the ones with lower performance (d_{SL}, d_{CL}, d_{AL}, d_{SMD}, d_H, d_L). For this combination method the performance of each of the constituent distances is lower than the performance of d_T (i.e. the best "single" distance measure), nevertheless, by combining the constituent distance measures we are able to obtain the best performance in diterpenes reported in the literature. The lower accuracy of $\text{Xing}_{\text{diag}}$ (96.01 %) and NCA_{diag} (97.74 %) is a result of assigning very high weights to d_M and d_T, respectively. In the version 1 of musk $\text{Xing}_{\text{diag}}$ (with performance of 75.00 %) assigns high weight to d_{RIBL} (65.22 %) and weights which are close to zero for other distances. This indicates that other configurations of weights are used within the different



(a) $\text{Xing}_{\text{diag}}$ (b) $\text{MCML}_{\text{diag}}$ (c) NCA_{diag}

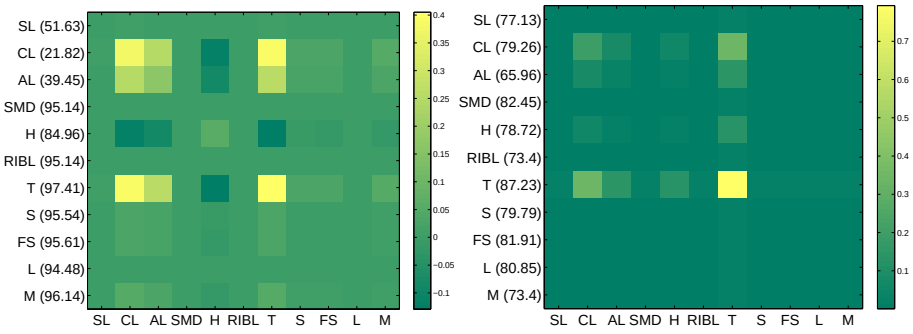
Figure 5.4: Stability of the distance combination techniques in musk (ver. 1). Weights in each of the 10 folds (i.e. rows) are normalized by a Frobenius norm of $\boldsymbol{A}$ ($\boldsymbol{W}$).

folds of internal cross-validation (as noted above the weights are obtained by application of $\text{METHOD}_{\text{diag}}$ of the full training set). This might be a result of the fact that musk (ver. 1) has a small number of instances, and hence the instantiations of $\text{METHOD}_{\text{diag}}$ are not stable with respect to the perturbations in training data.

To further investigate the issue of stability of the solutions obtained by $\text{METHOD}_{\text{diag}}$ we plot in Figure 5.4 the relative importance of the different set distance measures (for version 1 of musk), assigned by different methods, across the different folds of the 10-fold cross-validation. For example, in $\text{Xing}_{\text{diag}}$ only d_{RIBL} and d_L contribute to the final solution; however, the weights assigned to these distance measures are not stable, i.e. the weights are either 0 or 1 (with the exception of fold 7 where both components have weights close to 0.5). On the other hand, in $\text{MCML}_{\text{diag}}$ and NCA_{diag} the solutions are more stable among different folds and are in agreement with the results presented in Figure 5.3 (b). In Figures B.5 and B.6 in Appendix B.4 we present the same visualization for the remianing datasets; in these datasets the solutions of the distance combinations are in general stable across different cross-validation folds.

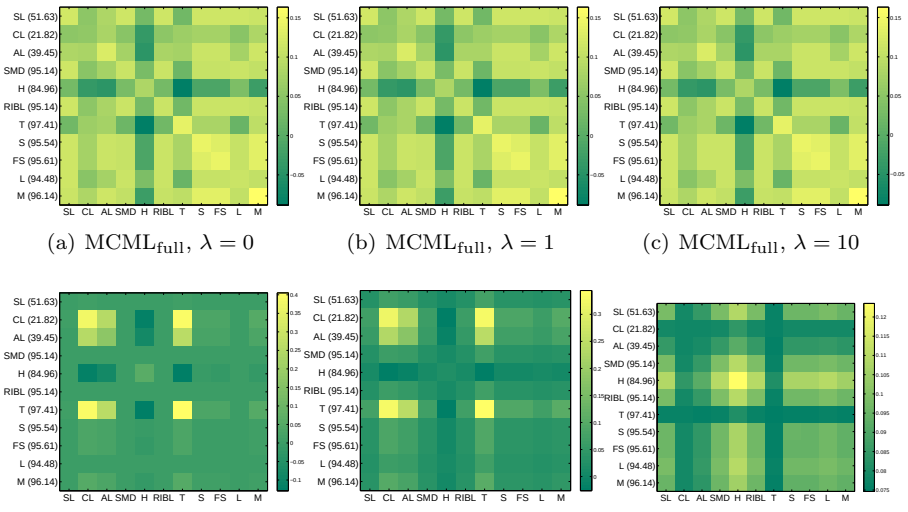
The full matrix obtained for the diterpenes and mutagenesis datasets using NCA_{full} ($\lambda = 0$) is displayed graphically in Figure 5.5. As with the diagonal case the weights are obtained on the full training dataset and are normalized by the Frobenius norm of $\boldsymbol{A}$. For example in diterpenes, presented in plot (a), the NCA_{full} method assigns the highest weights to d_T, d_{SL} and d_{AL}, as well as the combinations of these distances. The two latter distances have a very poor performance when used alone (21.81 % of accuracy for d_{SL} and 39.45 % for d_{AL}); however, when combined with d_T (97.41 %), the state-of-the-art accuracy of 98.67 % is achieved. On the other hand, in the version 1 of musk d_{CL} (79.26 %) and d_T (87.23 %) distances (and their combinations) are assigned high weights; however, the performance of the overall combination is 85.87 % and is lower than of d_T.

It is also interesting to analyze the impact of the λ parameter on the resulting weights. In Figure 5.6 we present the relative importance of the different set



(a) diterpenes (b) mutagenesis (ver. 1)

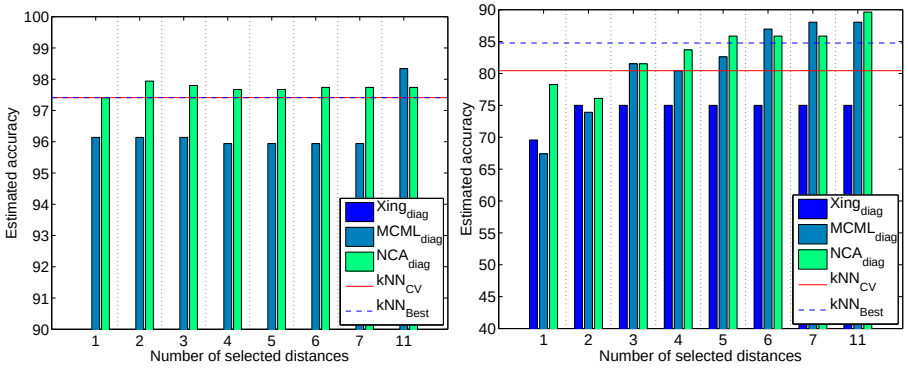
Figure 5.5: Relative importance of the different set distance measures, computed by NCA_{full} ($\lambda = 0$), for the diterpenes and mutagenesis datasets. The weights are computed on the full training set. For each set distance measure we also give the corresponding performance of kNN in parenthesis.



(a) $\text{MCML}_{\text{full}}$, $\lambda = 0$ (b) $\text{MCML}_{\text{full}}$, $\lambda = 1$ (c) $\text{MCML}_{\text{full}}$, $\lambda = 10$

(d) NCA_{full}, $\lambda = 0$ (e) NCA_{full}, $\lambda = 1$ (f) NCA_{full}, $\lambda = 10$

Figure 5.6: Relative importance of the different set distance measures as these are computed by $\text{MCML}_{\text{full}}$ and NCA_{full}, for different values of λ. The weights are computed in the diterpenes dataset and on the full training set.



(a) diterpenes (b) musk (ver. 1)

Figure 5.7: The estimated accuracy for $\text{METHOD}_{\text{diag}}$ (using kNN) where $l = 1, \ldots, 6, 11$ top ranked distance measures (according to the assigned coefficients) are combined for computing the actual distance. Additionally, the performance of kNN_{Best} and kNN_{CV} is presented.

distance measures computed by the $\text{MCML}_{\text{full}}$ and NCA_{full} methods, in the diterpenes dataset, for different values of $\lambda = \{0, 1, 10\}$. We can see that for $\text{MCML}_{\text{full}}$ the λ parameter does not seem to have an impact on the obtained weights, i.e. the relative importance of the different set distance measures is preserved across the considered values of λ. This is in contrast with NCA_{full} where the weights obtained for $\lambda = \{0, 1\}$ are different than when λ was fixed to 10, and in particular, the set distances which were assigned high weights for $\lambda = \{0, 1\}$, are neglected for $\lambda = 10$, and vice-versa. Visualization for other datasets is presented in Tables B.7 to B.14 in Appendix B.4.

Using the method instantiations based on diagonal matrices, it is easy to reduce the size of the problem by selecting only the l top-ranked distance measures (according to the assigned coefficients) which can be used to obtain the distance combination. We examined the performance of kNN_{DC} for l ranging from 1 to 7 (the total number of distance measures is 11). The visualization for the diterpenes and musk (ver. 1) datasets is given in Figure 5.7 (the visualization for other datasets is presented in Figures B.15 and B.16 in Appendix B.4). For comparison we also provide the results when all 11 distance measures are combined. It should be noted that the objective functions are optimized with respect to all the distance functions considered. Indeed, in diterpenes it is clear that the best performance of kNN_{DC} is achieved when all the distance measures are used (i.e. $l = 11$). However, the performance when we select the 4 top distance measures is very similar to the performance when all distances are used.

There is an interesting synergy that arises from the nature of the set distances and the fact that we learn combinations of them. Remember that all the set distances we consider (with the exception of d_T) are defined on the basis of a

given mapping, F, of elements of one set to elements of the other set. One can view the task of learning a set distance measure as learning the mapping F, i.e. which pairs of elements of the two sets should participate in the mapping and how important they are. Under this view learning a set distance measure would correspond to learning the weights ω_{ab} (associated with a pair of elements $(a, b) \in A \times B$), in the function

$$d_{set}(A, B) = \sum_{(a,b)\in A\times B} \omega_{ab} d(a, b)$$

where $\omega_{ab} \in \{0, 1\}$ (or more general $\omega_{ab} \in [0, 1]$). It can be seen that the combination of distances from Equation 3.14 provides an intermediate solution to this problem. By restricting the matrix from Equation 3.14 to be diagonal, we obtain a set distance measure of the form

$$d_{set}(A, B) = \sum_{(a,b)\in F} w_{ab} d(a, b)$$

where $F = \bigcup_{i=1}^{m} F_i$, F_i is the mapping corresponding to set distance measure d_i and w_{ab} is computed by adding the coefficients assigned to set distance measures in which (a, b) appears. The final mapping, F, is more expressive than any of its constituents, F_i, and cannot be obtained by considering any of the F_is individually. This flexible way of mapping of elements of the two sets might explain the good performance of the combinations of set distances.

5.4 Experiments on Representation Learning

In this section we apply the proposed adaptive framework on the task of learning the appropriate combinations of different graph decompositions for various graph classification problems (i.e. mutagenesis and four versions of carcinogenicity); there exist several possible decompositions of graphs into specific subparts and it is in general difficult to specify the appropriate decomposition a priori.

We decompose graphs into sets of walks and trees of various lengths and heights, respectively. We set the maximum length of walks to 11 and the maximum depth of trees to 4. More formally, a graph $\boldsymbol{x}_i$ is presented as tuple $(x_{i_1}, \ldots, x_{i_{11}}, x_{i_{12}}, \ldots, x_{i_{15}})^T$ where x_{i_l}, for $l = 1, \ldots, 11$, represents a *set* of walks of length l and x_{i_h}, for $h = 12, \ldots, 15$, corresponds to a *set* of (unordered) trees of height $h - 11$. The performances of the different graph decompositions will be examined in the context of graph kernels as presented in Section 5.2.1. These kernels are plugged into an SVM algorithm and it is the performance of the SVM with the corresponding kernel that we estimate.

In the experiments we want to examine a number of issues related to: the individual performances of the different decompositions and how these relate to the performance of the combined decompositions; the performance of different ways of combining the constituents of a given decomposition, i.e. the use of different set distances, and how they relate to more standard ways of combining

these constituents such as averaging; the importance of size of the different substructures and the performance of various weighting schemes; the performance of the feature (prototype) selection algorithms; and finally, the performance of our graphs kernels in comparison to other graph kernels.

More precisely, first, for different set distance measures we will explore the effect of the length of walks (and the height of trees) on the performance of $k_{Pd_{set}}$, i.e. we fix the walk length to a specific l for $l = 1, \ldots, 11$ (and the height of a tree is fixed to a specific h, $h = 2, \ldots, 5$) and we use only the corresponding decomposition to construct the final kernel; we do *not* combine decompositions. We will compare the performance of the above simplified kernels with $k_{Pd_{tuple,\boldsymbol{A}}}$ (for the MCML method) and $k_{Pd_{tuple,\boldsymbol{W}}}$ (for NCA) where we combine over all the different $l + h$ decompositions for $l = 1, \ldots, 11$ and $h = 2, \ldots, 5$. For the kernel k on the proximity space we will be using the linear kernel in order to make a fair comparison between the algorithms and to avoid the situation where an implicit mapping given by a nonlinear kernel will influence the results.

Second, we will examine how the performance of the kernel $k_{Pd_{tuple,\boldsymbol{A}}}$ ($k_{Pd_{tuple,\boldsymbol{W}}}$) compares with the following two set kernels based on averaging: (i) the direct sum kernel based on the cross product kernels applied on the different decompositions with the linear kernel (in case of walks) and the tree kernel (for trees) as kernels on the sets' elements, and (ii) the linear kernel in the proximity space induced by $d_{tuple,\boldsymbol{A}}$ ($d_{tuple,\boldsymbol{W}}$), also over the combination of the different $l + h$ decompositions, where d_{AL} set distance measure is applied over the decompositions. Kernels matching all subgraphs are a standard way of tackling classification problems where instances are represented as graphs.

Third, we will analyze the importance of size of different sub-structures, determined on the basis of weights assigned by our adaptive methods. We will focus on decompositions into walks and analyze the impact of walks' lengths. Additionally, we will examine how the performance of our methods applied on decompositions into walks of different lengths compares with two baseline weighting schemes which are widely used in practice. These weighting schemes are the isotropic scheme which assumes that all decompositions are equally important, and down-weighted scheme where decompositions into walks of length l are assigned weights of $1/2^l$.

Next, we are going to examine the influence of a prototype selection algorithm (or equivalently feature selection algorithm applied in the proximity space) to the performance of the kernels in proximity spaces. More precisely, we will use the CFS algorithm (Hall, 1998). The goal of this experiment is to check whether we can obtain sparse solutions such that the complexity is lower for both representation and computation of the kernels.

Finally, we will compare our graph kernels with other state-of-the-art graph kernels presented in the literature.

5.4.1 Experimental Setup

In order to combine different graph representations we need to define distance measures over corresponding sets of decompositions. For this purpose we ex-

ploit the set distance measures from Section 3.2.3 which allow for increased flexibility on the type of matchings of elements between the two sets of decompositions. More precisely, we experimented with the d_{SL}, d_{CL}, d_{AL}, d_{SMD}, d_H, d_{RIBL}, $d_{T,\theta=0.01}$, d_S, d_{FS}, d_L and d_M set distance measures. The distance over walks is defined as the normalized Euclidean distance measure on the walks' vectorial representation (Definition 3.28). The distance over trees is defined in Section 3.3 and is based on an alternating, recursive computation of the normalized Euclidean distances applied on the nodes of the trees and a modified version of the matching set distance measure, d_M, applied on the sets of children of the nodes.

As already mentioned, the different graph decompositions will be compared in the context of SVM with the graph kernels defined in the proximity space. For SVM the regularization parameter C was optimized in an inner 10-fold cross-validation loop over the set $C = \{0.1, 1, 10, 50\}$. In all the experiments, unless stated otherwise, we used the regularized version of the $\text{MCML}_{\text{full}}$ and NCA_{full} algorithms (Equations 5.7 and 5.9) where the regularization parameter λ is cross-validated over $\lambda = \{0, 0.1, 1, 10\}$; due to the poor performance of $\text{Xing}_{\text{full}}$ reported in Section 5.3.2 we did not use this method in the experiments. Note that we have 11 different instantiations of the $k_{Pd_{tuple,\boldsymbol{A}}}$ ($k_{Pd_{tuple,\boldsymbol{W}}}$) kernel, each one corresponding to one of the 11 set distance measures. In all the experiments accuracy was estimated using stratified 10-fold cross-validation and controlled for the statistical significance of observed differences using McNemar's test (McNemar, 1947), with significance level of 0.05.

5.4.2 Results and Analysis

Performance of Individual vs. Combined Decompositions

We first examine the influence of the length of the walks and heights of the trees on the predictive performance of SVM. The results for the mutagenesis dataset are presented in Figure 5.8; the results for the other datasets are given in Figures B.17 and B.18 in Appendix B.5. From the results it is clear that in mutagenesis the optimal decomposition depends on the actual set distance measure. For example, for d_{SMD} the highest predictive accuracy is obtained for walks of length 6 whereas for d_S the best decomposition is into walks of length 10. Additionally, with the exception of d_{SL}, decompositions in walks in general outperform decompositions into trees. For the other examined datasets the optimal decompositions are different. Thus we have no way to know a priori which type of decomposition is appropriate. By combining them using either NCA_{full} or $\text{MCML}_{\text{full}}$ (results also given in Figure 5.8) we usually get a classification performance that is as good as that of the single decompositions. Indeed, in mutagenesis the performance of MCML was significantly better in 120 cases, in 44 cases the differences were not significantly meaningful and in 1 case it was significantly worse compared to the single decompositions[4]; the corresponding

[4]The total number of comparisons is 165 = 11 walk decompositions x 11 set distances + 4 tree decompositions x 11 set distances

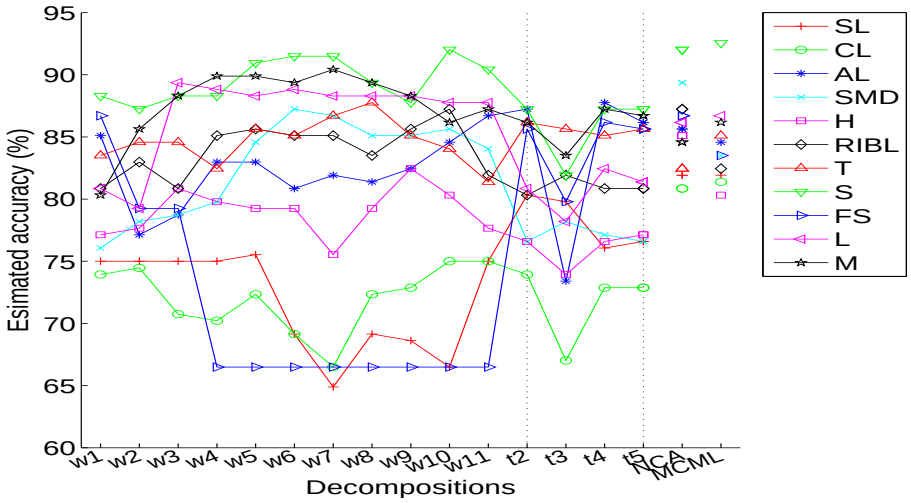
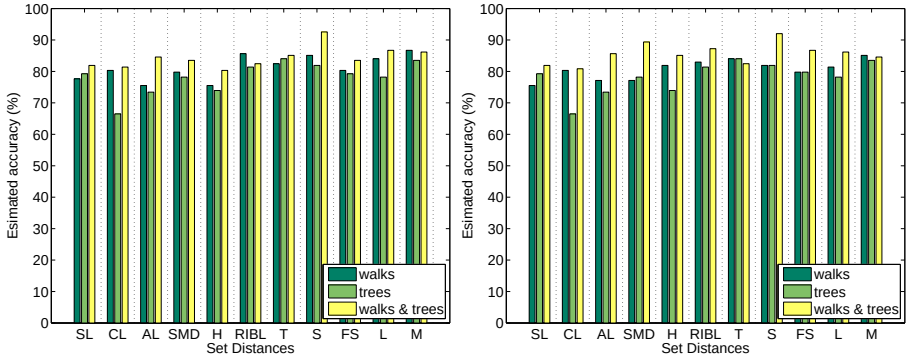


Figure 5.8: Estimated accuracy of different kernels in the proximity space ($k_{P,d_{set}}$) vs. different decompositions for different set distance measures in the mutagenesis dataset. wl denotes walks of length l whereas th denotes trees of height h. The last values in the plot denote the performance of $k_{Pd_{tuple,\boldsymbol{W}}}$ (and $k_{Pd_{tuple,\boldsymbol{A}}}$) where the different decompositions are combined.



(a) $\text{MCML}_{\text{full}}$ (b) NCA_{full}

Figure 5.9: Estimated accuracy (SVM) of different kernels in the proximity space, i.e. $k_{Pd_{tuple,\boldsymbol{A}}}$ (for $\text{MCML}_{\text{full}}$) and $k_{Pd_{tuple,\boldsymbol{W}}}$ (for NCA_{full}), combining only walks, only trees and both walks and trees vs. different set distance measures in the mutagenesis dataset.

number of cases when NCA was better, equal or worse than the performance of individual decompositions are 68, 92 and 5. The results for all the graph datasets are given in Table B.23 in Appendix B.5.

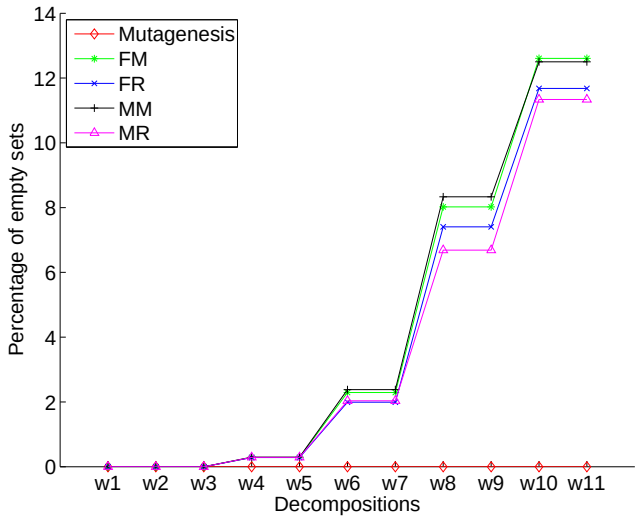


Figure 5.10: Percentage of empty sets vs. lengths of walks for the graph datasets (wl denotes walks of length l).

In Figure 5.9 we provide results of experiments on the mutagenesis dataset in which we combine only specific types of decompositions, i.e. only paths, only trees (we also repeat previous results where both paths and trees are combined); the visualizations for the FM, FR, MM, and MR datasets are given in Figures B.19 and B.20 in Appendix B.5. We can see that in general there is an advantage in combining *heterogeneous* decompositions as opposed to combing *homogeneous* decompositions only of specific type (only paths or only trees). Indeed, the homogeneous decompositions into walks and trees were never significantly better than than the decompositions into both walks and trees, and the latter decompositions were significantly better than both homogeneous decompositions in 2 cases for $\text{MCML}_{\text{full}}$ and 4 cases for NCA_{full}. The significance test results for other graph datasets are presented in Table B.24.

One problem with the decomposition of molecules into sets of walks of length l, in which tottering is not allowed, is that small molecules for large values of l will be associated with an empty set of walks[5]. To get more insight into this problem we present in Figure 5.10 the relation between the length of the walks and the percentage of molecules associated with empty sets. As we can see in the carcinogenicity datasets starting from walks of 8 or longer the number of molecules associated with empty sets starts to increase rapidly. On the other hand, in mutagenesis none of the molecules are associated with empty sets, even for the longest length of walks that we considered. As a result, one can expect

[5]The same observation holds also for decompositions into trees; however, the distance measure on trees from Section 3.3 can be applied on trees of different heights, hence we do not need to remove "incomplete" trees from corresponding decompositions.

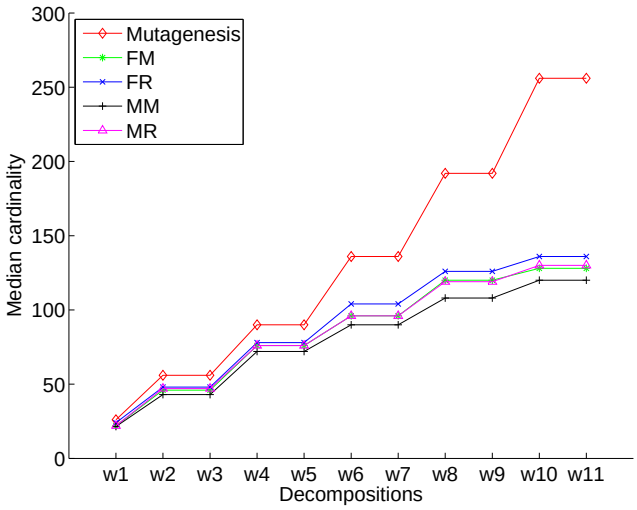
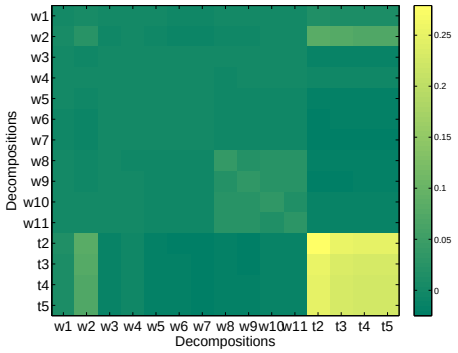


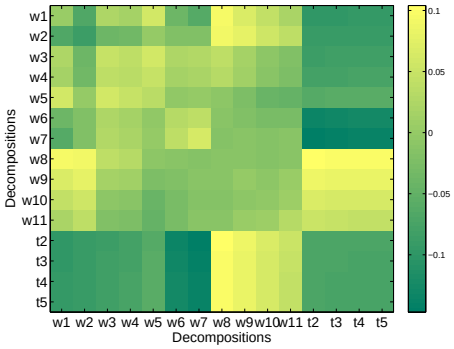
Figure 5.11: Medians of cardinalities vs. lengths of walks for the graph datasets (wl denotes walks of length l).

that the performance of a classifier operating on such incomplete descriptors will be harmed. On the other hand, the fact that a given molecule can not be described by walks of a specific length could be of predictive value. In Figure 5.11 we give the relation between the lengths of the considered walks and medians of cardinalities of the resulting decompositions; the longer the walks are the larger the cardinality of the resulting decompositions. More detailed statistics on the decompositions into walks in the considered graph datasets are presented in Figure B.21 in Appendix B.5.

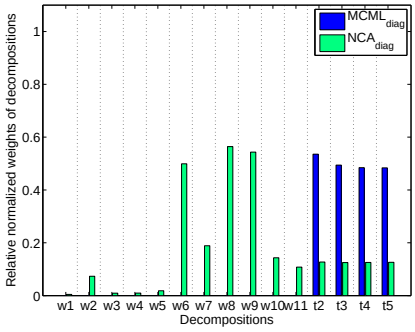
Similarly to the visualizations presented in Section 5.3.2 the results of the optimization process that learns the optimal decomposition can be graphically presented providing insight to the relative importance of different decompositions. In Figure 5.12 we give an example of such a visualization for the FM dataset and the d_{SMD} set distance measure; the visualization for other datasets (for d_{SMD}) is presented in Figure B.22 in Appendix B.5. In graph (c) of that figure the optimization was performed for diagonal matrices. The different elements in x-axis are the elements of the diagonal of the matrix $\boldsymbol{A}$ (or $\boldsymbol{W}^T\boldsymbol{W} = \boldsymbol{A}$) which correspond to a decomposition into walks and trees whereas the y-axis represents the (normalized) weights returned by the optimization method. What we see from the graph is that in FM for NCA_{diag} the highest weights are assigned to walks of lengths longer than 6 and all tress, independent of their heights, while $\text{MCML}_{\text{diag}}$ assigned high weights only to the trees. The graphs (a) and (b) of the Figure 5.12 provide the visualization of the optimization process for NCA_{full} and $\text{MCML}_{\text{full}}$, respectively, where now the matrix $\boldsymbol{A}$ is a full matrix. Note here that the off-diagonal elements in the full matrix correspond to hybrid distances, i.e. distances that are defined over two distinct decompositions, for example, for



(a) $\mathrm{MCML}_{\mathrm{full}}$, $\lambda = 0$



(b) $\mathrm{NCA}_{\mathrm{full}}$, $\lambda = 0$



(c) $\mathrm{MCML}_{\mathrm{diag}}$, $\mathrm{NCA}_{\mathrm{diag}}$

Figure 5.12: Relative importance of different decompositions (FM dataset) for the d_{SMD} set distance measure both for full and diagonal matrices $\boldsymbol{A}$ (for MCML) or $\boldsymbol{W}^T\boldsymbol{W} = \boldsymbol{A}$ (for NCA). wl denotes walks of length l whereas th denotes trees of height h. It is represented as normalized weights $\boldsymbol{A}$. Weights are normalized by a Frobenius norm of $\boldsymbol{A}$.

a given off-diagonal element one part might come from a decomposition to a set of walks of a given length and the other part from a decomposition to a set of trees of a given height. One observation is that for $\mathrm{NCA}_{\mathrm{full}}$ the decompositions into trees and long walks are assigned low weights. At the same time combinations of trees with long walks and combinations of shorter walks are of high importance.

Performance of Matchings Strategies

The next dimension of comparison is the relative performance of the instantiations of the $k_{Pd_{tuple},\boldsymbol{A}}$ kernel for different set distances and the following two set kernels based on averaging: (i) the direct sum kernel based on the cross product kernels applied on the different decompositions (we denote this kernel

Set Distance	mutagenesis	FM	FR	MM	MR
d_{SL}	81.91 (=)(=)	64.18 (=)(=)	64.96 (=)(=)	62.80 (-)(=)	63.66 (=)(=)
d_{CL}	81.38 (=)(=)	63.61 (=)(=)	65.53 (=)(=)	62.80 (-)(=)	64.24 (=)(+)
d_{SMD}	83.51 (=)(=)	64.76 (=)(=)	66.38 (=)(=)	66.07 (=)(=)	64.53 (=)(+)
d_H	80.32 (=)(=)	63.32 (=)(=)	66.95 (=)(=)	63.69 (=)(=)	59.30 (=)(=)
d_{RIBL}	82.45 (=)(=)	63.90 (=)(=)	65.81 (=)(=)	66.37 (=)(=)	60.17 (=)(=)
d_T	85.11 (=)(=)	64.18 (=)(=)	66.10 (=)(=)	67.56 (=)(=)	67.15 (=)(+)
d_S	92.55 (+)(+)	64.47 (=)(=)	65.81 (=)(=)	65.77 (=)(=)	63.37 (=)(=)
d_{FS}	83.51 (=)(=)	61.03 (=)(=)	66.10 (=)(=)	65.17 (=)(=)	62.79 (=)(=)
d_L	86.70 (=)(=)	64.47 (=)(=)	65.24 (=)(=)	65.47 (=)(=)	66.57 (=)(+)
d_M	86.17 (=)(=)	63.32 (=)(=)	67.52 (=)(=)	64.58 (=)(=)	63.08 (=)(=)
d_{AL}	84.57	65.04	66.95	66.96	61.34
$k_{\Sigma,CP}$	84.04	62.46	64.96	63.39	57.85

Table 5.3: Accuracy and significance test results of SVM in the graph datasets using $\text{MCML}_{\text{full}}$ (the + sign stands for a significant win of the first algorithm in the pair, - for a significant loss and = for no significant difference). The sign in the first parenthesis corresponds to the comparison of SVM vs. SVM with $k_{Pd_{tuple,A}}$ with d_{AL}, while the sign in the second parenthesis compares SVM vs. SVM with k_{CP}.

Set Distance	mutagenesis	FM	FR	MM	MR
d_{SL}	81.91 (=)(=)(=)	65.04 (=)(=)(=)	67.23 (=)(=)(=)	66.96 (=)(=)(+)	63.95 (=)(+)(=)
d_{CL}	80.85 (=)(=)(=)	60.46 (=)(=)(=)	63.24 (-)(=)(=)	65.18 (=)(=)(=)	67.73 (+)(+)(=)
d_{SMD}	89.36 (=)(+)(+)	62.17 (=)(=)(-)	67.52 (=)(=)(=)	66.37 (=)(=)(=)	65.99 (=)(+)(=)
d_H	85.11 (=)(=)(=)	64.18 (=)(=)(=)	66.09 (=)(=)(=)	66.07 (=)(=)(=)	65.11 (=)(+)(=)
d_{RIBL}	87.23 (=)(=)(=)	64.76 (=)(=)(=)	67.52 (=)(=)(=)	66.66 (=)(=)(=)	67.15 (+)(+)(+)
d_T	82.45 (=)(=)(=)	64.76 (=)(=)(=)	67.81 (=)(=)(=)	68.45 (=)(+)(=)	68.02 (+)(+)(=)
d_S	92.02 (+)(+)(=)	64.18 (=)(=)(=)	67.52 (=)(=)(=)	66.96 (=)(=)(=)	61.34 (=)(=)(=)
d_{FS}	86.70 (=)(=)(=)	61.60 (=)(=)(=)	64.10 (=)(=)(=)	63.99 (=)(=)(=)	61.34 (=)(=)(=)
d_L	86.17 (=)(=)(=)	64.18 (=)(=)(=)	65.81 (=)(=)(=)	64.88 (=)(=)(=)	66.28 (+)(+)(=)
d_M	84.57 (=)(=)(=)	64.46 (=)(=)(=)	66.66 (=)(=)(=)	65.48 (=)(=)(=)	67.73 (+)(+)(=)
d_{AL}	85.64	62.46	66.38	66.07	60.46
$k_{\Sigma,CP}$	84.04	62.46	64.96	63.39	57.85

Table 5.4: Accuracy and significance test results of SVM in the graph datasets using NCA_{full} (the + sign stands for a significant win of the first algorithm in the pair, - for a significant loss and = for no significant difference). The sign in the first parenthesis corresponds to the comparison of SVM vs. SVM with $k_{Pd_{tuple,A}}$ with d_{AL}, the sign in the second parenthesis compares SVM vs. SVM with k_{CP}, and finally the sing in the last parenthesis compares NCA_{full} with $\text{MCML}_{\text{full}}$ from Table 5.3.

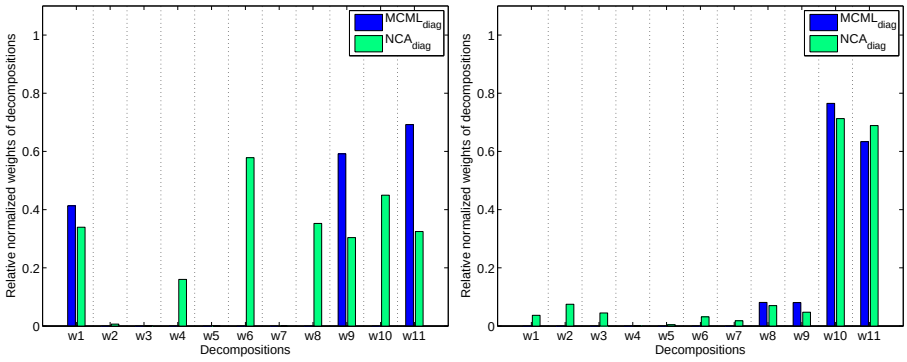
as $k_{\Sigma,CP}$), and (ii) the linear kernel in the proximity space induced by $d_{tuple,\boldsymbol{A}}$, also over the combination of the different $l+h$ decompositions, with the d_{AL} set distance measure. As already noted, the above two kernels constitute a standard approach to tackling graph problems. The main point of this comparison is to examine whether there are cases in which different ways of matching the elements of two sets of subgraphs can be more beneficial than the standard averaging which matches everything with everything.

The results (with the significance test results in parenthesis) are presented in Tables 5.3 (for $\text{MCML}_{\text{full}}$) and 5.4 (for NCA_{full}). From the results it is clear that the relative performance of kernels based on specific pairs of elements and kernels based on averaging depends on the actual application. For the mutagenesis and MR datasets there is an advantage of the kernels based on specific pairs of elements, while for the remaining datasets the performances are very similar. Overall, the choice of the appropriate way of matching the elements of two sets depends on the application and ideally should be guided by domain knowledge, if such exists. Nevertheless, the relative performance of the different kernels provides valuable information about the type of problem we are facing. For example examining mutagenesis and carcinogenicity we see that although they correspond to the same type of classification problem, i.e. classification of graphs, in the latter (except for the MR) averaging works better, hinting that the global structure of the molecules is important, whereas in the former averaging performs poorly, indicating that matching specific components of the molecules is more informative. It should be noted that these results agree with the observation from Section 4.4.4 where among others we compared kernels based on averaging and kernels based on specific pairs of elements; the difference is that in Section 4.4.4 we used only one graph representation that was based on trees.

The last parenthesis in Table 5.4 compares the performances of NCA_{full} vs. $\text{MCML}_{\text{full}}$. From the results we can see that in general these two methods achieve rather similar performances, i.e. in 51 out of 55 cases the differences in performances between these two methods were not statistically significant, while in 3 (1) cases NCA_{full} was significantly better (worse) than $\text{MCML}_{\text{full}}$.

Importance of Size of Sub-substructures

One of the observations presented in the previous section, where the results presented in Figure 5.12 were discussed, was that in some cases longer walks might be of high discriminatory information. This observation is somehow in contradiction to the common practice in various graph kernels (Gärtner et al., 2003; Collins and Duffy, 2002), based on the cross product kernel where typically larger structures are down-weighted. To further investigate the issue we decomposed graphs into walks of different lengths and visualized the relative importance of the resulting decompositions. In order for the weights to have a clear semantics we limited to a diagonal matrix of weights $\boldsymbol{A}$ (or $\boldsymbol{W}$). We examined the weights learned on the d_{AL} set distance measure. We have chosen d_{AL} because its semantics are very similar to the cross product kernel in the

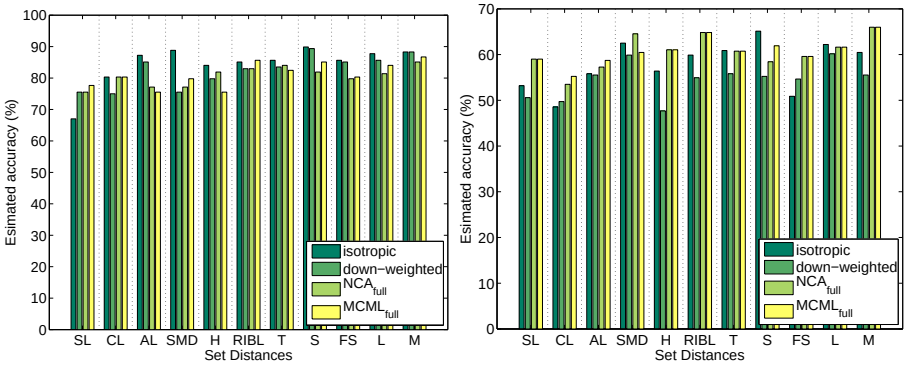


(a) mutagenesis (b) FM

Figure 5.13: Relative importance of different decompositions (mutagenesis and FM datasets) for the d_{AL} set distance (wl denotes walks of length l).

sense that it is also based on averaging, i.e. each element of one decomposition is matched against each element of the other decomposition. The results for the mutagenesis and FM datasets are given in Figure 5.13; the visualization for other datasets is presented in Figure B.23 in Appendix B.5. As we can see for both datasets the decompositions to sets of walks of longer lengths are given a high relative importance. Again, by examining these findings in view of Figure 5.10 we see that for the FM dataset a large percentage of the molecules are associated with empty sets of walks of length 10 and 11, roughly 12 %. So, in accordance with the observations in the above paragraph it seems that the absence of longer walks does indeed convey discriminatory information. However, if we turn now to the mutagenesis dataset we see that this has no empty sets for decompositions to sets of walks of longer lengths; in fact there are no empty sets at all, but still these decompositions are assigned a large importance. So it seems that the presence or absence of longer walks does not solely explain the importance assigned to the corresponding decompositions. The above indicates that larger structures can indeed convey important discriminatory information, not limited to their presence or absence, and they should not be down-weighted in advance.

To further analyze this aspect we examined how the performances of NCA_{full} and $\text{MCML}_{\text{full}}$ (where λ was internally cross-validated) compare with two baseline weighting schemes of different decompositions which are widely used in practice. In the first (*isotropic*) weighting scheme all the decompositions are assumed to be equally important and assigned equal weights of 1. In the second weighting scheme the decompositions into weights of length l are *down-weighted* as $1/2^l$; a similar scheme is used e.g. in the graph kernels of Gärtner et al. (2003). The visualization of the results for the mutagenesis and MR datasets is provided in Figure 5.14 where for each set distance measure we report the four estimated accuracies corresponding to the above two standard weighting



(a) mutagenesis (b) MR

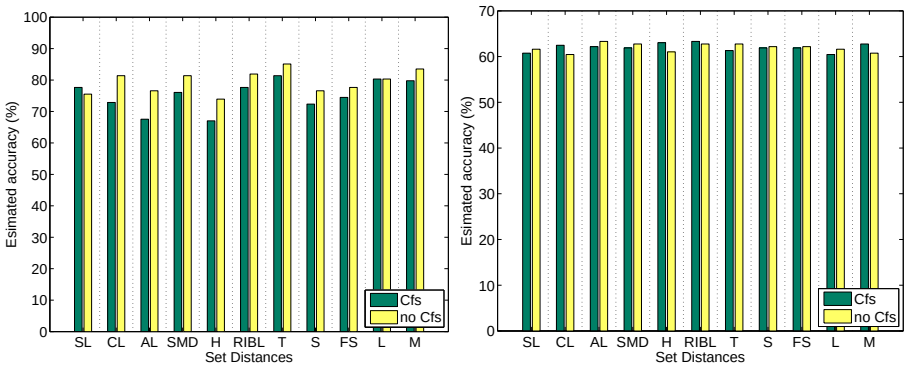
Figure 5.14: Performances (using SVM) of the various weighting schemes for mutagenesis and MR datasets.

schemes and the weighting schemes obtained by NCA_{full} and $\text{MCML}_{\text{full}}$. The results for other datasets are presented in Figure B.24 in Appendix B.5. From the results it is clear that the optimal weighting scheme depends on the problem at hand. While in mutagenesis the isotropic and down-weighted weighting schemes seem to have an advantage over the more complicated ones, in MR the opposite trend holds. The significance results are reported in Table B.25.

To explain the surprisingly good behavior of the simple weighting schemes we first note that the predictive error of the kNN algorithm is largely dominated by the high-variance component; the bias component is usually low since this algorithm does not make strong assumptions on the distribution of the data (Hastie et al., 2001). Now, by using multiple decompositions we in fact reduce the variance, and hence the error. NCA_{full} and $\text{MCML}_{\text{full}}$ also use multiple decompositions; however, in comparison with the simple weighting schemes, the reduction in variance is potentially lower since $\frac{m(m-1)}{2}$ parameters of the matrix $\boldsymbol{A}$ need to be estimated, and as we have shown in Section 5.3 this procedure might be sensitive to perturbations in the training set. This observation agrees with the empirical results presented above, since the highest advantage of the simple weighting schemes was reported in the mutagenesis dataset which, in comparison with carcinogenicity, has small number of instances, i.e. 188.

Performance of Prototype Selection Methods

Finally, we examined whether it is possible to reduce the size of the representation set without a reduction in the classification performance of our graph kernels. As described in Section 4.2.4 it is possible to select the prototypes by a feature selection algorithm working in the proximity space. The feature selection algorithm we experimented with is CFS (Hall, 1998). CFS evaluates the worth of a subset of attributes by considering the individual predictive ability of each



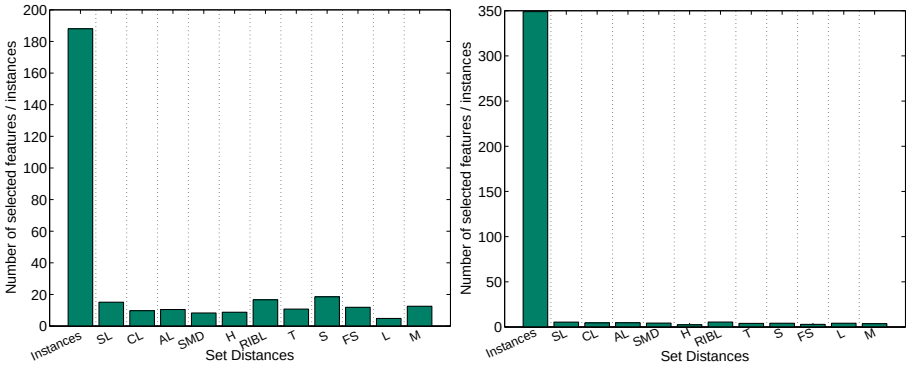
(a) mutagenesis (b) FM

Figure 5.15: Accuracy of $k_{Pd_{tuple,\boldsymbol{A}}}$ (using SVM) for the mutagenesis and FM datasets and for different set distance measures. The results are obtained using $\text{MCML}_{\text{diag}}$ where feature selection in the proximity space is performed using the CFS method (the first bar). For comparison we also report the accuracy of the $k_{Pd_{tuple,\boldsymbol{A}}}$ on the full set of prototypes (the second bar).

feature along with the degree of redundancy between them. Subsets of features that are highly correlated with the class while having low inter-correlation are preferred. We should note that CFS automatically selects the most appropriate number of features-prototypes. In all the experiments with prototype selection we used the NCA_{diag} and $\text{MCML}_{\text{diag}}$ methods where λ was fixed to 0.

The accuracy results for the CFS algorithm for the mutagenesis and FM datasets (using $\text{MCML}_{\text{diag}}$) are given in Figure 5.15; the results of other datasets is given in Figures B.25 (for $\text{MCML}_{\text{diag}}$) and B.26 (for NCA_{diag}) in Appendix B.5. The main observation is that for the FM dataset the performance is not harmed when a subset of the training instances is used. Indeed, in this dataset all the differences in accuracies between the "full" and "sparse" models were not statistically significant. In mutagenesis the advantage of the non-reduced data representation is more evident which might indicate that in mutagenesis the data is uniformly distributed making it harder to select a small set of prototypes representative of the underlying distribution. In this dataset the performance of the model based on all the prototypes was significantly better in 3 cases for MCML and 3 cases for NCA; in the remaining cases the differences between the performances were not statistically significant. In FR and MM the differences for all the methods were not statistically significant. In MR and for NCA the "full" models were better in 4 cases, while in MCML one "full" model was significantly worse than the corresponding "sparse" model.

As already mentioned, CFS feature selection automatically selects the most appropriate number of features. Figure 5.16 presents the number of selected features (prototypes) in mutagenesis and FM by the CFS algorithm for the



(a) mutagenesis (b) FM

Figure 5.16: Number of selected features (prototypes) by the CFS algorithm in the mutagenesis and FM datasets. The results are reported for different set distance measures and using $\text{MCML}_{\text{diag}}$. The first bar denotes the number of instances in the training set.

different set distance measures; similar graphs are presented in Figures B.27 (for $\text{MCML}_{\text{diag}}$) and B.28 (for NCA_{diag}) in Appendix B.5. The numbers of selected features are averaged over the models built by CFS within the folds in cross validation[6]. For comparison, we also provide the number of training instances in the considered datasets. From these results it is clear that all the resulting models are sparse. In particular in mutagenesis by selecting approximately 10 % of initial instances we obtain results which are similar to the results where the full dataset was used. For the other datasets the number of selected prototypes was at most 3% of the training set, thus the final models rely on a very small representation set. The sparsity of the above models in the examined datasets considerably improves the computational efficiency during the testing phase.

Comparison with Other Graph Kernels

To situate the performance of our kernels to other graph kernels we provide in Table 5.5 the results reported in the literature on the same benchmark datasets; these are the results already reported in Table 4.5 under column "Kernel-based"[7]. Additionally, we provide the best results of the cross product kernels from Section 4.4.3 or kernels based on specific pairs of elements from Section 4.4.4; these results are also reported in Table 4.5. The corresponding results show that our kernel achieves a performance accuracy which is similar

[6]In total there are 400 = 10x10x4 folds as in each fold of 10-fold cross-validation we internally cross-validate (again in 10-folds) over $C = \{0.1, 1, 10, 50\}$.

[7]As in Table 4.5 we only cite works which use similar features to describe atoms and bonds. For example, our results in carcinogenicity are not directly comparable with the ones reported by Fröhlich et al. (2005).

Graph kernels from:	mutagenesis	FM	FR	MM	MR
(Kashima et al., 2003)	85.1	63.4	66.1	64.3	58.4
(Mahé et al., 2004)	91.0	-	-	-	-
(Ralaivola et al., 2005)	91.5	64.5	66.9	66.4	65.7
Best Cross Product Kernel	87.2	62.5	*68.4*	64.6	61.3
Best kernel based on mappings	85.6	*67.0*	67.8	66.7	64.5
Best adaptive kernel (this section)	*92.5*	65.0	67.8	*68.4*	*68.0*

Table 5.5: Comparison with other related graph kernels. The best kernels in each datasets are emphasized. The results for the best Cross Product Kernel are taken from Section 4.4.3; the results the best kernel based on mappings are taken form Section 4.4.4.

or better than the performance of the existing graph kernels. In particular, for mutagenesis, MM and MR we get classification results that are better than the best results reported so far in the literature. These performances are also better than the results reported in the previous chapter; however, the differences are not statistically significant. The two main sources of the above improvements over the current literature results are the possibility of flexible matchings between the elements of the decompositions and the fact that we can combine decompositions of different types. In FM and FR the results are better than the results reported elsewhere; however, they are worse than the results obtained using the cross product kernels and kernels based on specific pairs.

5.5 Related Work

In this section we will describe the existing work that is relevant in our context. More precisely, we report previous work on metric learning and methods for generating representations for graph data. Graph kernels were also discussed in Sections 4.5 and 5.2.1.

Metric Learning

Related work on metric learning can be categorized into supervised metric learning, where the goal is to learn the metric from labels (or more generally from side-information), and unsupervised metric learning where the goal is to find a low-dimensional embedding of the data such that the topological relations between the instances are preserved. For a survey on metric learning the reader is referred to (Yang and R.Jin, 2006). The focus in this section is on the supervised methods as these are more relevant in our context.

As already noted, any supervised (or based on side-information) metric learning method can be adapted to the representation or distance combination in complex domains, as long as in the objective function the access to data is only through a distance function. A classical example of such method is Linear

Discriminant Analysis (LDA) (Bishop, 2006, e.g.) which defines a projection that maximizes the distance between the means of the classes while minimizing the variance within each class. The main problem with this method is that the dimensionality of the projected space depends on the number of classes and not on the inputs themselves. Moreover, LDA is based on the Gaussian assumption (which is rarely true in practice) since it uses covariance matrices. Additionally, LDA suffers from a small sample size problem when dealing with high-dimensional data. As already noted, MCML is a generalization of LDA that makes a much weaker assumption, namely that each class is distributed as a uni-modal blob, which can be separated (under the right metric) from the other class blobs (Globerson and Roweis, 2006). We mention that several other extensions of the standard LDA have been proposed in the literature. For example, Sugiyama (2006) proposed Local Linear Discriminant Analysis which extends LDA by assigning greater weights to those instances that are close together.

The RCA algorithm from (Bar-Hillel et al., 2005) constructs a Mahalanobis metric from a weighted sum of in-class covariance matrices; however, it only takes into account similar pairs of points and discards the dissimilar ones. As a result it is unlikely that this method will perform well on fully labeled data. The objective functions in this method can be shown to be similar to the one of Xing's method (up to a constant); however, the constraint in the optimization problem in RCA, i.e. $det(\boldsymbol{A}) \geqslant 1$, is in general more difficult to fulfill. One may imagine the straightforward extension of RCA that would also take into account the dissimilar pairs and try to maximize some measure of the variance or the distance between the dissimilar pairs while minimizing the corresponding measure between the similar pairs, an approach that would be in fact similar to the one used in LDA. An example of such extension is the Discriminative Component Analysis (DCA) proposed by Hoi et al. (2006) which exploits cannot-link constraints and captures non-linear relationships using contextual information.

The method proposed in (Schultz and Joachims, 2004) learns the metric from relative and qualitative examples of the form "A is closer to B than A is to C". It uses a different form of side-information which renders the method applicable in a wider range of domains than it is the case with other metric learning methods. However, such side-information might not be always easy to obtain.

Two algorithms from (Shalev-Shwartz et al., 2004) and (Weinberger et al., 2006) can be easily adapted in our context. The cost functions in these algorithms are based on the notion of large margin which separates elements with different labels while keeping elements of the same class together. The main difference is that the latter focuses on the local neighborhood while the former seeks to minimize distance between *all* similarly labeled examples. The other algorithm which can be easily adapted to complex objects is the method of Chopra et al. (2005) which is parameterized by pairs of identical convolution neural networks.

As already noted, some of the metric learning methods have been recently kernelized which enables their use on complex data as long as a valid kernel

function over the learning objects is defined. For example, the work of Tsang et al. (2005) extends the RCA algorithm from (Bar-Hillel et al., 2005), the algorithm presented by Kwok and Tsang (2003) can be seen as the extension of the Xing algorithm from Section 5.1.2, and Kernel DCA of Hoi et al. (2006) is a direct extension of DCA. The one problem with these methods is that most of the kernels for complex objects are defined as cross product kernels between sets of objects' decompositions (see discussion in Section 4.5), i.e. they are based on averaging, and hence their expensiveness is limited. In contrast the different distances on decompositions from Equations 5.1 and 5.2 give us more flexibility in defining (dis-)similarities between complex objects.

In addition to metric learning several attempts have been recently made to learn kernel operators directly from the data (Lanckriet et al., 2004; Ong et al., 2005; Cristianini et al., 2002; Chapelle et al., 2002; Duan et al., 2003; Bousquet and Herrmann, 2003; Crammer et al., 2002). This approach is more general than metric learning in the sense that any valid kernel k can be directly used to compute a pseudo metric in the feature space. The proposed methods differ in the objective functions (e.g. CV risk, margin based, etc.) as well as in the classes of kernels that they consider (e.g. finite or infinite set of kernels, etc.). In the case of set distances based on kernels there are two main problems. First, most of the kernels over sets proposed so far in the literature are based on averaging (i.e. they depend on *all* the elements in the two sets). As mentioned above, this feature might be inappropriate for some applications (e.g. multiple-instance problems) where the classification depends on *specific* pairs of elements from the two sets. Second, with the exception of the work of Argyriou et al. (2005), all of the methods work only in a transductive setting, i.e. completing the labeling of a partially labeled dataset. This in turn limits the application area of such methods.

The goal of unsupervised metric learning methods is to find an embedding of the original feature space. These methods are divided into linear and non-linear methods. Well-known methods for linear dimensionality are the Principal Component Analysis (Bishop, 2006) where the goal is to preserve the variance of the data, Multidimensional Scaling (Cox and Cox, 1994) that finds a projection that best preserves the distances between the instances, and Independent Component Analysis (Comon, 1994) that seeks a linear transformation to coordinates in which the data are maximally statistically independent. Well known methods for non-linear unsupervised metric learning methods are ISOMAP (Tenenbaum et al., 2000) which seeks an embedding that best preserves the geodesic distances between the instances, and Locally Linear Embedding (Roweis and Saul, 2000) and Laplacian Eigenmap (Belkin and Niyogi, 2003) that focus on preserving local neighborhood structures. For an overview of non-linear unsupervised methods the reader is referred to (Saul et al., 2006).

Representation Generation for Graphs

Representation (or feature) generation for structured data has been proved to be difficult in practice due to the vast number of possible feature candidates.

In particular, in graphs where the number of possible representations is virtually unlimited, one can not hope to extract relational features (i.e. sub-graphs) without some assumptions on the type of generated features. As a result, several methods have been proposed in the literature (Rückert and Kramer, 2007; Deshpande et al., 2003; Kramer et al., 2001; Horváth et al., 2006) to automatically extract subgraphs that correspond to frequently occurring, maximally class-correlated, or maximally diverse subgraphs. The generated features are propositional, they assume a single value that indicates either presence or absence of the corresponding relational structure, or the frequency with which it is encountered, within a given training instance. Subsequently, any standard propositional framework can be applied on the set of extracted features.

Kudo et al. (2005) proposed a classification method which does not make any assumptions about the type of generated subgraphs. The actual learning algorithm used is AdaBoost (Bishop, 2006) where binary decision stumps taking the corresponding subgraphs as arguments (one subgraph for one decision stump) are exploited as weak learners. Instead of exhaustively enumerating all the subgraphs of a given graph (which is not feasible) the authors propose a variant of branch-and-bound algorithm which allows for an efficient pruning of the search space which is based on upper-bounding of the error used within the AdaBoost algorithm. As already noted, this method does not make any assumptions about the types of subgraphs; however, the classification model is in fact based on the weighted binary representation of the graph data where the binary features (generated by the decision stumps) correspond to a particular subgraph.

The representation combination methods proposed in this chapter are based on the assumption that the a priori selected decompositions provided by the analyst are informative for the problem at hand. However, in contrast with all the previous approaches the features do not just indicate presence or absence of the corresponding relational substructure, instead the features are relational in nature. The main advantage of this approach is that for these features we can apply operators (e.g. distances and kernels) which take into account the features' relational nature. As a result the algorithms operating over this representation are of higher expressivity.

5.6 Conclusions

In this chapter we proposed a general framework that allows for combining different composite representations of a given learning problem and/or different distances defined on these representations. The building blocks of our method are couplings of representations and distances over these representations, the relative importance of which is learned directly from the training data. These building blocks are a priori provided to the algorithm, and reflect the practitioner's "initial guesses" of good representations and distance. We exploited ideas developed previously on metric learning developed for propositional data and adapted three methods so that they can be used in the context of compos-

ite structures. These methods boil down to mathematical optimization problems which are amenable to various optimization techniques. We also proposed two techniques for regularizing the solutions of our adaptive methods.

We showed the utility of the proposed framework for two learning problems. The goal in the first problem is to combine distances over objects which are represented as sets, i.e. the combined elements are different distances on sets. We demonstrated that for various learning problems it is indeed possible to increase the predictive performance of kNN by combining different set distance measures. In particular, in the diterpenes dataset the results we reported are the best reported so far. We also showed that the results of the optimization processes can be visualized to provide insight into the relative importance of the various distances. Finally, we demonstrated that it is possible to reduce the size of the problem by combining only a few of the top set distances which are selected according to the assigned coefficients.

In the second learning task we analyzed the performance of our framework for the task of combinations of various graph decompositions into substructures of specific types. We exploited this adaptive combination to construct positive semi-definite kernels in the proximity space resulting in a flexible and powerful class of graph kernels. These kernels for graphs are based on walks and trees without repetitive occurrences of nodes, are computable in polynomial time, are positive semi-definite, and are applicable to a wide range of graphs. The distinctive feature of our kernels is that they allow for combinations of different types of decompositions, and instead of accounting for all the subparts of a given decomposition our framework allows for specific types of mappings between subparts, exploiting notions from set distances. We reported experimental results for the task of activity prediction of drug molecules. Our main observation is that the performance of our kernels which combine several decompositions is in most of the cases not worse than the performance of the corresponding single decompositions. Moreover, depending on the actual application, the adaptive matching based kernels have an advantage over kernels based on all possible pairs of points from decompositions. Finally, we demonstrated that state-of-the-art classification performances can be achieved.

Chapter 6

Overview, Discussion and Future Work

The goal of this dissertation was to examine various aspects of the distance- and kernel-based learning paradigms applied in relational settings. As a first step, in order to represent composite objects, we defined a representation language that is built over an extension of relational algebra, a fact that makes it directly accessible to a wide audience that is familiar with the relational algebra concepts. The proposed formalism is typed, modular, intuitive to use and naturally extends the propositional case. The relational instances are defined in a recursive manner traversing the relations of the relational schema in order to gather the relevant information. The recursion among the different relations is guided by the concept of links which are based on the notion of foreign keys. Foreign keys provide a natural and intuitive way to provide a declarative bias and render unnecessary type definitions extensively used in inductive logic programming.

At the core of our extended relational representation formalism there are three data types, i.e. tuples, sets and lists. By combining these data types we can directly model a variety of complex structures such as trees, graphs (through various approximations) or more general structures that do not fall to a specific topological category. Associated with each basic data type there is a number of different data mining operators, each one with different semantics, and it is up to the analyst to declare which specific operator should be used for a given data type. In our system it is also possible to define different operators for objects of the same general type e.g. sets of lists, sets of tuples, etc. The final operator over the full complex learning instances is given as a recursive combination of operators assigned to the sub-structures which constitute the learning instances. Obviously, there are as many different instantiations of the final data mining operator as there are combinations of data mining operators over the components of the full complex learning instances. The above flexibility of our system is in contrast with the most of the existing relational data mining algorithms and systems which can be characterized as monolithic in the sense

that they either focus only on a specific complex data type, or they rely on a single type of data mining operator, or most often both. In this study we have only focused on distance- and kernel-based data mining operators.

One of the core elements of our relational learning approach is the decomposition of the complex relational objects into multi-sets of simpler components which have to be matched in a systematic manner. This matching is typically achieved through the definition of a set operator matching the elements of the corresponding multi-set decompositions, in order to compute the final data mining operator, whether kernel- or distance-based. For example, within the context of relational learning with kernels over labeled graphs, the graphs were decomposed into sets of walks, trees, tree-like structures etc. No matter what the decomposition was, it always resulted in a set of elements that required the definition of a set operator. We also mention that other complex structures, such as lists, or even tuples, can be considered as special cases of sets that simply possess more structure, thus less degrees of freedom, reducing the number of permissible matchings. For example, in distances and kernels over sequences considered in this thesis, sequences are decomposed into sub-sequences and the elements of decompositions can be matched only in such ways that retain the relative order of sub-components within the original sequences. Other than that, the final operator is computed in the same manner as in the case of sets, usually as some aggregation function over the matched components.

The distance-based approaches over multi-sets of decompositions of composite objects have given rise to a number of set distances where the main difference came from the family of permissible matchings considered each time (and the aggregation function over the matched components). We examined the behavior of set distances corresponding to different types of mappings over a number of relational benchmark datasets. Overall, the examined set distances gave satisfactory results when used together with the k-Nearest Neighbor (kNN) algorithm; however, no general statement can be made about the superiority of one set distance measure over another. In general, everything depends on the type of application and its underlying assumptions which should guide the selection of the distance measure. However, there are some distance measures that exhibited a good and stable performance over the datasets that we examined. For example, d_{SMD}, the good performance of which coupled with its quadratic computational complexity make it a good first choice. The characterization of the various instantiations of the set distance measures was supported by the empirical results, i.e. set distances that were semantically similar were in general also similar in terms of their relative performance on the relational benchmarks examined. Finally, the right choice of distance measure gives encouraging classification results that in many cases compare favorably with these of other relational learners reported in literature. For example, in the diterpenes dataset we obtained the accuracy of 97.41 % which was better than the best result from the literature.

In comparison with set distance measures based on different mappings, the options in kernels are much more limited due to the requirement of positive semi-definiteness. In fact, the only known matching strategy which is guaranteed to result in a valid set kernel is based on averaging, matching of all elements of

one (multi-)set to all elements of the other (multi-)set. Our experimental results demonstrated that this simple way of matching the elements in general achieves a good predictive performance, and by carefully selecting parameters of this kernel, it compares favorably with other relational kernel-based systems, e.g. in version 2 of musk we reported 94.12 % of accuracy which is the best result obtained so far. Additionally, this kernel was shown to be quite stable to the parameter setting of the kernels defined over the elements of the sets, different normalization schemes, etc. This stability might be precisely a result of the above mentioned "averaging" property. Finally, we note that by using kernels based on lists, which as already mentioned can be considered as a special case of matching-based kernel on sets, in protein fingerprints we obtained state-of-the-art cross-validation results of 87.69 % (the accuracy on a holdout test set was 87.35 %).

The above mentioned kernels which work by matching everything with everything might be inappropriate in cases where only specific elements of decompositions are important for a problem at hand (e.g. multiple-instance problems). To address this limitation we proposed three new and flexible families of kernels over sets, where the overall similarity is based only on specific elements of the two sets. More precisely, these kernels are defined respectively as the sum of elementary kernels between specific pairs of elements, set distance substitution kernels, and kernels in the the proximity space induced by set distances. The main problem with kernels in the first group and the distance substitution kernels is that they are not positive semi-definite in general; however, encouraged by recent experimental and theoretical results we are able to use such kernels within Support Vector Machines (SVM). The main finding which came out of our experiments is that the relative performance of kernels based on specific pairs of elements and kernels based on averaging depends on the actual application. An even more interesting observation is that in datasets corresponding to the same general type of learning problem (i.e. molecule classification) different matching strategies were more appropriate. e.g. in carcinogenicity the global structure of the molecules are important, while in mutagenesis only the specific components of the molecules are informative. We note that the state-of-the-art averaging-based Bhattacharrya set kernel from (Kondor and Jebara, 2003) performed poorly in all the examined benchmark datasets. The other finding is that the kernels in proximity space outperform distance substitution kernels and kernels directly based on specific pairs of elements. Moreover, in general SVM with our kernel outperformed the standard kNN where the set distances are exploited in a "naive" way. We also mention that in version 1 of musk we obtained 94.54 % accuracy, which is the best result reported so far.

Finally, we addressed the problem of adaptation of representation of learning instances and data mining / machine learning operators applied on the selected representation. The motivation for this work were the experimental results discussed above, which clearly indicated that there is no distance (kernel) which is overall better than any other, and the optimal representation of learning instances depends on the problem at hand. The approach we followed was to combine a number of predefined representations and operators. Couplings of rep-

resentations and operators are the building blocks of our method, the relative importance of which is learned directly from the training data. These building blocks are a priori provided to the algorithm, and reflect the practitioner's "initial guesses" of good representations and operators. This approach allows for the simultaneous combination of representations and operators, nevertheless one can choose to focus on only one of them, i.e. one can apply a fixed operator on different decompositions, or apply different operators which are defined on the same representation. We only focused on the distance-based paradigm and exploited three metric learning methods which were developed for propositional data, and adapted them so that they can be used in the context of composite structures. These methods boil down to mathematical optimization problems which are amenable to various optimization techniques. We also proposed two techniques for regularizing the solutions of our adaptive methods.

We demonstrated the utility of the proposed framework for two learning problems. The goal in the first task was to combine a number of predefined set distances. For various learning problems it was indeed possible to increase the predictive performance of kNN by combining different set distance measures. Moreover, it was possible to reduce the size of the problem by combining only a few of the top set distances which were selected according to the assigned coefficients. We demonstrated that the results of the optimization processes could be visualized, providing insight into the relative importance of the various distances. Finally, in the diterpenes dataset we reported predictive accuracy of 98.34 % which is the best result reported in the literature so far.

In the second learning task we analyzed the performance of our framework for the task of combinations of various graph decompositions into substructures of specific types. We exploited this adaptive combination to construct positive semi-definite kernels in the proximity space resulting in a flexible and powerful class of graph kernels. It has been argued in (Borgwardt and Kriegel, 2005) that a "good" kernel for graphs should fulfill at least the following requirements: (i) should be a good similarity measure for graphs, (ii) its computation should be possible in polynomial time, (iii) should be positive semi-definite and (iv) should be applicable for various graphs. Our adaptive kernels for graphs which are based on walks and trees without repetitive occurrences of nodes, are computable in polynomial time, positive semi-definite and applicable to a wide range of graphs. Additionally, there are two distinctive features of the class of graph kernel we defined. First, they allow for combinations of different types of decompositions. We have demonstrated this by combining decompositions that correspond to walks and trees, however in the same manner we can combine decompositions based on cyclic patterns, paths, etc. We show that our combination schema solves the problem of selecting the appropriate representation. Second, instead of accounting for all the subparts of a given decomposition our framework allows for specific types of mappings between subparts, exploiting notions from set distances. To practically demonstrate the effectiveness of our graph kernels we report experimental results for the task of activity prediction of drug molecules. Our main finding is that the performance of our kernels which combine several decompositions is in most of the cases not worse than the performance when

the single decomposition is used. Next, depending on the actual application, the adaptive matching based kernels have an advantage over kernels based on all possible pairs of points from decompositions. Finally, we demonstrated that state-of-the-art classification performance can be achieved. In particular for the mutagenesis, MM and MR datasets we get classification accuracy of respectively 92.5 %, 68.4 % and 68.0 %, that are better than the best results reported so far in the literature

6.1 Future Work

In this section we will list the potential extensions of the work presented in this thesis. More precisely, in Section 6.1.1 we raise the issue of adaptive generation of representation of composite objects; in Section 6.1.2 we discuss the limitations of the different cost functions used in Chapter 5; in Section 6.1.3 we argue that more aggressive regularization techniques in the context of our adaptive framework might be more beneficial; in Section 6.1.4 we discuss the extension of our system for learning with side-information; in Section 6.1.5 we provide a description of how to learn adaptive operators over sets; and finally in Section 6.1.6 we argue that it is important to theoretically analyze the set kernels based on mappings.

6.1.1 Extracting Relational Representations

Till now we have considered that the system is given a set of representations and operators for a given relational problem. In Chapter 5 we demonstrated that it is possible to learn how to combine these representations and operators by solving an optimization problem over the given learning data. It would be interesting to go one step further and turn to the automatic extraction of discriminative relational representations, which can be seen in fact as an automatic extraction of relational features, that will be used subsequently within the combination learning module.

The main idea to pursue is the following. Once the data analyst has given to the system a set of learning data stored in the form of a general relational dataset, the system will construct a set of relational features by exploring the search space which is defined by the relational schema of the learning dataset. This is a typical search problem so we need a cost function to evaluate the current state, a number of operators that allow us to move in that space by generating new states, and a structure over our search space. This approach is similar to approaches used in the context of propositionalization of relational problems. For example in the context of labeled graphs several methods have been proposed in the literature (Rückert and Kramer, 2007; Deshpande et al., 2003; Kramer et al., 2001; Horváth et al., 2006) to automatically extract relational features that correspond to frequently occurring, maximally class-correlated, or maximally diverse subgraphs, etc.

The possible cost functions that are worth considering in this context are the

cost functions that we have used as a part of the optimization problem where the goal is to learn to combine representations and operators. Nevertheless, here only a single representation (i.e. relational feature) will be examined at time for inclusion in our set of representations; thus there is no optimization problem, just an evaluation of the quality of the given relational representation. For example, Xing's cost function from Section 5.1.2 tries to optimize a measure of class separation which boils down to how compact are the different classes and far apart from each other. It is obvious that the same measure can be used to judge the quality of the different relational features. Discriminative features will be the ones that reduce the distances of instances of the same class, while keeping far apart instances from different classes.

The structure of the search space, at a first stage, could be determined by the relational schema, a relational feature generation mechanism will be parsing the relational schema, in a depth-first manner, starting the search from the relation that was defined as the main relation. We will have a restricted dictionary of parsing operators, such as paths and trees, or combinations of them. The system would automatically extract such representations from the training data and evaluate their quality with respect to the employed cost function. Special care should be given to the incremental construction of the relational features; since the search space is very large, one needs to derive efficient ways of generating these features that do not require a complete re-computation of the cost functions. For example, in the case of paths the quality of paths of length n can be based on an incremental computation of the quality of the paths of length $n-1$. Different heuristic criteria need to be established that will determine when the search should stop; these could be simple conditions such as search terminates when the quality of the extracted relational features does not improve above a threshold.

6.1.2 Cost Functions

As already mentioned in Section 5.1, the three cost functions, i.e. Xing's, MCML's and NCA's we have explored for representation and operator combination, make different assumptions about the data distribution which makes them suitable for different types of problems. For example, Xing's and MCML's functions implicitly assume that the instances of each class form a single compact and connected set. The main advantage of these cost functions is that they are convex. On the other hand, the NCA method is non-parametric and makes no assumptions about the shape of the class conditional distributions; however, the price for this flexibility is that the objective function is not convex.

One way to address the problems of the above cost functions is to use the method proposed by Weinberger et al. (2006). The corresponding cost function is based on the notion of large margin which separates elements (only in the local neighborhood) with different labels while keeping elements of the same class together. No assumptions are made about the structure or distribution of the data and the optimization problem is cast as an instance of semi-definite programming. Thus optimization is convex, and its global minimum can be

efficiently computed.

The other possible option is to explore local distance combinations (Yang et al., 2006; Domeniconi and Gunopulos, 2002). Local methods form an interesting alternative and in comparison with global approaches were shown to achieve better performance for data exhibiting "difficult" distributions. The adaptation of our framework to local methods will result in an even more flexible method for representation and operator combination on complex domains.

6.1.3 Regularization

The other research direction is to examine other regularization strategies than the ones proposed in Section 5.1.5. More precisely, it would be interesting to examine more aggressive regularization based on the L_1 norm similar to that used in LASSO (Tibshirani, 1996). To the best of our knowledge such regularization has not been considered in the context of metric learning. This regularization technique can be implemented by solving the optimization problem from Equation 5.10 where $\Omega(\boldsymbol{Z}) = \sum_{ij} |Z_{ij}|$. L_1 regularization enjoys several favorable properties compared to L_2 regularization. For example, in the context of logistic regression, it was shown, both theoretically and empirically, to work well in domains with a high fraction of irrelevant features (Ng, 2004). Moreover, the L_1 regularizer usually produces sparse solutions in which most of the model parameters are zero, and hence the resulting models are faster and more interpretable. This sparsity is a consequence of the fact that its first partial derivative with respect to each variable is constant as the variable moves toward zero, "pushing" the value all the way to zero if possible (Andrew and Gao, 2007).

The main problem of this regularization technique comes form the fact that the resulting problem is not differentiable at zero and hence is much more difficult to solve. In particular the objective function can not be optimized with general purpose gradient-based optimization algorithms. Several special-purpose algorithms have been designed to overcome this difficulty (Tibshirani, 1996; Perkins and Theiler, 2003; Lee et al., 2006; Andrew and Gao, 2007), however, it remains to be seen which optimization technique to use in our context.

6.1.4 Learning with Side-Information

So far our work has been focused only on the supervised learning paradigm where the information was provided in the form of equivalence relations as pairwise constraints, namely whether two points are in the same class or not. As a direct extension of this paradigm it is possible to adapt our methods to learn the optimal combination even if we do not have complete information about the class labels, but instead we only have access to some limited knowledge, i.e. side-information, that might be advantageous to exploit by the adaptive methods.

There are three main benefits of using side-information in our context. First, by exploiting various forms of side-information we are able to extend the methods from the simple classification context to other learning paradigms, such as

clustering, information retrieval, regression, etc. Second, unlike labels the various forms of side information can be automatically obtained without the need of human intervention. For example, side-information in the form of relative, qualitative examples (e.g. "A is closer to B than B is to C") was shown by Schultz and Joachims (2004) to be readily available from search-engine query logs; side-information in the form of multiple, absolute, qualitative examples (e.g. "A, B and C are similar") can be automatically obtained from stationary indoor surveillance cameras Bar-Hillel et al. (2005), etc. Finally, even in the supervised setting, we can reduce the computational complexity of the adaptive methods by limiting the size of $\mathcal{S}$ and $\mathcal{D}$. In particular the complexity of all the considered methods with respect to $\mathcal{S}$ and $\mathcal{D}$ was shown in Section 5.1.7 to scale as $O(|\mathcal{S}|^2 + |\mathcal{D}|^2)$, and by reducing the size of $\mathcal{S}$ and $\mathcal{D}$ e.g. by sampling, we can make our methods more efficient.

More precisely, we envisage two aspects of representation and operator learning with side-information which are worth considering. First, it would be interesting to examine how the performance of our system is affected when the size of $\mathcal{S}$ and $\mathcal{D}$ is reduced. As already mentioned, by limiting the size of these sets we are able to reduce the computational complexity of the adaptive methods. One problem with this approach is that for small $\mathcal{S}$ and $\mathcal{D}$ over-fitting might occur, i.e. a particular configuration of small $\mathcal{S}$ and $\mathcal{D}$ is likely to have a big impact on the particular solution $\boldsymbol{A}$ (or (W)). In such cases one should pay attention to regularization of the solutions using the different regularization strategies.

The second research direction is to apply our framework in domains where it is difficult to obtain class labels and which require richer data representation; possible applications are sentence redundancy detection and evaluation of pharmaceutical projects. In the former case the goal is to find structural features of the two sentences which could indicate if these sentences are on the same topic, or whether one of them "subsumes" the other. It is clear that with the standard bag-of-words representations it is not possible to capture such dependencies, and we need richer representation based e.g. on parse or dependency parsing. Different forms of side information can be used within this context, e.g. multiple, absolute, qualitative examples, i.e. "sentences A,B,... are similar". The second structured application for which it is natural to define side-information is the problem of valuating pharmaceutical projects. In this case it is most natural to represent the learning examples as tuples consisting of both primitive attributes and sets; the sets in this composite representation correspond e.g. to the different action mechanisms of a given drug. This problem can be naturally tackled within the framework of clustering with side-information of the form of pairwise, absolute, qualitative examples, i.e. "projects A and B are similar, while C and D are dissimilar".

6.1.5 Adaptive Distances on Sets

The central idea in most of the operators on sets that we have considered in this work, both distance- and kernel-based, is the definition of a family of mappings $\mathcal{F}$, from which a specific mapping, F, of the elements of the sets is selected,

usually via some optimization function. More precisely, the set operator op_{set} between A and B can be written as

$$op_{set}(A, B) = \sum_{(a,b)\in F} op(a, b),\ F \in \mathcal{F} \tag{6.1}$$

where *op* is the operator over the elements of the sets, i.e. distance or kernel.

It would be interesting to explore ways in which we can improve these set operators by making them more flexible. We envisage two research directions. In the first, the goal is to learn and optimize the mapping F with respect to the representation of the elements of the sets. This will result in more optimal solutions than the ones that we get when we are limited in the original fixed representation of the elements of the sets. In the second, and more ambitious direction, the goal is to relax the representation bias of the set operators by removing the constraint that the mapping F should belong to a specific family of mappings $\mathcal{F}$.

The definition of set operator in Equation 6.1 is such that within a given family of mappings $\mathcal{F}$, the actual mapping F of elements depends on the operator *op*, defined on the sets' elements. Changing *op* we can actually change the way the elements of two sets are matched. In what follows we assume that the computation of *op* depends on some parameters $\boldsymbol{P}$ so that we obtain a parameterized operator denoted by $op_{\boldsymbol{P}}$. For example, in the context of distances over sets where elements are vectors, $\boldsymbol{P}$ might take the form of a positive semi-definite matrix such that $d_{\boldsymbol{P}}$ is the Mahalanobis distance defined over the elements of the sets. The set operator can be then presented by

$$op_{set,\boldsymbol{P}}(A, B) = \sum_{(a,b)\in F} op_{\boldsymbol{P}}(a, b),\ F \in \mathcal{F} \tag{6.2}$$

where $F \in \mathcal{F}$ for some family $\mathcal{F}$. We mention that the parameterization of the above Equation has already been exploited in a similar context (graph matching) where it was used to guide the search for optimal matchings within the family of injections (Caetano et al., 2007).

It is also possible to re-parametrize the set operator given in Equation 6.1 so that it is possible to represent the F mappings in a more structured way. The new formulation is based on the matching matrix $\boldsymbol{\Pi}$, where $\boldsymbol{\Pi}_{ij} = 1$ if element i from set A is mapped to elements j in B, and $\boldsymbol{\Pi}_{ij} = 0$ otherwise, for some ordering of the elements of the two sets. Moreover, let $\boldsymbol{O}$ be a matrix which contains the elements from $\{op(a, b)\}$ (indexed in the same order as in $\boldsymbol{\Pi}$). Based on the above definitions the set operator is now given by

$$op_{set,\boldsymbol{\Pi}}(A, B) = \sum_{ij} \boldsymbol{\Pi}_{ij}\boldsymbol{O}_{ij}. \tag{6.3}$$

The matching matrix $\boldsymbol{\Pi}$ can be relaxed by allowing the elements of $\boldsymbol{\Pi}$ to take continuous values e.g. in $[0, 1]$. A continuous $\boldsymbol{\Pi}$ allows fuzzy, partial matchings between elements of sets. As a result, the direct search for the optimal $\boldsymbol{\Pi}$ might

be easier since the mappings are able to improve gradually and continuously during the search, without "jumping around" in the space of binary matching matrices.

Based on the above versions of the set operators, similarly to Equation 5.3, we can define a generic formulation of learning a "good" mapping which consists in finding the optimal parameters $\boldsymbol{Z}$ given by the solution of the following optimization problem

$$\min_{\boldsymbol{Z}} \mathcal{G}(\mathcal{S}, \mathcal{D}, op_{set,\boldsymbol{Z}}) \tag{6.4}$$

where $\boldsymbol{Z}$ is either $\boldsymbol{P}$ or $\boldsymbol{\Pi}$, $\mathcal{G}$ is an objective function that needs to be minimized with respect to the elements of $\boldsymbol{Z}$. In general different choices of $\mathcal{G}$ will give rise to different mappings; when the cost functions are differentiable a standard gradient-based optimization method can be used to find the optimal solution.

It is interesting to note that the optimization problem of Equation 6.4 generalizes the problem of learning mappings when that is tackled in the framework of structural supervised learning (Caetano et al., 2007; Taskar, 2004; Tsochantaridis et al., 2005). The main problem with this approach is that it assumes that pairs of sets are provided together with the "true" mappings, which are usually difficult to obtain. As a consequence, it is more advantageous in practice to exploit other objective functions $\mathcal{G}$ which do not depend on training data being annotated with mappings. The obvious choices are the cost functions that we have already explored in the context of representation or distance combination in Chapter 5.

There are several challenges related to applications of the above cost functions for the two proposed parameterizations, and the way the resulting optimization problems are solved. For example, the main difficulty in the optimization of $\boldsymbol{P}$ comes from the fact that this has to be done with respect to a particular mapping family $\mathcal{F}$. While it is easy to do it for the mapping F, that contains all the matchings of elements, it is not obvious how to do it for a general family of mappings. The main difficulty comes from the fact that in the general case the cost function $\mathcal{G}$ depends on the parameters $\boldsymbol{P}$ in a complicated way, resulting in a very difficult (in particular non-convex or possibly even non-differentiable) optimization problem.

There are two main problems related to learning the operators given in Equation (6.3). The first problem is that of ordering the elements of the sets, and the second is the varying cardinality of the sets. If we do not address these problems, the direct optimization of the mapping $\boldsymbol{\Pi}$ is not feasible since the corresponding search space will lack structure. In particular, for any pair of sets the matching matrix $\boldsymbol{\Pi}$ depends on the permutation, i.e. order, of elements that we consider within the sets. Moreover, the dimensions of the matching matrices $\boldsymbol{\Pi}$ vary according to the cardinalities of the sets involved. In order to learn optimal $\boldsymbol{\Pi}$ it is hence necessary to define an appropriate coordinate system in which we can represent different sets in a unique, meaningful, and unambiguous manner.

6.1.6 Theoretical Analysis of Mapping-Based Set Kernels

Finally, it is interesting to focus on set kernels based on mappings from Equation 6.1 and analyze the conditions the mapping family $\mathcal{F}$ between two sets should fulfill so that the resulting kernel is positive semi-definite. To the best of our knowledge there is only one family of mappings defined in the literature which gives rise to valid kernels, i.e. by setting $F = A \times B,\ F \in \mathcal{F}$ we obtain the well known cross product kernel (Section 4.2.3). Other mapping-based kernels have been defined in the literature (Fröhlich et al., 2005; Wallraven et al., 2003; Boughorbel et al., 2004), however, they have been shown not to be positive semi-definite in general. In particular, it was recently shown in (Vert, 2008) that the optimal assignment kernel from (Fröhlich et al., 2005), which was claimed to be valid, is not positive semi-definite in general. Moreover, all the set kernels from Section 4.2.3 which are directly based on mappings have been verified not to be valid. An obvious question is what are the conditions on $\mathcal{F}$ for the kernel to be positive semi-definite. It is our feeling that only $F \subseteq A \times B,\ F \in \mathcal{F}$ gives rise to a valid kernel; however, this statement needs to be verified theoretically.

Bibliography

David W. Aha. Editorial. *Artificial Intelligence Review*, 11:1–6, 1997. Special Issue on Lazy Learning.

David W. Aha, Dennis Kibler, and Marc K. Albert. Instance-based learning algorithms. *Mach. Learn.*, 6(1):37–66, 1991. ISSN 0885-6125.

Galen Andrew and Jianfeng Gao. Scalable training of l1-regularized log-linear models. In *ICML '07: Proceedings of the 24th International Conference on Machine Learning*, pages 33–40, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-793-3.

Stuart Andrews, Ioannis Tsochantaridis, and Thomas Hofmann. Support vector machines for multiple-instance learning. In *NIPS*, pages 561–568, 2002.

Cosimo Anglano, Attilio Giordana, Giuseppe Bello, and Lorenza Saitta. An experimental evaluation of coevolutive concept learning. In Jude W. Shavlik, editor, *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 19–27, 1998.

Andreas Argyriou, Charles A. Micchelli, and Massimiliano Pontil. Learning convex combinations of continuously parameterized basic kernels. In *COLT 2005*, Bertinoro, Italy, 2005.

M. J. Atallah. A linear time algorithm for the hausdorff distance between convex polygons. *Information Processing Letters*, 17:207–209, 1983.

T. K. Attwood, P. Bradley, D. R. Flower, A. Gaulton, N. Maudling, A. L. Mitchell, G. Moulton, A. Nordle, K. Paine, P. Taylor, A. Uddin, and C. Zygouri. PRINTS and its automatic supplement, prePRINTS. *Nucleic Acids Research*, 31(1):400–402, 2003.

Mordecai Avriel. *Nonlinear Programming: Analysis and Methods*. Dover Publications, 2003.

Claus Bahlmann, Bernard Haasdonk, and Hans Burkhardt. On-line handwriting recognition with support vector machines—a kernel approach. In *Proc. 8th Int. Workshop Front. Handwriting Recognition (IWFHR)*, pages 49–54, 2002.

Aharon Bar-Hillel, Tomer Hertz, Noam Shental, and Daphna Weinshall. Learning a mahalanobis metric from equivalence constraints. *Journal of Machine Learning Research*, 6:937–965, 2005.

Etienne Baudrier, Gilles Millon, Frederic Nicolier, and Su Ruan. A new similarity measure using hausdorff distance map. In *In Proc of international conference on image processing (ICIP)*, pages 669–672, 2004.

M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 2003.

S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(4):509–522, 2002a. ISSN 0162-8828.

Serge Belongie, Charless Fowlkes, Fan Chung, and Jitendra Malik. Spectral partitioning with indefinite kernels using the nystrom extension. In *ECCV '02: Proceedings of the 7th European Conference on Computer Vision-Part III*, pages 531–542, London, UK, 2002b. Springer-Verlag. ISBN 3-540-43746-0.

S. Ben-David, N. Eiron, and H.U. Simon. Limitations of learning via embeddings in euclidean half spaces. *Journal of Machine Learning Research*, 3:441–461, 2002.

Charles H. Bennett, Peter Gacs, Ming Li, Paul M. B. Vitanyi, and Wojciech H. Zurek. Information distance. *IEEETIT: IEEE Transactions on Information Theory*, 44, 1998.

C. Berg, J. P. R. Christensen, and P. Ressel. *Harmonic Analysis on Semigroups*. Springer, Berlin, 1984.

Philip Bille. A survey on tree edit distance and related problems. *Theoretical Computer Science (TCS)*, pages 217–239, 2005.

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

Gilles Bisson. Conceptual clustering with a similarity measure. In Bernd Neumann, editor, *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 458–462, 1992.

Hendrik Blockeel. *Top-down induction of logical decision trees*. PhD thesis, Department of Computer Science, K.U.Leuven, 1998. Chapter 4, First Order Logic Representations.

Hendrik Blockeel and Luc De Raedt. Top-down induction of logical decision trees. Technical report, Department of Computer Science, K.U.Leuven, 1997.

Karsten M. Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *Proceedings of the 5th IEE Internatinal Conference on Data Mining*, 2005.

Karsten M. Borgwardt, Cheng Soon Ong, Stefan Schönauer, S.V.N Vishwanathan, Alex J. Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(1):i47–i56, 2005.

S. Boughorbel, J.P. Tarel, and F. Fleuret. Non-mercer kernels for svm object recognition. In *BMVC04*, pages xx–yy, 2004.

S. Boughorbel, J.-P. Tarel, and N. Boujemaa. The intermediate matching kernel for image local features. In *Proceedings of International Joint Conference on Neural Networks (IJCNN'05)*, pages 889 – 894, Montréal, Canada, 2005.

O Bousquet and D. Herrmann. On the complexity of learning the kernel matrix. In *Advances in Neural Information Processing Systems 14*, Cambridge, MA, 2003. MIT Press.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, March 2004. ISBN 0521833787.

Björn Bringmann, Albrecht Zimmermann, Luc De Raedt, and Siegfried Nijssen. Don't be afraid of simpler patterns. In *PKDD*, pages 55–66, 2006.

C. Caetano, L. Cheng, Q.V. Le, and A.J. Smola. Learning graph matching. In *Proceedings of the 11th IEEE International Conference on Computer Vision, 2007*, 2007.

Michael Campa, Michael Fitzgerald, and Edward Patz. Editorial: Proteomics 9/2003. *Proteomics*, 9, 2003.

R. G. Cattell, Douglas K. Barry, Mark Berler, Jeff Eastman, David Jordan, Craig Russell, Olaf Schadow, Torsten Stanienda, and Fernando Velez, editors. *The Object Data Standard: ODMG 3.0*. Morgan Kaufmann, 2000.

Deepayan Chakrabarti and Christos Faloutsos. Graph mining: Laws, generators, and algorithms. *ACM Comput. Surv.*, 38(1), 2006. ISSN 0360-0300. doi: 10.1145/1132952.1132954.

O. Chapelle and A. Zien. Semi-supervised classification by low density separation. In *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics (AI & Statistics 2005)*, pages 57–64, 01 2005.

Olivier Chapelle, Patrick Haffner, and Vladimir Vapnik. Svms for histogram-based image classification. *IEEE Transactions on Neural Networks, special issue on Support Vectors*, pages 1055–1064, September 1999.

Olivier Chapelle, Vladimir Vapnik, Olivier Bousquet, and Sayan Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46(1-3):131–159, 2002. ISSN 0885-6125.

Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien. *Semi-Supervised Learning*. The MIT Press, September 2006. ISBN 0262033585.

Jiun-Hung Chen. M-estimator based robust kernels for support vector machines. In *Proceedings of the 17th International Conference on Pattern Recognition*, 2004.

Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *Proc. of Computer Vision and Pattern Recognition Conference*. IEEE Press, 2005.

Rudi L. Cilibrasi and Paul M. B. Vitányi. Clustering by compression. *Information Theory, IEEE Transactions on*, 51(4):1523–1545, 2005.

M. Collins and N. Duffy. Convolution kernels for natural language. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, Cambridge, MA, 2002. MIT Press.

P. Comon. Independent component analysis, a new concept ? *Signal Processing*, pages 287–314, 1994.

T. Cox and M. Cox. *Multidimensional Scaling*. Distance metric learning: A comprehensive survey, 1994.

Koby Crammer, Joseph Keshet, and Yoram Singer. Kernel design using boosting. In *Advances in Neural Information Processing Systems 14*, Cambridge, MA, 2002. MIT Press.

N. Cristianini, J. Shawe-Taylor, A. Elisseeff, and J. Kandola. On kernel-target alignment. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, Cambridge, MA, 2002. MIT Press.

Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, March 2000. ISBN 0521780195.

Nello Cristianini and John Shawe-Taylor. Support vector and kernel methods. In Michael Berthold and David J. Hand, editors, *Intelligent Data Analysis*. Springer, 2003.

Aron Culotta and Jeffery Sorensen. Dependency tree kernels for relation extraction. In *ACL*, Barcelona, Spain, 2004.

Chad Cumby and Dan Roth. On kernel methods for relational learning. In *ICML '03: Proceedings of the 20th International Conference on Machine Learning*, Washington, DC, 2003.

Marco Cuturi, Kenji Fukumizu, and Jean-Philippe Vert. Semigroup kernels on measures. *Journal of Machine Learning Research*, 6:1169–1198, 2005.

Mukund Deshpande, Michihiro Kuramochi, and George Karypis. Frequent substructure-based approaches for classifying chemical compounds. In *ICDM '03: Proceedings of the Third IEEE International Conference on Data Mining*, page 35, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1978-4.

F. Desobry, M. Davy, and W.J. Fitzgerald. A class of kernels for sets of vectors. In *ESANN 2005*, 2005.

Reinhard Diestel. *Graph Theory (Third Edition)*. Springer-Verlag, 2005.

Thomas G. Dietterich, Richard H. Lathrop, and Tomas Lozano-Perez. Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1-2):31–71, 1997.

Carlotta Domeniconi and Dimitrios Gunopulos. Adaptive nearest neighbor classification using support vector machines. In *NIPS 14*. MIT Press, Cambridge, MA, 2002.

K. Duan, S.S. Keerthi, and A.N. Poo. Evaluation of simple performance measures for tuning svm hyperparameters. *Neurocomputing*, 51:41–59, 2003.

Marie-Pierre Dubuisson and Anil K. Jain. A modified hausdorff distance for object matching. In *International Conference on Pattern Recognition*, 1994.

Richard Duda, Peter Hart, and D. Stork. *Pattern Classification and Scene Analysis*. John Willey and Sons, 2001.

Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons Inc, 1973.

James Dugundji. *Topology*. Allyn and Bacon, 1966.

Robert P.W. Duin, Elżbieta Pekalska, and Dick de Ridder. Relational dicriminant analysis. *Pattern Recongnition Letters*, 20:1175–1181, 1999.

Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme Mitchison. *Biological Sequence Analysis : Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, July 1999. ISBN 0521629713.

Sašo Džeroski. Towards a general framework for data mining. In Saso Dzeroski and Jan Struyf, editors, *KDID*, volume 4747 of *Lecture Notes in Computer Science*, pages 259–300. Springer, 2007. ISBN 978-3-540-75548-7.

Sašo Džeroski and Nada Lavrac. *Relational Data Mining*. Springer-Verlag, Berlin, September 2001. ISBN 3-540-42289-7.

Sašo Džeroski, Steffen Schulze-Kremer, Karsten R. Heidtke, Karsten Siems, and Dietrich Wettschereck. Applying ILP to diterpene structure elucidation from 13 c NMR spectra. In *Inductive Logic Programming Workshop*, pages 41–54, 1996.

T Eiter and H. Mannila. Distance measures for point sets and their computation. *Acta Informatica*, 34(2):109–133, 1997.

W. Emde and D. Wettschereck. Relational instance-based learning. In L. Saitta, editor, *ML96*, pages 122–130. MK, 1996.

Peter A. Flach, Christophe G. Giraud-Carrier, and John W. Lloyd. Strongly typed inductive concept learning. In *ILP '98: Proceedings of the 8th International Workshop on Inductive Logic Programming*, pages 185–194, London, UK, 1998. Springer-Verlag. ISBN 3-540-64738-4.

J. H. Friedman. Another approach to polychotomous classification. Technical report, Department of Statistics, Stanford University, 1996.

Holger Fröhlich, Jörg K. Wegner, Florian Sieker, and Andreas Zell. Optimal assignment kernels for attributed molecular graphs. In *ICML '05: Proceedings of the 22nd International Conference on Machine Learning*, pages 225–232, New York, NY, USA, 2005. ACM. ISBN 1-59593-180-5.

Thomas Gärtner. Exponential and geometric kernels for graphs. In *NIPS*02 Workshop on Unreal Data: Principles of Modeling Nonvectorial Data*, 2002.

Thomas Gärtner. A survey of kernels for structured data. *SIGKDD Explor. Newsl.*, 5(1):49–58, 2003. ISSN 1931-0145.

Thomas Gärtner, Peter A. Flach, Adam Kowalczyk, and Alex J. Smola. Multi-instance kernels. In Claude Sammut and Achim Hoffmann, editors, *Proceedings of the 19th International Conference on Machine Learning*, pages 179–186. Morgan Kaufmann, July 2002. ISBN 1-55860-873-7.

Thomas Gärtner, Peter Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Proceedings of the 16th Annual Conference on Computational Learning Theory and the 7th Kernel Workshop*, 2003.

Thomas Gärtner, John W. Lloyd, and Peter A. Flach. Kernels and distances for structured data. *Machine Learning*, 57(3):205–232, 2004. ISSN 0885-6125.

Thomas Gärtner, Tamás Horváth, Quoc V. Le, Alex J. Smola, and Stefan Wrobel. Kernel methods for graphs. In *Mining Graph Data*. John Wiley & Sons, 2007.

Lise Getoor and Ben Taskar, editors. *Introduction to Statistical Relational Learning*. The MIT Press, 2007.

Amir Globerson and Sam Roweis. Metric learning by collapsing classes. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *NIPS 18*, pages 451–458. MIT Press, Cambridge, MA, 2006.

Jacob Goldberger, Sam Roweis, Geoff Hinton, and Ruslan Salakhutdinov. Neighbourhood component analysis. In *NIPS*. MIT Press, Cambridge, MA, 2005.

Thore Graepel, Ralf Herbrich, Peter Bollmann-Sdorra, , and Klaus Obermayer. Classification on pairwise proximity data. In *In Advances in Neural Information Processing Systems 11*, pages 438–444, 1999.

Kristen Grauman and Trevor Darrell. The pyramid match kernel: Efficient learning with sets of features. *J. Mach. Learn. Res.*, 8:725–760, 2007. ISSN 1533-7928.

Dan Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology.* Cambridge University Press, 1997.

B. Haasdonk. *Transformation Knowledge in Pattern Analysis with Kernel Methods.* PhD thesis, University of Freiburg, May 2005a.

Bernard Haasdonk. Feature space interpretation of svms with indefinite kernels. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(4):482–492, April 2005b.

Bernard Haasdonk and Claus Bahlmann. Learning with distance substitution kernels. In *26th Pattern Recognition Symposium of the German Association for Pattern Recognition (DAGM 2004)*, Tübingen, Germany, 2004. Springer Verlag.

Bernard Haasdonk and Daniel Keysers. Tangent distance kernels for support vector machines. In *16th ICPR*, volume 2, pages 864–868. Springer Verlag, 2002.

M. A. Hall. *Correlation-based Feature Subset Selection for Machine Learning.* PhD thesis, University of Waikato, Hamilton, New Zealand, 1998.

T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning.* Springer, August 2001. ISBN 0387952845.

Trevor Hastie and Robert Tibshirani. Discriminant adaptive nearest neighbor classification and regression. In David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, editors, *NIPS 8*, pages 409–415. The MIT Press, 1996.

D. Haussler. Convolution kernels on discrete structures. Technical report, UC Santa Cruz, 1999.

M. Hein and O. Bousquet. Hilbertian metrics and positive definite kernels on probability measures. In *Proceedings of AISTATS 2005*, 2005.

Matthias Hein, Thomas Navin Lal, and Olivier Bousquet. Hilbertian metrics on probability measures and their application in svm?s. In Carl Edward Rasmussen, Heinrich H. Bülthoff, Bernhard Schölkopf, and Martin A. Giese, editors, *DAGM-Symposium*, volume 3175 of *Lecture Notes in Computer Science*, pages 270–277. Springer, 2004. ISBN 3-540-22945-0.

C. Helma, R. D. King, S. Kramer, and A. Srinivasan. The predictive toxicology challenge 2000–2001. *Bioinformatics*, 17:107–108, 2001.

Tomer Hertz, Aharon Bar-Hillel, and Daphna Weinshall. Boosting margin based distance functions for clustering. In *ICML '04: Proceedings of the 21st International Conference on Machine Learning*, page 50, New York, NY, USA, 2004. ACM Press. ISBN 1-58113-828-5.

M. Hilario, A. Mitchell, J. H. Kim, P. Bradley, and T. Attwood. Classifying protein fingerprints. In *Proc. 8th Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD-04 Best Paper Award)*, Pisa, Italy, September 2004. Springer-Verlag.

S. C. H. Hoi, W. Liu, M. R. Lyu, and W.-Y. Ma. Learning distance metrics with contextual constraints for image retrieval. In *Computer Vision and Pattern Recognition*, 2006.

Tamás Horváth. Cyclic pattern kernels revised. In *The 9th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Hanoi, Vietnam, 2005.

Tamás Horváth, Stefan Wrobel, and Uta Bohnebeck. Relational instance-based learning with lists and terms. *Machine Learning*, 43(1/2):53–80, 2001.

Tamás Horváth, Thomas Gärtner, and Stefan Wrobel. Cyclic pattern kernels for predictive graph mining. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 158–167, New York, NY, USA, 2004. ACM Press. ISBN 1-58113-888-1.

Tamás Horváth, Björn Bringmann, and Luc De Raedt. Frequent hypergraph mining. In *ILP*, pages 244–259, 2006.

Tamás Horváth, Jan Ramon, and Stefan Wrobel. Frequent subgraph mining in outerplanar graphs. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 197–206, New York, NY, USA, 2006. ACM Press. ISBN 1-59593-339-5.

Alan Hutchinson. Metrics on terms and clauses. In Maarten van Someren and Gerhard Widmer, editors, *Proceedings of the Ninth European Conference on Machine Learning*, volume distance, pages 138–145, 1997.

Daniel P. Huttenlocher, Gregory A. Klanderman, and William J. Rucklidge. Comparing images using the hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15:850–863, 1993.

T. Jaakkola and D. Haussler. Exploiting generative models in discriminative classifiers. In *Advances in Neural Information Processing Systems 11*, pages 487–493, 1999.

Tatsuya Akutsu Jean-Philippe Vert, Hiroto Saigo. Local alignment kernels for biological sequences. In Koji Tsuda Bernhard Schölkopf and Jean-Philippe Vert, editors, *Kernel Methods in Computational Biology*, pages 131–153. MIT Press, 2004.

Tony Jebara and Risi Kondor. Bhattacharyya and expected likelihood kernels. In B. Schölkopf and M. Warmuth, editors, *16th Annual Conference on Learning Theory (COLT) and 7th Annual Workshop on Kernel Machines, Proceedings*, Lecture Notes in Artificial Intelligence. Springer Verlag, 2003.

Tony Jebara, Risi Kondor, and Andrew Howard. Probability product kernels. *J. Mach. Learn. Res.*, 5:819–844, 2004. ISSN 1533-7928.

Oliver Jesorsky, Klaus J. Kirchberg, and Robert Frischholz. Robust face detection using the hausdorff distance. In *AVBPA '01: Proceedings of the Third International Conference on Audio- and Video-Based Biometric Person Authentication*, pages 90–95, London, UK, 2001. Springer-Verlag. ISBN 3-540-42216-1.

Thorsten Joachims. *Learning to Classify Text using Support Vector Machines. Methods, Theory, and Algorithms.* Kluwer Academic Publishers / Springer, 2002.

A. Kalousis and T. Theoharis. Noemon: Design, implementation and performance results for an intelligent assistant for classifier selection. *Intelligent Data Analysis Journal*, 3:319–337, 1999.

A. Kalousis, A. Woznica, and M. Hilario. A unifying framework for relational distance-based learning. Technical report, University of Geneva, 2005.

H. Kashima and T. Koyanagi. Kernels for semi-structured data. In *ICML 2002*, 2002.

Hisashi Kashima, Koji Tsuda, and Akihiro Inokuchi. Marginalized kernels between labeled graphs. In Tom Fawcett and Nina Mishra, editors, *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*, pages 321–328. AAAI Press, 2003.

Eamonn Keogh, Stefano Lonardi, and Chotirat Ann Ratanamahatana. Towards parameter-free data mining. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 206–215, New York, NY, USA, 2004. ACM. ISBN 1-58113-888-1.

Won Kim. *Introduction to Object-Oriented Databases.* The MIT press, 1990.

Mathias Kirsten, Stephan Wrobel, and Tamas Horvath. *Relational Data Mining*, chapter Distance Based Approaches to Relational Learning and Clustering, pages 212–232. Springer, 2001.

Erwin Klein and Anthony C. Thompson. *Theory of Correspondences.* John Wiley & Sons Inc, 1984.

Philip N. Klein. Computing the edit-distance between unrooted ordered trees. In *Proceeding of 6th European Symposium on Algorithms*, pages 91–102, 1998.

R. Kondor and T. Jebara. A kernel between sets of vectors. In *ICML '03: Proceedings of the 20th International Conference on Machine Learning*, Washington, DC, 2003.

S. Kramer, L.D Readt, and C. Helma. Molecular feature mining in hiv data. In *Proceedings of KDD'01*, 2001.

Taku Kudo, Eisaku Maeda, and Yuji Matsumoto. An application of boosting to graph classification. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 729–736, Cambridge, MA, 2005. MIT Press.

J.T. Kwok and I.W. Tsang. Learning with idealized kernels. In *ICML '03: Proceedings of the 20th International Conference on Machine Learning*, 2003.

J. Lafferty and G. Lebanon. Information diffusion kernels. In Suzanna Becker, Sebastian Thrun, and Klaus Obermayer, editors, *NIPS*, 2002.

G. Lanckriet, N. Cristianini, P. L. Bartlett, L. El Ghaoui, and M. Jordan. Learning the kernel matrix with semi-definite programming. *Journal of Machine Learning Research*, 5:27–72, 2004.

Su-In Lee, Honglak Lee, Pieter Abbeel, and Andrew Y. Ng. Efficient l1 regularized logistic regression. In *AAAI*. AAAI Press, 2006.

Christina Leslie, Rui Kuang, and Eleazar Eskin. Inexact matching string kernels for protein classification. In *Kernel Methods in Computational Biology*, MIT Press series on Computational Molecular Biology, pages 95–111. MIT Press, 2003.

V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10:707–710, 1966.

Ming Li, Xin Chen, Xin Li, Bin Ma, and Paul Vitányi. The similarity metric. In *SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 863–872, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics. ISBN 0-89871-538-5.

Li Liao and William Stafford Noble. Combining pairwise sequence similarity and support vector machines for remote protein homology detection. In *RECOMB '02: Proceedings of the sixth annual international conference on Computational biology*, pages 225–232, New York, NY, USA, 2002. ACM. ISBN 1-58113-498-3.

Hsuan-Tien Lin and Chih-Jen Lin. A study on sigmoid kernels for svm and the training of non-psd kernels by smo-type methods. Technical report, National Taiwan University, March 2003.

J.W. Lloyd. *Logic for Learning Learning Comprehensible Theories from Structured Data*. Springer, July 2003.

H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, 2: 419–444, 2002.

Huma Lodhi and Stephen Muggleton. Is mutagenesis still challenging? In *Proceedings of the 15th International Conference on Inductive Logic Programming (ILP-2005) (late breaking papers)*, 2005.

S. Lyu. Kernels for unordered sets: the gaussian mixture approach. In *Proceedings of European Conference on Machine Learning (ECML)*, Porto, Portugal, 2005a.

Siwei Lyu. Mercer Kernels for Object Recognition with Local Features. Technical Report TR2004-520, Dartmouth College, Computer Science, Hanover, NH, October 2004.

Siwei Lyu. Mercer kernels for object recognition with local features. In *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2*, pages 223–229, Washington, DC, USA, 2005b. IEEE Computer Society. ISBN 0-7695-2372-2.

P. Mahé, L. Ralaivola, V. Stoven, and J.-P. Vert. The pharmacophore kernel for virtual screening with support vector machines. *Journal of Chemical Information and Modeling*, 46:2003–2014, 2006.

Pierre Mahé, Nobuhisa Ueda, Tatsuya Akutsu, Jean-Luc Perret, and Jean-Philippe Vert. Extensions of marginalized graph kernels. In *ICML '04: Proceedings of the 21st International Conference on Machine Learning*, page 70, New York, NY, USA, 2004. ACM. ISBN 1-58113-828-5.

Kantilal Vardichand Mardia, John T. Kent, and John Bibby. *Multivariate Analysis*. Academic Press, 1979.

Oded Maron and Tomás Lozano-Pérez. A framework for multiple-instance learning. In *NIPS '97: Proceedings of the 1997 conference on Advances in neural information processing systems 10*, pages 570–576, Cambridge, MA, USA, 1998. MIT Press. ISBN 0-262-10076-2.

Q. McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12:153–157, 1947.

Sauro Menchetti, Fabrizio Costa, and Paolo Frasconi. Weighted decomposition kernels. In *ICML '05: Proceedings of the 22nd International Conference on Machine Learning*, pages 585–592, New York, NY, USA, 2005. ACM Press. ISBN 1-59593-180-5.

Pedro J. Moreno, Purdy P. Ho, and Nuno Vasconcelos. A kullback-leibler divergence based kernel for svm classification in multimedia applications. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

Stephen Muggleton. Inverse entailment and progol. *New Generation Computing*, 13:245–286, 1995.

Andrew Y. Ng. Feature selection, l1 vs. l2 regularization, and rotational invariance. In *ICML '04: Proceedings of the 21st International Conference on Machine Learning*, New York, NY, USA, 2004. ACM Press.

Shan-Hwei Nienhuys-Cheng. Distances between herbrand interpretations: a measure for aproximations to a target concept. In Saso Dzeroski and Nada Lavrac, editors, *Inductive Logic Programming, Seventh International Workshop*, pages 213–226, 1997.

Cheng Soon Ong, Xavier Mary, Stéphane Canu, and Alexander J. Smola. Learning with non-positive kernels. In *ICML '04: Proceedings of the 21st International Conference on Machine Learning*, page 81, New York, NY, USA, 2004. ACM Press. ISBN 1-58113-828-5.

Cheng Soon Ong, Alexander J. Smola, and Robert C. Williamson. Learning the kernel with hyperkernels. *Journal of Machine Learning Research*, 6:1043–1071, 2005.

Andrea Passerini, Paolo Frasconi, and Luc De Raedt. Kernels on prolog proof trees: Statistical learning in the ilp settings. *Journal of Machine Learning Research*, pages 307–342, 2006.

Elzbieta Pekalska and Robert P. W. Duin. *The Dissimilarity Representation for Pattern Recognition: Foundations And Applications (Machine Perception and Artificial Intelligence)*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2005. ISBN 9812565302.

Elżbieta Pekalska, Pavel Paclík, and Robert P.W. Duin. A generalized kernel approach to dissimilarity-based classification. *Journal of Machine Learning Research*, 2:175–211, 2001.

Elżbieta Pekalska, Robert P.W. Duin, and Pavel Paclík. Prototype selection for dissimilarity-based classifiers. *Pattern Recognition*, 39:189–208, 2006.

Simon Perkins and James Theiler. Online feature selection using grafting. In *ICML '03: Proceedings of the 20th International Conference on Machine Learning*, 2003.

Julien Prados, Alexandros Kalousis, Jean-Charles Sanchez, Laure Allard, Odile Carrette, and Melanie Hilario. Mining mass-spectra for diagnosis and biomarker discovery of cerebral accidents. *Proteomics*, 4:2320–2332, 2004.

Luc De Raedt. Logical settings for concept-learning. *Artificial Intelligence*, 95: 187–201, 1997.

Liva Ralaivola, Sanjay J. Swamidass, Hiroto Saigo, and Pierre Baldi. Graph kernels for chemical informatics. *Neural Networks*, pages 1093–1110, 2005.

Jan Ramon. *Clustering and instance based learning in first order logic*. PhD thesis, Department of Computer Science, K.U.Leuven, Leuven, 2002.

Jan Ramon and Maurice Bruynooghe. A framework for defining distanes between first-order logic objects. In David Page, editor, *Inductive Logic Programming, Eight International Workshop*, pages 271–280, 1998.

Jan Ramon and Maurice Bruynooghe. A polynomial time computable metric between point sets. *Acta Informatica*, 37(10):765–780, 2001.

Jan Ramon and Thomas Gärtner. Expressivity versus efficiency of graph kernels. In *First International Workshop on Mining Graphs, Trees and Sequences (help with ECML/PKDD'03)*, 2003.

S. Roweis and L. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 2000.

Ulrich Rückert and Stefan Kramer. Frequent free tree discovery in graph data. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 564–570, New York, NY, USA, 2004. ACM. ISBN 1-58113-812-1. doi: http://doi.acm.org/10.1145/967900.968018.

Ulrich Rückert and Stefan Kramer. Optimizing feature sets for structured data. In *Machine Learning: ECML 2007, 18th European Conference on Machine Learning*, 2007.

Hiroto Saigo, Jean-Philippe Vert, Nobuhisa Ueda, and Tatsuya Akutsu. Protein homology detection using string alignment kernels. *Bioinformatics*, 2004.

L. K. Saul, K. Q. Weinberger, J. H. Ham, F. Sha, and D. D. Lee. Spectral methods for dimensionality reduction. In *Semisupervised Learning*. MIT Press: Cambridge, MA, 2006.

C. Schaffer. Selecting a classification method by cross validation. *Machine Learning*, 13:135–143, 1993.

Cullen Schaffer. A conservation law for generalization performance. In *ICML '94: Proceedings of the 11th International Conference on Machine Learning*, pages 259–265, 1994.

Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001.

Bernhard Schölkopf, Jason Weston, Eleazar Eskin, Christina Leslie, and William Stafford Noble. A kernel approach for learning from almost orthogonal patterns. In *ECML '02: Proceedings of the 13th European Conference on Machine Learning*, pages 511–528, London, UK, 2002. Springer-Verlag. ISBN 3-540-44036-4.

Matthew Schultz and Thorsten Joachims. Learning a distance metric from relative comparisons. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

Michele Sebag. Distance induction in first order logic. In *Proceedings of the seventh Inductive Logic Programming Workhsop*, pages 264–272, 1997.

Shai Shalev-Shwartz, Yoram Singer, and Andrew Y. Ng. Online and batch learning of pseudo-metrics. In *ICML '04: Proceedings of the 21st International Conference on Machine Learning*, page 94, New York, NY, USA, 2004. ACM Press. ISBN 1-58113-828-5.

Amnon Shashua and Tamir Hazan. Algebraic set kernels with application to inference over local image representations. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 1257–1264, Cambridge, MA, 2005. MIT Press.

John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, New York, NY, USA, 2004. ISBN 0521813972.

A. Srinivasan, S. Muggleton, R.D. King, and M.J.E. Sternberg. Mutagenesis: ILP experiments in a non-determinate biological domain. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237, pages 217–232, 1994.

Graham A Stephen. *String Searching Algorithms*. World Scientific Publishing Company, 1994.

Michael Stonebraker. *Object-Relational DBMS - The Next Wave*. Morgan Kaufmann Pub, 1996.

M. Sugiyama. Local fisher discriminant analysis for supervised dimensionality reduction. In *ICML '06: Proceedings of the 23th International Conference on Machine Learning*, 2006.

I.J. Taneja. On generalized information measures and their applications. In P.W. Hawkes, editor, *Advances in Electronics and Electron Physics*, pages 327–413. Academic Press, 1989.

Ben Taskar. *Learning Structured Prediction Models: A Large Margin Approach*. PhD thesis, Stanford University, 2004.

Nikolaj Tatti. Distances between data sets based on summary statistics. *Journal of Machine Learning Research*, 8:131–154, 2007.

J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, pages 2319–2323, 2000.

R. Tibshirani. Regression shrinkage and selection via the lasso. *J. Royal. Statist. Soc B.*, 58:267–288, 1996.

G. T. Toussaint and B. K. Bhattacharya. Optimal algorithms for computing the minimum distance between two finite planar sets. *Pattern Recognition Letters*, 1983:79–82, 1983.

Ivor W. Tsang, Pak-Ming Cheung, and James T. Kwok. Kernel relevant component analysis for distance metric learning. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN'05)*, 2005.

Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *J. Mach. Learn. Res.*, 6:1453–1484, 2005. ISSN 1533-7928.

K. Tsuda, T. Kin, and K. Asai. Marginalized kernels for biological sequences. *Bioinformatics*, 2002.

Koji Tsuda. Support vector classifier with asymmetric kernel functions. In *ESANN'1999: European Symposium on Artificial Neural Networks*, pages 183–188, 1999.

Jeffrey Ullman. *Principles of databse systems*. Computer Science Press, 1982.

V. Vapnik. *The nature of statistical learning theory*. Statistics for Engineering and Information Science. Springer Verlag, Berlin, 2000.

Jean-Philippe Vert. The optimal assignment kernel is not positive definite, 2008. URL `http://www.citebase.org/abstract?id=oai:arXiv.org:0801.4061`.

K. Tsuda Vert, J.P. and B. Schölkopf. A primer on kernel methods. In K. Tsuda Schölkopf, B. and J.P. Vert, editors, *Kernel Methods in Computational Biology*, pages 35–70. MIT Press, Cambridge, MA, USA, 2004.

S.V. N. Vishwanathan, Karsten Borgwardt, and Nicol N. Schraudolph. Fast computation of graph kernels. In Bernhard Schölkopf, John C. Platt, and Thomas Hofmann, editors, *NIPS 2007*, volume 19, Cambridge, MA, to appear 2007. MIT Press.

S.V.N. Viswanathan and Alexander J. Smola. Fast kernels for string and tree matching. In S. Thrun S. Becker and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 69–576, Cambridge, MA, 2003. MIT Press.

Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, 1974. ISSN 0004-5411.

Christian Wallraven, Barbara Caputo, and Arnulf Graf. Recognition with local features: the kernel recipe. In *ICCV '03: Proceedings of the Ninth IEEE International Conference on Computer Vision*, page 257, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1950-4.

Takashi Washio and Hiroshi Motoda. State of the art of graph-based data mining. *SIGKDD Explor. Newsl.*, 5(1):59–68, 2003. ISSN 1931-0145.

Larry Wasserman. *All of Statistics: A Concise Course in Statistical Inference*. Springer, 2004.

Kilian Weinberger, John Blitzer, and Lawrence Saul. Distance metric learning for large margin nearest neighbor classification. In *Advances in Neural Information Processing Systems 18*, Cambridge, MA, 2006. MIT Press.

J. Weston and C. Watkins. Support vector machines for multiclass pattern recognition. In *Proceedings of the Seventh European Symposium On Artificial Neural Networks*, 4 1999.

Dietrich Wettschereck, David W. Aha, and Takao Mohri. A review and empirical evaluation of feature weighting methods for aclass of lazy learning algorithms. *Artif. Intell. Rev.*, 11(1-5):273–314, 1997. ISSN 0269-2821.

D. Randall Wilson and Tony R. Martinez. Reduction techniques for instance-based learning algorithms. *Machine Learning*, 38(3):257–286, 2000.

Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 2005.

Lior Wolf and Amnon Shashua. Learning over sets using kernel principal angles. *Journal of Machine Learning Research*, 4:913–931, October 2003.

DH Wolpert. The supervised learning no-free-lunch theorems. In *Proc. 6th Online World Conference on Soft Computing*, 2001.

Adam Woźnica, Alexandros Kalousis, and Melanie Hilario. Kernel-based distances for relational learning. In *Workshop on Multirelational Data Mining, in conjunction with KDD-04*, Seattle, Washington, 2004.

Adam Woźnica, Alexandros Kalousis, and Melanie Hilario. Kernels over relational algebra structures. In *The Ninth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD-2005 Best Student Papers Award)*, Hanoi, Vietnam, 2005a.

Adam Woźnica, Alexandros Kalousis, and Melanie Hilario. Distance-based learning over extended relational algebra structures. In *In Proceedings of the 15th International Conference on Inductive Logic Programming (ILP-2005) (late breaking papers)*, Bonn, Germany, 2005b.

Adam Woźnica, Alexandros Kalousis, and Melanie Hilario. Distances and (indefinite) kernels for sets of objects. In *ICDM '06: Proceedings of the Sixth International Conference on Data Mining*, pages 1151–1156, Washington, DC, USA, 2006a. IEEE Computer Society. ISBN 0-7695-2701-9.

Adam Woźnica, Alexandros Kalousis, and Melanie Hilario. Matching based kernels for labeled graphs. In *Mining and Learning with Graphs (MLG 2006), in conjunction with ECML/PKDD 2006*, Berlin, Germany, 2006b.

Adam Woźnica, Alexandros Kalousis, and Melanie Hilario. Kernels on lists and sets over relational algebra: an application to classification of protein fingerprints. In *The 10th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Singapore, 2006c.

Adam Woźnica, Alexandros Kalousis, and Melanie Hilario. Learning relational distances. In *The 16th International Conference on Inductive Logic Programming (ILP'06), "in area" short papers*, Santiago de Compostela, Spain, 2006d.

Adam Woźnica, Alexandros Kalousis, and Melanie Hilario. Learning to combine distances for complex representations. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 1031–1038, New York, NY, USA, 2007. ACM Press. ISBN 978-1-59593-793-3.

Adam Woźnica, Alexandros Kalousis, and Melanie Hilario. Adaptive matching based approaches for classification of labeled graphs. In *Sumbitted to ECML'08*, 2008.

Stefan Wrobel. An algorithm for multi-relational discovery of subgroups. In *PKDD '97: Proceedings of the First European Symposium on Principles of Data Mining and Knowledge Discovery*, pages 78–87. Springer-Verlag, 1997. ISBN 3-540-63223-9.

Stefan Wrobel, Thomas Gärtner, and Tamás Horváth. Kernels for predictive graph mining. In *From Data and Information Analysis to Knowledge Engineering*. Springer-Verlag, 2006.

Eric P. Xing, Andrew Y. Ng, Michael I. Jordan, and Stuart Russell. Distance metric learning with application to clustering with side-information. In *NIPS 15*, pages 505–512. MIT Press, Cambridge, MA, 2003.

Chyuan-Huei Thomas Yang, Shang-Hong Lai, and Long-Wen Chang. Hybrid image matching combining hausdorff distance with normalized gradient matching. *Pattern Recogn.*, 40(4):1173–1181, 2007. ISSN 0031-3203.

L. Yang and R.Jin. Distance metric learning: A comprehensive survey, 2006.

L. Yang, R. Jin, R. Sukthankar, and Y. Liu. An efficient algorithm for local distance metric learning. In *Proceedings of AAAI*, 2006.

Dmitry Zelenko, Chinatsu Aone, and Anthony Richardella. Kernel methods for relation extraction. *Journal of Machine Learning Research*, 3:1083–1106, 2003.

Hao Zhang and Jitendra Malik. Learning a discriminative classifier using shape context distances. *cvpr*, 01:242, 2003. ISSN 1063-6919.

Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal of Computing*, pages 1245–1262, 1989.

Q. Zhang and S. A. Goldman. Em-dd: An improved multiple-instance learning technique. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, Cambridge, MA, 2002. MIT Press.

Chunjiang Zhao, Wenkang Shi, and Yong Deng. A new hausdorff distance for image matching. *Pattern Recognition Letters*, 26:581–586, 2005.

Appendix A

Datasets

In the following appendix we will briefly present the relational benchmark datasets examined in this study. All the considered problems have in common the fact that the learning instances have no natural propositional representation as a single tuple in a single, fixed-width table. The presentation of the relational datasets is divided into three parts. The first group of learning problems is characterized by the fact that the learning instances are most naturally represented as sets of vectors (Appendix A.1). In Appendix A.2 we describe graph datasets which can have many plausible representations. All the considered graph learning problems are from the domain of chemo-informatics. Finally, in Appendix A.3 we describe the problem of classifying protein fingerprints where the learning examples are represented as general relational structures. Table A.1 gives the overall description of the considered datasets.

Dataset	# instances	# classes
Diterpenes	1503	23
Musk ver. 1	92	2
Musk ver. 2	102	2
Duke	41	2
Mutagenicity	188	2
FR	351	2
FM	349	2
MR	344	2
MM	336	2
Protein Fingerprint	1487	3

Table A.1: Datasets used in this work.

A.1 Sets

The first group of learning problems is where the training examples are given in the form of sets of vectors. In Table A.1 we provide some basic statistics on the considered datasets.

Dataset	min.	max.	median
Diterpenes	20	20	20
Musk ver. 1	2	40	4
Musk ver. 2	1	1044	10
Duke	301	895	522

Table A.2: Datasets where instances are represented as sets of vectors. Minimum, maximum and median number of sets cardinalities is given by min., max. and median, respectively.

Diterpenes

In the diterpene dataset (Džeroski et al., 1996) the goal is to identify the type of diterpenoid compound skeletons given their ^{13}C-*NMR*-Spectrum. Diterpenes are compounds make up from four isoprene units and are thus terpenes. They are found in essential oils in many plants and are often biologically active or exhibit some medical properties. Each ^{13}C-*NMR*-spectrum consists of a set of points, each of the form *(multiplicity, frequency)*. Depending on the number of protons connected to a particular carbon atom, the multiplicity of a signal peak can take the following value: singulet (no proton), doublet (one proton), triplet (two protons) and quadruplet (three protons). There are 1503 learning instances (i.e. spectras) and 23 classes. Each instance is described by 20 tuples.

Musk

The musk datasets was first described in (Dietterich et al., 1997) and the goal is to predict the activity of synthetic musk molecules. A molecule is active if it binds well to larger protein molecules such as enzymes or cell-surface receptors. It is the shape of molecules which determines the binding strength, however, molecules can change their shapes by rotating some of the their internal bonds. Every combination of angles of the rotatable bonds of a molecule, i.e. its shape, defines a "conformation". Since we do not know a priori which of the conformation of a molecule will bind well to an enzyme or cell-surface receptor, we should consider all the possible shapes of a molecule by representing it by a set of descriptions of its different conformations. We are hence conformed with an multiple-instance learning problem.

The class labels were provided by human domain experts and they are available two overlapping datasets. Version 1 of the dataset contains 92 molecules

where 47 are labeled as "Musk" and 45 are labeled as "Non-Musk". The cardinality of the sets in this dataset ranges mostly between 2 and 10 with a maximum value of 40 and a median of 4. Each tuple has 166 features. Version 2 contains altogether 102 molecules where 63 are labeled as "Non-Musk" and 39 are labeled as "Musk". Here the cardinality ranges mostly between 1 and 60 with a maximum value of 1044 and a median of 12. The second version of the problem has sets of much more varied and greater cardinality than the first version.

Duke

This is a dataset from the domain of proteomics (Campa et al., 2003). The goal is to classify mass-spectra acquired from blood serum of individuals that have developed lung cancer or healthy individuals. The dataset consists of mass-spectra of 24 diseased and 17 healthy persons. Each spectrum is described by a set of peaks of the form *(mass,intensity)*. The median of the cardinalities of the sets is 522. We will consider a version of this problem where a peak will consist only of its mass. This representation corresponds roughly to a binary representation of a spectrum where presence of a given mass indicates presence of the corresponding peak. The most similar spectra will be the ones that have more similar peaks determined by the mass dimension. In our context this is a more appropriate representation for a mass spectrometry problem since what is more important is to find an appropriate matching of masses and only then take into account intensity differences. It should be mentioned that other representations are possible where both mass and intensity are used. This representation places on an equal footing both dimensions, it thus violates the semantics of the problem and does not naturally fit within our context so we will not consider it here.

A.2 Graphs

The second group of relational datasets are graph problems, which are from the domain of chemo-informatics. In all the learning problems the goal is to predict the activity of chemical molecules represented by *undirected* labeled graphs where vertices are atoms, connected by edges that are covalent bonds. In such domains the vertices and edges are usually labeled, modeling atom and bond types, respectively. Various statistics of the examined datasets are summarized in Table A.3.

Mutagenicity

The Mutagenicity was introduced in (Srinivasan et al., 1994). The application task is the prediction of mutagenicity of a set of 230 aromatic and heteroaromatic nitro-compounds. Mutagenic compounds are often know to be carcinogenic and could cause damage to DNA. Clearly, it is of great importance to the pharmaceutical industry to determine which compounds show mutagenic activity.

	mutagenesis	fr	fm	mr	mm
# pos.	125	121	143	152	129
# neg.	63	230	206	192	207
min. $\|G(\mathcal{V})\|$	14	2	2	2	2
max. $\|G(\mathcal{V})\|$	40	109	109	109	109
med. $\|G(\mathcal{V})\|$	26	23	22	22	21
min. $\|G(\mathcal{E})\|$	14	1	1	1	1
max. $\|G(\mathcal{E})\|$	44	108	108	108	108
med. $\|G(\mathcal{E})\|$	28	24	23	23	21
mean branching factor	2.13	1.99	1.99	1.98	1.99

Table A.3: Various statistics of the graph datasets. # pos. and # neg. denote the number of positive and negative examples, respectively. Minimum, maximum and median number of atoms is given by min. $|G(\mathcal{V})|$, max. $|G(\mathcal{V})|$ and med. $|G(\mathcal{V})|$. Minimum, maximum and median number of bonds is given by min. $|G(\mathcal{E})|$, max. $|G(\mathcal{E})|$ and med. $|G(\mathcal{E})|$.

Training instances are divided to 42 "regression unfriendly" and 188 "regression friendly". We worked with the latter dataset where 125 instances have positive log mutagenicity and are labeled as active whereas 63 molecules have zero or negative log mutagenecity labeled as inactive. Bonds are described by the type of the bond while atoms are described by their charge numeric values, type e.g. aromatic carbon, aryl carbon and name e.g. N, F, S, O, etc.

Carcinogenicity

The other graph classification problem comes from the Predictive Toxicology Challenge and is defined over carcinogenicity properties of chemical compounds (Helma et al., 2001). This dataset lists the bioassays of 417 chemical compounds for four type of rodents: male rats (MR), male mice (MM), female rats (FR) and female mice (FM) which give rise to four independent classification problems. We transformed the original dataset with eight classes into a binary problem by ignoring EE equivocal evidence, E equivocal and IS inadequate study classes, grouping SE some evidence, CE clear evidence and P positive in the positive class and N negative and NE no evidence in the negative one. After this transformation the MR, MM, FR and FM datasets had 344, 336, 351 and 349 instances, respectively.

A.3 General relational structures

Protein Fingerprint

The Protein Fingerprint classification problem was first introduced in (Hilario et al., 2004). Protein fingerprints are groups of conserved motifs regions drawn from multiple sequences alignment that can be used as diagnostic signatures to

identify and characterize collections of protein sequences. Protein fingerprints are schematically presented in Figure A.1. The fingerprints are stored in the *PRINTS* database (Attwood et al., 2003), after time-consuming annotation by domain experts who must first determine the fingerprint type. Broadly speaking, fingerprints may be diagnostic for a gene *family* or *superfamily* united by a common function, or a *domain family* united by a common structural motif. Here we attempt to build classifiers based on information drawn from the fingerprints' component motifs and protein sequences from the *SWISS-PROT* database. We are therefore confronted with a multi-relational learning problem, which can be addressed most naturally using a relational approach. Our approach will be different from the one presented in (Hilario et al., 2004) since there the task representation is propositionalized by aggregating protein and motif characteristic over a fingerprint.

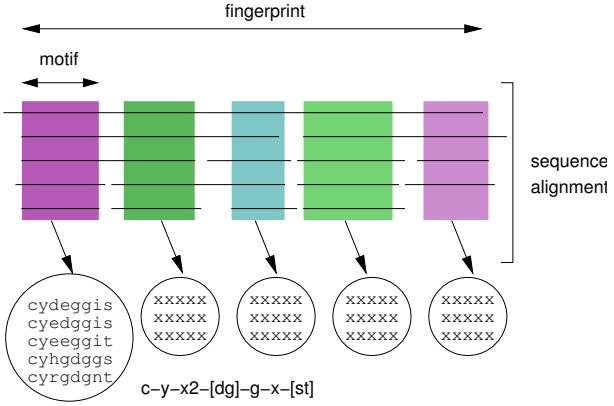


Figure A.1: Schematic representation of a protein fingerprint. Each row is a protein sequence. Solid rectangles denote motifs.

We modeled this data in the following way: the "fingerprints" main relation with global characteristics of the instances is associated through an one-to-many relation with the "motifs" relation. Additionally there is a number of relations with aggregated information about proteins (actually proteins IDs) associated with the main relation using one-to-one relations. The actual relational representation used in this study in given in Figure A.2.

Protein fingerprints are globally characterized by number of component motifs (attribute *#motifs* in Figure A.2) and proteins (*#proteins*). The coherence of a fingerprint is expressed by the proportion of protein sequences that match all (*TPRate*) or only a part of the motifs (*PPRate*) in the fingerprint. Fingerprints are also described by the relative frequency of SWISS-PROT IDS in the set of constituent proteins (*pSP*).

Individual *motifs* are characterized by their size and conservation. The size is expressed by the number of amino acids *length* and *coverage* i.e., the fraction of protein sequences in the fingerprint that match the motif. The conservation

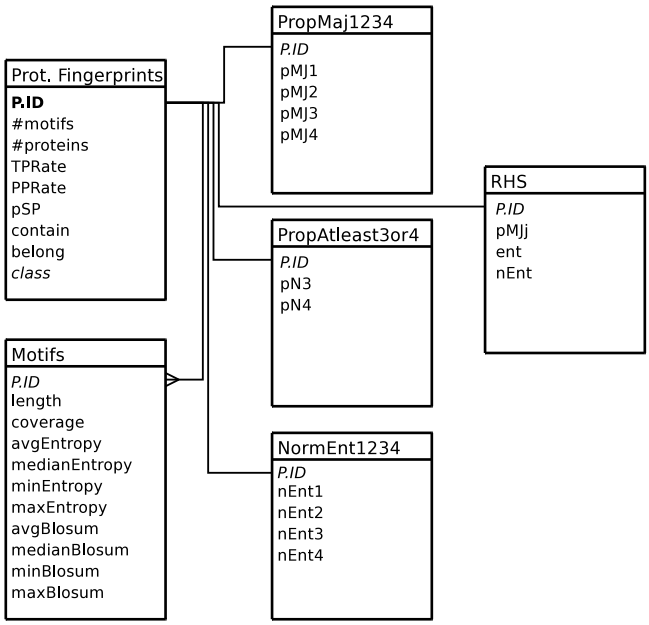


Figure A.2: Relational representation of protein fingerprints.

was encoded by motif's entropy averaged over the entropies of its individual columns and normalized by the number of sequences. The other way to measure the motif's conservation is to use domain knowledge about similarity between residues which has been codified in substitution matrices. Here the *Blosum* substitution matrices were used and a motif's *blosum score* was computed by averaging over the blosum scores of its individual residues.

The last source of information are *protein sequences* and more precisely their *SWISS-PROT/TrEMBL* labels or accession numbers. Here we use these codes to retrieve the SWISS-PROT entry for a given protein. We focus on SWISS-PROT'S CC similarity field, which often contains information about the family membership of a protein. Instead of boolean value indication the presence or absence of the specific value in this filed we use the proportion computed over the proteins constituting the fingerprint. Besides of that the SWISS-PROT ID field itself is particularly informative be virtue of its structure. It is composed of two parts separated by an underscore: the left hand side LHS and the right hand side RHS. Here we use four set of features based on the SWISS-PROT ID: (i) the majority score (the *PropMaj1234* table) defined as the proportion of LHSs sharing the most frequent common root of 1-4 characters, (ii) (normalized) entropy (*NormEnt1234*) averaged over the first 1-4 characters of the LHS, (iii) fraction of proteins whose LHS length is greater than 3-4 characters (*PropAtLeast3or4*), and (iv) majority score and entropy (*RHS*), but computed over the RHS as a whole. Features computed on the basis of these labels can be considered as statistics computed on the set of proteins and hence they can

be stored in the "fingerprints" relation. However, keeping them in separate tables provides us a way to treat a missing values which is the case here since not all proteins have an SWISS-PROT entry. More information about different attributes in different relations can be found in (Hilario et al., 2004).

We defined three different data representations based on different building blocks of learning instances. First, we assumed that each "fingerprint" is associated with a *set* of "motifs". In the second approach we assumed that the order in which motifs appear along the sequence of amino acids is important i.e. we assumed that each "fingerprint" is associated with a *list* of motifs. The latter representation is justified by the fact that a motif is basically a multiple sequence alignment with a number of conserved regions so there exists an intrinsic notion of order along protein sequences. Finally, in the third approach (which is exploited in Section 4.4.5) we combine the two previous representations: an fingerprint is related with both with the set and list of motifs.

Appendix B

Detailed Experimental Results

In the following appendix we will report the detailed significance results. In all the tables a cell with the +/- signs indicate that the predictive performance of an algorithm associated with the corresponding row is significantly better/worse than that of the algorithm associated with the given column, the = sign indicates that the performance difference is not significant. We provide results both for complex distances (Appendix B.1) and complex kernels (Appendix B.2). In Appendix B.1 for d_T we set $\theta = 0.01$.

B.1 Detailed Experimental Results for Distances

	diterpenes										
	d_{SL}	d_{CL}	d_{AL}	d_{SMD}	d_H	d_{RIBL}	d_T	d_S	d_{FS}	d_L	d_M
d_{SL}	x	+	+	-	-	-	-	-	-	-	-
d_{CL}	-	x	-	-	-	-	-	-	-	-	-
d_{AL}	-	+	x	-	-	-	-	-	-	-	-
d_{SMD}	+	+	+	x	+	=	-	=	=	+	=
d_H	+	+	+	-	x	-	-	-	-	-	-
d_{RIBL}	+	+	+	=	+	x	-	=	=	=	=
d_T	+	+	+	+	+	+	x	+	+	+	+
d_S	+	+	+	=	+	=	-	x	=	=	-
d_{FS}	+	+	+	=	+	=	-	=	x	=	-
d_L	+	+	+	-	+	=	-	=	=	x	-
d_M	+	+	+	=	+	=	-	+	+	+	x
	duke										
	d_{SL}	d_{CL}	d_{AL}	d_{SMD}	d_H	d_{RIBL}	d_T	d_S	d_{FS}	d_L	d_M
d_{SL}	x	=	=	-	=	-	=	=	=	-	=
d_{CL}	=	x	=	-	=	-	=	-	=	-	=
d_{AL}	=	=	x	=	=	=	=	=	=	=	=
d_{SMD}	+	+	=	x	=	=	+	=	=	=	=
d_H	=	=	=	=	x	=	=	=	=	=	=
d_{RIBL}	+	+	=	=	=	x	=	=	=	=	=
d_T	=	=	=	-	=	=	x	=	=	=	=
d_S	=	+	=	=	=	=	=	x	=	=	=
d_{FS}	=	=	=	=	=	=	=	=	x	=	=
d_L	+	+	=	=	=	=	=	=	=	x	=
d_M	=	=	=	=	=	=	=	=	=	=	x

Table B.1: Detailed results of McNemar's test of statistical significance for kNN in diterpenes and duke.

	musk (ver. 1)										
	d_{SL}	d_{CL}	d_{AL}	d_{SMD}	d_{H}	d_{RIBL}	d_{T}	d_{S}	d_{FS}	d_{L}	d_{M}
d_{SL}	x	+	=	=	=	+	+	=	=	=	=
d_{CL}	-	x	-	-	-	=	=	-	-	-	=
d_{AL}	=	+	x	=	=	+	+	=	=	=	=
d_{SMD}	=	+	=	x	=	+	+	=	=	=	=
d_{H}	=	+	=	=	x	+	+	=	=	=	=
d_{RIBL}	-	=	-	-	-	x	=	-	-	-	=
d_{T}	-	=	-	-	-	=	x	-	-	-	-
d_{S}	=	+	=	=	=	+	+	x	=	=	+
d_{FS}	=	+	=	=	=	+	+	=	x	=	=
d_{L}	=	+	=	=	=	+	+	=	=	x	+
d_{M}	=	=	=	=	=	=	+	-	=	-	x
	musk (ver. 2)										
	d_{SL}	d_{CL}	d_{AL}	d_{SMD}	d_{H}	d_{RIBL}	d_{T}	d_{S}	d_{FS}	d_{L}	d_{M}
d_{SL}	x	=	=	=	=	=	=	=	=	=	=
d_{CL}	=	x	=	=	=	=	=	=	=	=	=
d_{AL}	=	=	x	=	=	=	=	=	=	=	=
d_{SMD}	=	=	=	x	=	=	=	=	=	=	=
d_{H}	=	=	=	=	x	=	=	=	=	=	=
d_{RIBL}	=	=	=	=	=	x	=	=	-	=	=
d_{T}	=	=	=	=	=	=	x	=	-	=	=
d_{S}	=	=	=	=	=	=	=	x	=	=	=
d_{FS}	=	=	=	=	=	+	+	=	x	=	+
d_{L}	=	=	=	=	=	=	=	=	=	x	=
d_{M}	=	=	=	=	=	=	=	=	-	=	x

Table B.2: Detailed results of McNemar's test of statistical significance for kNN in musk.

	mutagenesis (ver. 1)										
	d_{SL}	d_{CL}	d_{AL}	d_{SMD}	d_H	d_{RIBL}	d_T	d_S	d_{FS}	d_L	d_M
d_{SL}	x	=	+	=	=	=	-	=	=	=	=
d_{CL}	=	x	+	=	=	=	-	=	=	=	=
d_{AL}	-	-	x	-	-	=	-	-	-	-	=
d_{SMD}	=	=	+	x	=	+	=	=	=	=	+
d_H	=	=	+	=	x	=	-	=	=	=	=
d_{RIBL}	=	=	=	-	=	x	-	=	-	-	=
d_T	+	+	+	=	+	+	x	+	=	+	+
d_S	=	=	+	=	=	=	-	x	=	=	+
d_{FS}	=	=	+	=	=	+	=	=	x	=	+
d_L	=	=	+	=	=	+	-	=	=	x	+
d_M	=	=	=	-	=	=	-	-	-	-	x
	mutagenesis (ver. 2)										
	d_{SL}	d_{CL}	d_{AL}	d_{SMD}	d_H	d_{RIBL}	d_T	d_S	d_{FS}	d_L	d_M
d_{SL}	x	=	=	-	=	-	-	=	-	-	=
d_{CL}	=	x	=	-	-	-	-	-	-	-	=
d_{AL}	=	=	x	-	-	-	-	-	-	-	=
d_{SMD}	+	+	+	x	+	=	=	=	=	=	+
d_H	=	+	+	-	x	=	-	=	=	=	=
d_{RIBL}	+	+	+	=	=	x	-	=	=	=	=
d_T	+	+	+	=	+	+	x	+	=	=	+
d_S	=	+	+	=	=	=	-	x	=	=	=
d_{FS}	+	+	+	=	=	=	=	=	x	=	+
d_L	+	+	+	=	=	=	=	=	=	x	+
d_M	=	=	=	-	=	=	-	=	-	-	x

Table B.3: Detailed results of McNemar's test of statistical significance for kNN in mutagenesis.

	FM (ver. 1)										
	d_{SL}	d_{CL}	d_{AL}	d_{SMD}	d_H	d_{RIBL}	d_T	d_S	d_{FS}	d_L	d_M
d_{SL}	x	+	=	=	=	=	=	=	=	=	=
d_{CL}	-	x	-	=	-	-	-	=	=	-	=
d_{AL}	=	+	x	=	=	=	=	=	=	=	=
d_{SMD}	=	=	=	x	-	=	-	=	=	=	=
d_H	=	+	=	+	x	=	=	+	=	=	=
d_{RIBL}	=	+	=	=	=	x	=	=	=	=	=
d_T	=	+	=	+	=	=	x	+	+	=	+
d_S	=	=	=	=	-	=	-	x	=	-	=
d_{FS}	=	=	=	=	=	=	-	=	x	=	=
d_L	=	+	=	=	=	=	=	+	=	x	=
d_M	=	=	=	=	=	=	-	=	=	=	x
	FM (ver. 2)										
	d_{SL}	d_{CL}	d_{AL}	d_{SMD}	d_H	d_{RIBL}	d_T	d_S	d_{FS}	d_L	d_M
d_{SL}	x	+	=	=	=	=	=	=	=	=	=
d_{CL}	-	x	=	-	-	=	-	=	=	-	=
d_{AL}	=	=	x	=	=	=	-	=	=	=	=
d_{SMD}	=	+	=	x	=	=	=	+	=	=	=
d_H	=	+	=	=	x	=	=	+	=	=	=
d_{RIBL}	=	=	=	=	=	x	=	=	=	=	=
d_T	=	+	+	=	=	=	x	+	+	=	=
d_S	=	=	=	-	-	=	-	x	=	-	=
d_{FS}	=	=	=	=	=	=	-	=	x	-	=
d_L	=	+	=	=	=	=	=	+	+	x	=
d_M	=	=	=	=	=	=	=	=	=	=	x

Table B.4: Detailed results of McNemar's test of statistical significance for kNN in FM.

	FR (ver. 1)										
	d_{SL}	d_{CL}	d_{AL}	d_{SMD}	d_H	d_{RIBL}	d_T	d_S	d_{FS}	d_L	d_M
d_{SL}	x	+	=	=	=	=	=	=	=	=	=
d_{CL}	-	x	-	=	=	-	=	-	=	-	=
d_{AL}	=	+	x	=	=	=	=	=	=	=	=
d_{SMD}	=	=	=	x	=	=	=	=	=	=	=
d_H	=	=	=	=	x	=	=	=	=	=	=
d_{RIBL}	=	+	=	=	=	x	=	=	=	=	=
d_T	=	=	=	=	=	=	x	=	=	=	=
d_S	=	+	=	=	=	=	=	x	=	=	=
d_{FS}	=	=	=	=	=	=	=	=	x	=	=
d_L	=	+	=	=	=	=	=	=	=	x	=
d_M	=	=	=	=	=	=	=	=	=	=	x
	FR (ver. 2)										
	d_{SL}	d_{CL}	d_{AL}	d_{SMD}	d_H	d_{RIBL}	d_T	d_S	d_{FS}	d_L	d_M
d_{SL}	x	+	=	=	=	=	=	=	=	=	=
d_{CL}	-	x	=	-	=	=	=	=	=	-	=
d_{AL}	=	=	x	=	=	=	=	=	=	=	=
d_{SMD}	=	+	=	x	=	=	=	=	=	=	=
d_H	=	=	=	=	x	=	=	=	=	=	=
d_{RIBL}	=	=	=	=	=	x	=	=	=	=	=
d_T	=	=	=	=	=	=	x	=	=	=	=
d_S	=	=	=	=	=	=	=	x	=	=	=
d_{FS}	=	=	=	=	=	=	=	=	x	=	=
d_L	=	+	=	=	=	=	=	=	=	x	=
d_M	=	=	=	=	=	=	=	=	=	=	x

Table B.5: Detailed results of McNemar's test of statistical significance for kNN in FR.

	MM (ver. 1)										
	d_{SL}	d_{CL}	d_{AL}	d_{SMD}	d_H	d_{RIBL}	d_T	d_S	d_{FS}	d_L	d_M
d_{SL}	x	+	+	=	=	=	=	=	=	=	=
d_{CL}	-	x	-	-	-	-	-	-	-	-	-
d_{AL}	-	+	x	-	=	=	=	=	=	-	=
d_{SMD}	=	+	+	x	=	=	=	=	=	=	=
d_H	=	+	=	=	x	=	=	=	=	=	=
d_{RIBL}	=	+	=	=	=	x	=	=	=	-	=
d_T	=	+	=	=	=	=	x	=	=	=	=
d_S	=	+	=	=	=	=	=	x	=	=	=
d_{FS}	=	+	=	=	=	=	=	=	x	=	=
d_L	=	+	+	=	=	+	=	=	=	x	=
d_M	=	+	=	=	=	=	=	=	=	=	x

	MM (ver. 2)										
	d_{SL}	d_{CL}	d_{AL}	d_{SMD}	d_H	d_{RIBL}	d_T	d_S	d_{FS}	d_L	d_M
d_{SL}	x	=	=	=	=	=	=	=	=	=	=
d_{CL}	=	x	=	-	-	=	=	-	=	-	-
d_{AL}	=	=	x	=	=	=	=	=	=	=	=
d_{SMD}	=	+	=	x	=	=	=	=	=	=	=
d_H	=	+	=	=	x	=	+	=	=	=	=
d_{RIBL}	=	=	=	=	=	x	=	=	=	=	=
d_T	=	=	=	=	-	=	x	=	=	=	=
d_S	=	+	=	=	=	=	=	x	=	=	=
d_{FS}	=	=	=	=	=	=	=	=	x	=	=
d_L	=	+	=	=	=	=	=	=	=	x	=
d_M	=	+	=	=	=	=	=	=	=	=	x

Table B.6: Detailed results of McNemar's test of statistical significance for kNN in MM.

	MR (ver. 1)										
	d_{SL}	d_{CL}	d_{AL}	d_{SMD}	d_H	d_{RIBL}	d_T	d_S	d_{FS}	d_L	d_M
d_{SL}	x	+	=	=	=	=	=	=	=	=	=
d_{CL}	-	x	-	-	-	-	-	-	-	-	=
d_{AL}	=	+	x	=	=	=	=	=	=	=	=
d_{SMD}	=	+	=	x	+	=	=	+	=	=	+
d_H	=	+	=	-	x	=	=	=	=	-	=
d_{RIBL}	=	+	=	=	=	x	=	=	=	=	=
d_T	=	+	=	=	=	=	x	=	=	=	=
d_S	=	+	=	-	=	=	=	x	=	-	=
d_{FS}	=	+	=	=	=	=	=	=	x	=	=
d_L	=	+	=	=	+	=	=	+	=	x	+
d_M	=	=	=	-	=	=	=	=	=	-	x

	MR (ver. 2)										
	d_{SL}	d_{CL}	d_{AL}	d_{SMD}	d_H	d_{RIBL}	d_T	d_S	d_{FS}	d_L	d_M
d_{SL}	x	=	=	=	=	+	=	=	=	=	+
d_{CL}	=	x	=	=	-	=	-	=	=	=	=
d_{AL}	=	=	x	=	=	=	=	=	=	=	=
d_{SMD}	=	=	=	x	=	+	=	=	=	=	+
d_H	=	+	=	=	x	+	=	=	=	=	+
d_{RIBL}	-	=	=	-	-	x	-	=	=	=	=
d_T	=	+	=	=	=	+	x	=	=	=	+
d_S	=	=	=	=	=	=	=	x	=	=	=
d_{FS}	=	=	=	=	=	=	=	=	x	=	+
d_L	=	=	=	=	=	=	=	=	=	x	+
d_M	-	=	=	-	-	=	-	=	-	-	x

Table B.7: Detailed results of McNemar's test of statistical significance for kNN in MR.

	Non-weighted Protein Fingerprint											
	d_{SL}	d_{CL}	d_{AL}	d_{SMD}	d_H	d_{RIBL}	d_T	d_S	d_{FS}	d_L	d_M	d_{edit}
d_{SL}	x	=	=	-	=	=	+	=	=	+	-	+
d_{CL}	=	x	=	=	=	=	+	=	=	+	-	+
d_{AL}	=	=	x	=	=	=	+	=	=	+	-	+
d_{SMD}	+	=	=	x	=	=	+	=	=	+	=	+
d_H	=	=	=	=	x	=	+	=	=	+	-	+
d_{RIBL}	=	=	=	=	=	x	+	=	=	+	-	+
d_T	-	-	-	-	-	-	x	-	-	=	-	=
d_S	=	=	=	=	=	=	+	x	=	+	-	+
d_{FS}	=	=	=	=	=	=	+	=	x	+	=	+
d_L	-	-	-	-	-	-	=	-	-	x	-	-
d_M	+	+	+	=	+	+	+	+	=	+	x	+
d_{edit}	-	-	-	-	-	-	=	-	-	+	-	x
	Weighted Protein Fingerprint											
	d_{SL}	d_{CL}	d_{AL}	d_{SMD}	d_H	d_{RIBL}	d_T	d_S	d_{FS}	d_L	d_M	d_{edit}
d_{SL}	x	+	=	=	=	=	+	=	=	+	=	+
d_{CL}	-	x	-	-	-	=	=	-	-	=	-	=
d_{AL}	=	+	x	-	=	=	+	-	-	=	=	=
d_{SMD}	=	+	+	x	+	+	+	=	=	+	=	+
d_H	=	+	=	-	x	=	+	=	-	=	=	=
d_{RIBL}	=	=	=	-	=	x	+	-	-	=	=	=
d_T	-	=	-	-	-	-	x	-	-	=	-	-
d_S	=	+	+	=	=	+	+	x	=	+	=	+
d_{FS}	=	+	+	=	+	+	+	=	x	+	=	+
d_L	-	=	=	-	=	=	=	-	-	x	-	=
d_M	=	+	=	=	=	=	+	=	=	+	x	=
d_{edit}	-	=	=	-	=	=	+	-	-	=	=	x

Table B.8: Detailed results of McNemar's test of statistical significance for kNN in Protein Fingerprints.

B.2 Detailed experimental for Kernels

B.2.1 Results for CP kernel using SVMs

	diterpenes											
	1	2	3	4	5	6	7	8	9	10	11	12
	$f_{norm}(\cdot) = \lvert\cdot\rvert$											
1 k_{lin}	x	-	-	+	-	-	=	-	-	+	-	-
2 $k_{poly,p=2}$	+	x	-	+	-	-	+	-	-	+	-	-
3 $k_{poly,p=3}$	+	+	x	+	-	-	+	=	-	+	-	-
4 $k_{RBF,\gamma=0.1}$	-	-	-	x	-	-	-	-	-	=	-	-
5 $k_{RBF,\gamma=1}$	+	+	+	+	x	-	+	+	=	+	-	-
6 $k_{RBF,\gamma=10}$	+	+	+	+	+	x	+	+	+	+	+	-
	$f_{norm}(\cdot) = \sqrt{k(\cdot,\cdot)}$											
7 k_{lin}	=	-	-	+	-	-	x	-	-	+	-	-
8 $k_{poly,p=2}$	+	+	=	+	-	-	+	x	-	+	-	-
9 $k_{poly,p=3}$	+	+	+	+	=	-	+	+	x	+	-	-
10 $k_{RBF,\gamma=0.1}$	-	-	-	=	-	-	-	-	-	x	-	-
11 $k_{RBF,\gamma=1}$	+	+	+	+	+	-	+	+	+	+	x	-
12 $k_{RBF,\gamma=10}$	+	+	+	+	+	+	+	+	+	+	+	x
	duke											
	1	2	3	4	5	6	7	8	9	10	11	12
	$f_{norm}(\cdot) = \lvert\cdot\rvert$											
1 k_{lin}	x	=	=	=	=	=	=	=	=	=	=	=
2 $k_{poly,p=2}$	=	x	=	=	=	=	=	=	=	=	=	=
3 $k_{poly,p=3}$	=	=	x	=	=	=	=	=	=	=	=	=
4 $k_{RBF,\gamma=0.1}$	=	=	=	x	=	=	=	=	=	=	=	=
5 $k_{RBF,\gamma=1}$	=	=	=	=	x	=	=	=	=	=	=	=
6 $k_{RBF,\gamma=10}$	=	=	=	=	=	x	=	=	=	=	=	=
	$f_{norm}(\cdot) = \sqrt{k(\cdot,\cdot)}$											
7 k_{lin}	=	=	=	=	=	=	x	=	=	=	=	=
8 $k_{poly,p=2}$	=	=	=	=	=	=	=	x	=	=	=	=
9 $k_{poly,p=3}$	=	=	=	=	=	=	=	=	x	=	=	=
10 $k_{RBF,\gamma=0.1}$	=	=	=	=	=	=	=	=	=	x	=	=
11 $k_{RBF,\gamma=1}$	=	=	=	=	=	=	=	=	=	=	x	=
12 $k_{RBF,\gamma=10}$	=	=	=	=	=	=	=	=	=	=	=	x

Table B.9: Detailed results of McNemar's test of statistical significance for SVM in diterpenes and duke.

	musk (ver. 1)											
	1	2	3	4	5	6	7	8	9	10	11	12
	$f_{norm}(\cdot) = \lvert\cdot\rvert$											
1 k_{lin}	x	=	=	=	=	+	=	=	=	=	=	+
2 $k_{poly,p=2}$	=	x	=	=	=	+	=	=	=	=	=	+
3 $k_{poly,p=3}$	=	=	x	=	=	+	=	=	=	=	=	+
4 $k_{RBF,\gamma=0.1}$	=	=	=	x	=	+	=	=	=	=	=	+
5 $k_{RBF,\gamma=1}$	=	=	=	=	x	+	=	=	=	=	=	+
6 $k_{RBF,\gamma=10}$	-	-	-	-	-	x	-	-	-	-	-	=
	$f_{norm}(\cdot) = \sqrt{k(\cdot,\cdot)}$											
7 k_{lin}	=	=	=	=	=	+	x	=	=	=	=	+
8 $k_{poly,p=2}$	=	=	=	=	=	+	=	x	=	=	=	+
9 $k_{poly,p=3}$	=	=	=	=	=	+	=	=	x	=	=	+
10 $k_{RBF,\gamma=0.1}$	=	=	=	=	=	+	=	=	=	x	=	+
11 $k_{RBF,\gamma=1}$	=	=	=	=	=	+	=	=	=	=	x	+
12 $k_{RBF,\gamma=10}$	-	-	-	-	-	=	-	-	-	-	-	x

	musk ver. 2											
	1	2	3	4	5	6	7	8	9	10	11	12
	$f_{norm}(\cdot) = \lvert\cdot\rvert$											
1 k_{lin}	x	=	=	-	=	+	=	=	=	=	=	+
2 $k_{poly,p=2}$	=	x	=	-	=	+	=	=	-	=	=	+
3 $k_{poly,p=3}$	=	=	x	-	=	+	=	=	=	=	=	+
4 $k_{RBF,\gamma=0.1}$	+	+	+	x	=	+	+	+	=	=	=	+
5 $k_{RBF,\gamma=1}$	=	=	=	=	x	+	=	=	=	=	=	+
6 $k_{RBF,\gamma=10}$	-	-	-	-	-	x	-	-	-	-	-	=
	$f_{norm}(\cdot) = \sqrt{k(\cdot,\cdot)}$											
7 k_{lin}	=	=	=	-	=	+	x	=	-	=	=	+
8 $k_{poly,p=2}$	=	=	=	-	=	+	=	x	=	=	=	+
9 $k_{poly,p=3}$	=	+	=	=	=	+	+	=	x	=	=	+
10 $k_{RBF,\gamma=0.1}$	=	=	=	=	=	+	=	=	=	x	=	+
11 $k_{RBF,\gamma=1}$	=	=	=	=	=	+	=	=	=	=	x	+
12 $k_{RBF,\gamma=10}$	-	-	-	-	-	=	-	-	-	-	-	x

Table B.10: Detailed results of McNemar's test of statistical significance for SVM in musk.

	mutagenesis (ver. 1)											
	1	2	3	4	5	6	7	8	9	10	11	12
							k_Σ					
1 k_{lin}	x	=	=	+	=	=	-	-	-	=	-	=
2 $k_{poly,p=2}$	=	x	=	+	=	=	-	-	-	=	-	=
3 $k_{poly,p=3}$	=	=	x	=	=	=	-	-	-	=	-	=
4 $k_{RBF,\gamma=0.1}$	-	-	=	x	=	-	-	-	-	=	-	-
5 $k_{RBF,\gamma=1}$	=	=	=	=	x	=	-	-	-	=	-	-
6 $k_{RBF,\gamma=10}$	=	=	=	+	=	x	-	-	-	=	=	=
							k_Π					
7 k_{lin}	+	+	+	+	+	+	x	=	=	+	=	=
8 $k_{poly,p=2}$	+	+	+	+	+	+	=	x	=	+	=	=
9 $k_{poly,p=3}$	+	+	+	+	+	+	=	=	x	+	=	=
10 $k_{RBF,\gamma=0.1}$	=	=	=	=	=	=	-	-	-	x	-	-
11 $k_{RBF,\gamma=1}$	+	+	+	+	+	=	=	=	=	+	x	=
12 $k_{RBF,\gamma=10}$	=	=	=	+	+	=	=	=	=	+	=	x

	mutagenesis (ver. 2)											
	1	2	3	4	5	6	7	8	9	10	11	12
							k_Σ					
1 k_{lin}	x	=	=	=	=	=	=	=	=	=	=	=
2 $k_{poly,p=2}$	=	x	=	=	=	=	=	=	=	=	=	=
3 $k_{poly,p=3}$	=	=	x	=	=	=	=	=	=	=	=	=
4 $k_{RBF,\gamma=0.1}$	=	=	=	x	=	=	=	=	=	=	-	-
5 $k_{RBF,\gamma=1}$	=	=	=	=	x	=	=	=	=	=	=	=
6 $k_{RBF,\gamma=10}$	=	=	=	=	=	x	=	=	=	=	=	-
							k_Π					
7 k_{lin}	=	=	=	=	=	=	x	=	=	=	=	=
8 $k_{poly,p=2}$	=	=	=	=	=	=	=	x	=	=	=	=
9 $k_{poly,p=3}$	=	=	=	=	=	=	=	=	x	=	=	=
10 $k_{RBF,\gamma=0.1}$	=	=	=	=	=	=	=	=	=	x	=	=
11 $k_{RBF,\gamma=1}$	=	=	=	+	=	=	=	=	=	=	x	=
12 $k_{RBF,\gamma=10}$	=	=	=	+	=	+	=	=	=	=	=	x

Table B.11: Detailed results of McNemar's test of statistical significance for SVM in mutagenesis.

	FM (ver. 1)											
	1	2	3	4	5	6	7	8	9	10	11	12
	k_Σ											
1 k_{lin}	x	=	=	=	=	=	=	=	=	=	=	=
2 $k_{poly,p=2}$	=	x	=	=	=	=	=	=	=	=	=	=
3 $k_{poly,p=3}$	=	=	x	=	=	=	=	=	=	=	=	=
4 $k_{RBF,\gamma=0.1}$	=	=	=	x	=	=	=	=	=	=	=	=
5 $k_{RBF,\gamma=1}$	=	=	=	=	x	=	=	=	=	=	=	=
6 $k_{RBF,\gamma=10}$	=	=	=	=	=	x	=	=	=	=	=	=
	k_Π											
7 k_{lin}	=	=	=	=	=	=	x	=	=	=	=	=
8 $k_{poly,p=2}$	=	=	=	=	=	=	=	x	=	=	=	=
9 $k_{poly,p=3}$	=	=	=	=	=	=	=	=	x	=	=	=
10 $k_{RBF,\gamma=0.1}$	=	=	=	=	=	=	=	=	=	x	=	=
11 $k_{RBF,\gamma=1}$	=	=	=	=	=	=	=	=	=	=	x	=
12 $k_{RBF,\gamma=10}$	=	=	=	=	=	=	=	=	=	=	=	x

	FM (ver. 2)											
	1	2	3	4	5	6	7	8	9	10	11	12
	k_Σ											
1 k_{lin}	x	=	=	=	=	=	=	=	=	=	=	=
2 $k_{poly,p=2}$	=	x	=	=	=	=	=	=	=	=	=	=
3 $k_{poly,p=3}$	=	=	x	=	=	=	=	=	=	=	=	=
4 $k_{RBF,\gamma=0.1}$	=	=	=	x	=	=	=	=	=	=	=	=
5 $k_{RBF,\gamma=1}$	=	=	=	=	x	=	=	=	=	=	=	-
6 $k_{RBF,\gamma=10}$	=	=	=	=	=	x	=	=	=	=	=	=
	k_Π											
7 k_{lin}	=	=	=	=	=	=	x	=	=	=	=	=
8 $k_{poly,p=2}$	=	=	=	=	=	=	=	x	=	=	=	=
9 $k_{poly,p=3}$	=	=	=	=	=	=	=	=	x	=	=	=
10 $k_{RBF,\gamma=0.1}$	=	=	=	=	=	=	=	=	=	x	=	=
11 $k_{RBF,\gamma=1}$	=	=	=	=	=	=	=	=	=	=	x	=
12 $k_{RBF,\gamma=10}$	=	=	=	=	+	=	=	=	=	=	=	x

Table B.12: Detailed results of McNemar's test of statistical significance for SVM in FM.

	FR (ver. 1)											
	1	2	3	4	5	6	7	8	9	10	11	12
						k_Σ						
1 k_{lin}	x	=	=	=	=	=	=	=	=	=	=	=
2 $k_{poly,p=2}$	=	x	=	=	=	=	=	=	=	=	=	=
3 $k_{poly,p=3}$	=	=	x	=	=	=	=	=	=	=	=	=
4 $k_{RBF,\gamma=0.1}$	=	=	=	x	=	=	=	=	=	=	=	=
5 $k_{RBF,\gamma=1}$	=	=	=	=	x	=	=	=	=	=	=	=
6 $k_{RBF,\gamma=10}$	=	=	=	=	=	x	=	=	=	=	=	=
						k_Π						
7 k_{lin}	=	=	=	=	=	=	x	=	=	=	=	=
8 $k_{poly,p=2}$	=	=	=	=	=	=	=	x	=	=	=	=
9 $k_{poly,p=3}$	=	=	=	=	=	=	=	=	x	=	=	=
10 $k_{RBF,\gamma=0.1}$	=	=	=	=	=	=	=	=	=	x	=	=
11 $k_{RBF,\gamma=1}$	=	=	=	=	=	=	=	=	=	=	x	=
12 $k_{RBF,\gamma=10}$	=	=	=	=	=	=	=	=	=	=	=	x

	FR (ver. 2)											
	1	2	3	4	5	6	7	8	9	10	11	12
						k_Σ						
1 k_{lin}	x	=	=	+	=	=	=	=	=	+	=	=
2 $k_{poly,p=2}$	=	x	=	+	=	=	=	=	=	+	=	=
3 $k_{poly,p=3}$	=	=	x	+	=	=	=	=	=	+	=	=
4 $k_{RBF,\gamma=0.1}$	-	-	-	x	-	-	-	-	-	=	-	-
5 $k_{RBF,\gamma=1}$	=	=	=	+	x	=	=	=	=	+	=	=
6 $k_{RBF,\gamma=10}$	=	=	=	+	=	x	=	=	=	+	=	=
						k_Π						
7 k_{lin}	=	=	=	+	=	=	x	=	=	+	=	=
8 $k_{poly,p=2}$	=	=	=	+	=	=	=	x	=	+	=	=
9 $k_{poly,p=3}$	=	=	=	+	=	=	=	=	x	+	=	=
10 $k_{RBF,\gamma=0.1}$	-	-	-	=	-	-	-	-	-	x	-	-
11 $k_{RBF,\gamma=1}$	=	=	=	+	=	=	=	=	=	+	x	=
12 $k_{RBF,\gamma=10}$	=	=	=	+	=	=	=	=	=	+	=	x

Table B.13: Detailed results of McNemar's test of statistical significance for SVM in FR.

	MM (ver. 1)											
	1	2	3	4	5	6	7	8	9	10	11	12
	k_Σ											
1 k_{lin}	x	=	=	=	=	=	=	=	=	=	=	=
2 $k_{poly,p=2}$	=	x	=	=	=	=	=	=	=	=	=	=
3 $k_{poly,p=3}$	=	=	x	=	=	=	=	=	=	=	=	=
4 $k_{RBF,\gamma=0.1}$	=	=	=	x	=	=	=	=	=	=	=	=
5 $k_{RBF,\gamma=1}$	=	=	=	=	x	=	=	=	=	=	=	=
6 $k_{RBF,\gamma=10}$	=	=	=	=	=	x	=	=	=	=	=	=
	k_Π											
7 k_{lin}	=	=	=	=	=	=	x	=	=	=	=	=
8 $k_{poly,p=2}$	=	=	=	=	=	=	=	x	=	=	=	=
9 $k_{poly,p=3}$	=	=	=	=	=	=	=	=	x	=	=	=
10 $k_{RBF,\gamma=0.1}$	=	=	=	=	=	=	=	=	=	x	=	=
11 $k_{RBF,\gamma=1}$	=	=	=	=	=	=	=	=	=	=	x	=
12 $k_{RBF,\gamma=10}$	=	=	=	=	=	=	=	=	=	=	=	x

	MM (ver. 2)											
	1	2	3	4	5	6	7	8	9	10	11	12
	k_Σ											
1 k_{lin}	x	=	=	=	=	=	=	=	=	=	=	=
2 $k_{poly,p=2}$	=	x	=	=	=	=	=	=	=	=	=	=
3 $k_{poly,p=3}$	=	=	x	=	=	=	=	=	=	=	=	=
4 $k_{RBF,\gamma=0.1}$	=	=	=	x	=	=	=	=	=	=	=	=
5 $k_{RBF,\gamma=1}$	=	=	=	=	x	=	=	=	=	=	=	=
6 $k_{RBF,\gamma=10}$	=	=	=	=	=	x	=	=	=	=	=	=
	k_Π											
7 k_{lin}	=	=	=	=	=	=	x	=	=	=	=	=
8 $k_{poly,p=2}$	=	=	=	=	=	=	=	x	=	=	=	=
9 $k_{poly,p=3}$	=	=	=	=	=	=	=	=	x	=	=	=
10 $k_{RBF,\gamma=0.1}$	=	=	=	=	=	=	=	=	=	x	=	=
11 $k_{RBF,\gamma=1}$	=	=	=	=	=	=	=	=	=	=	x	=
12 $k_{RBF,\gamma=10}$	=	=	=	=	=	=	=	=	=	=	=	x

Table B.14: Detailed results of McNemar's test of statistical significance for SVM in MM.

	MR (ver. 1)											
	1	2	3	4	5	6	7	8	9	10	11	12
						k_Σ						
1 k_{lin}	x	=	=	=	=	=	=	=	=	=	=	=
2 $k_{poly,p=2}$	=	x	=	=	=	=	=	=	=	=	=	=
3 $k_{poly,p=3}$	=	=	x	=	=	=	=	=	=	=	=	=
4 $k_{RBF,\gamma=0.1}$	=	=	=	x	=	=	=	=	=	=	=	=
5 $k_{RBF,\gamma=1}$	=	=	=	=	x	=	=	=	=	=	=	=
6 $k_{RBF,\gamma=10}$	=	=	=	=	=	x	=	=	=	=	=	=
						k_Π						
7 k_{lin}	=	=	=	=	=	=	x	=	=	=	=	=
8 $k_{poly,p=2}$	=	=	=	=	=	=	=	x	=	=	=	=
9 $k_{poly,p=3}$	=	=	=	=	=	=	=	=	x	=	=	=
10 $k_{RBF,\gamma=0.1}$	=	=	=	=	=	=	=	=	=	x	=	=
11 $k_{RBF,\gamma=1}$	=	=	=	=	=	=	=	=	=	=	x	=
12 $k_{RBF,\gamma=10}$	=	=	=	=	=	=	=	=	=	=	=	x

	MR (ver. 2)											
	1	2	3	4	5	6	7	8	9	10	11	12
						k_Σ						
1 k_{lin}	x	=	=	+	=	=	=	=	=	=	=	=
2 $k_{poly,p=2}$	=	x	=	+	=	=	=	=	=	=	=	=
3 $k_{poly,p=3}$	=	=	x	+	=	=	=	=	=	+	=	=
4 $k_{RBF,\gamma=0.1}$	-	-	-	x	-	-	-	=	=	=	-	-
5 $k_{RBF,\gamma=1}$	=	=	=	+	x	=	=	=	=	+	=	=
6 $k_{RBF,\gamma=10}$	=	=	=	+	=	x	=	=	=	+	=	=
						k_Π						
7 k_{lin}	=	=	=	+	=	=	x	=	=	=	=	=
8 $k_{poly,p=2}$	=	=	=	=	=	=	=	x	=	=	=	=
9 $k_{poly,p=3}$	=	=	=	=	=	=	=	=	x	=	=	=
10 $k_{RBF,\gamma=0.1}$	=	=	-	=	-	-	=	=	=	x	=	=
11 $k_{RBF,\gamma=1}$	=	=	=	+	=	=	=	=	=	=	x	=
12 $k_{RBF,\gamma=10}$	=	=	=	+	=	=	=	=	=	=	=	x

Table B.15: Detailed results of McNemar's test of statistical significance for SVM in MR.

B.2.2 Results for CP kernel using kNN

	diterpenes											
	1	2	3	4	5	6	7	8	9	10	11	12
	$f_{norm}\cdot) = \lvert\cdot\rvert$											
1 k_{lin}	x	-	-	=	-	-	+	-	-	=	-	-
2 $k_{poly,p=2}$	+	x	=	+	-	-	+	=	=	+	-	-
3 $k_{poly,p=3}$	+	=	x	+	-	-	+	=	=	+	=	-
4 $k_{RBF,\gamma=0.1}$	=	-	-	x	-	-	+	=	-	=	-	-
5 $k_{RBF,\gamma=1}$	+	+	+	+	x	-	+	+	=	+	=	-
6 $k_{RBF,\gamma=10}$	+	+	+	+	+	x	+	+	+	+	+	=
	$f_{norm}\cdot = \sqrt{k\cdot,\cdot)}$											
7 k_{lin}	-	-	-	-	-	-	x	-	-	-	-	-
8 $k_{poly,p=2}$	+	=	=	=	-	-	+	x	-	=	-	-
9 $k_{poly,p=3}$	+	=	=	+	=	-	+	+	x	+	=	-
10 $k_{RBF,\gamma=0.1}$	=	-	-	=	-	-	+	=	-	x	-	-
11 $k_{RBF,\gamma=1}$	+	+	=	+	=	-	+	+	=	+	x	-
12 $k_{RBF,\gamma=10}$	+	+	+	+	+	=	+	+	+	+	+	x
	duke											
	1	2	3	4	5	6	7	8	9	10	11	12
	$f_{norm}\cdot) = \lvert\cdot\rvert$											
1 k_{lin}	x	=	=	=	=	=	=	=	=	=	=	=
2 $k_{poly,p=2}$	=	x	=	=	=	=	=	=	=	=	=	=
3 $k_{poly,p=3}$	=	=	x	=	=	=	=	=	=	=	=	=
4 $k_{RBF,\gamma=0.1}$	=	=	=	x	=	=	=	=	=	=	=	=
5 $k_{RBF,\gamma=1}$	=	=	=	=	x	=	=	=	=	=	=	=
6 $k_{RBF,\gamma=10}$	=	=	=	=	=	x	=	=	=	=	=	=
	$f_{norm}\cdot) = \sqrt{k\cdot,\cdot)}$											
7 k_{lin}	=	=	=	=	=	=	x	=	=	=	=	=
8 $k_{poly,p=2}$	=	=	=	=	=	=	=	x	=	=	=	=
9 $k_{poly,p=3}$	=	=	=	=	=	=	=	=	x	=	=	=
10 $k_{RBF,\gamma=0.1}$	=	=	=	=	=	=	=	=	=	x	=	=
11 $k_{RBF,\gamma=1}$	=	=	=	=	=	=	=	=	=	=	x	=
12 $k_{RBF,\gamma=10}$	=	=	=	=	=	=	=	=	=	=	=	x

Table B.16: Detailed results of McNemar's test of statistical significance for kNN in diterpenes and duke.

	musk ver. 1											
	1	2	3	4	5	6	7	8	9	10	11	12
	$f_{norm}\cdot) = \lvert\cdot\rvert$											
1 k_{lin}	x	=	=	=	+	+	=	=	=	=	=	+
2 $k_{poly,p=2}$	=	x	=	=	+	+	=	=	=	=	=	+
3 $k_{poly,p=3}$	=	=	x	=	+	+	=	=	=	=	=	+
4 $k_{RBF,\gamma=0.1}$	=	=	=	x	+	+	=	=	=	=	=	+
5 $k_{RBF,\gamma=1}$	-	-	-	-	x	+	-	-	-	-	-	=
6 $k_{RBF,\gamma=10}$	-	-	-	-	-	x	-	-	-	-	-	=
	$f_{norm}\cdot = \sqrt{k\cdot,\cdot)}$											
7 k_{lin}	=	=	=	=	+	+	x	=	=	=	=	+
8 $k_{poly,p=2}$	=	=	=	=	+	+	=	x	=	=	=	+
9 $k_{poly,p=3}$	=	=	=	=	+	+	=	=	x	=	=	+
10 $k_{RBF,\gamma=0.1}$	=	=	=	=	+	+	=	=	=	x	=	+
11 $k_{RBF,\gamma=1}$	=	=	=	=	+	+	=	=	=	=	x	+
12 $k_{RBF,\gamma=10}$	-	-	-	-	=	=	-	-	-	-	-	x

	musk ver. 2											
	1	2	3	4	5	6	7	8	9	10	11	12
	$f_{norm}(\cdot) = \lvert\cdot\rvert$											
1 k_{lin}	x	=	=	=	=	=	=	=	=	=	=	=
2 $k_{poly,p=2}$	=	x	=	=	=	=	=	=	=	=	=	=
3 $k_{poly,p=3}$	=	=	x	-	=	=	=	=	=	=	=	=
4 $k_{RBF,\gamma=0.1}$	=	=	+	x	=	=	=	+	=	=	=	=
5 $k_{RBF,\gamma=1}$	=	=	=	=	x	+	=	+	=	=	=	=
6 $k_{RBF,\gamma=10}$	=	=	=	=	-	x	=	=	=	=	=	=
	$f_{norm}(\cdot) = \sqrt{k(\cdot,\cdot)}$											
7 k_{lin}	=	=	=	=	=	=	x	=	=	=	=	=
8 $k_{poly,p=2}$	=	=	=	-	-	=	=	x	=	=	-	=
9 $k_{poly,p=3}$	=	=	=	=	=	=	=	=	x	=	=	=
10 $k_{RBF,\gamma=0.1}$	=	=	=	=	=	=	=	=	=	x	=	=
11 $k_{RBF,\gamma=1}$	=	=	=	=	=	=	=	+	=	=	x	=
12 $k_{RBF,\gamma=10}$	=	=	=	=	=	=	=	=	=	=	=	x

Table B.17: Detailed results of McNemar's test of statistical significance for kNN in musk.

	mutagenesis (ver. 1)											
	1	2	3	4	5	6	7	8	9	10	11	12
							k_Σ					
1 k_{lin}	x	=	=	=	=	=	=	=	=	=	=	=
2 $k_{poly,p=2}$	=	x	=	=	=	=	=	=	=	=	-	=
3 $k_{poly,p=3}$	=	=	x	=	=	=	=	=	=	=	=	=
4 $k_{RBF,\gamma=0.1}$	=	=	=	x	=	=	=	=	=	=	=	=
5 $k_{RBF,\gamma=1}$	=	=	=	=	x	=	=	=	=	=	=	=
6 $k_{RBF,\gamma=10}$	=	=	=	=	=	x	=	=	=	=	=	=
							k_Π					
7 k_{lin}	=	=	=	=	=	=	x	=	=	=	=	=
8 $k_{poly,p=2}$	=	=	=	=	=	=	=	x	=	=	-	=
9 $k_{poly,p=3}$	=	=	=	=	=	=	=	=	x	=	=	=
10 $k_{RBF,\gamma=0.1}$	=	=	=	=	=	=	=	=	=	x	=	=
11 $k_{RBF,\gamma=1}$	=	+	=	=	=	=	=	+	=	=	x	=
12 $k_{RBF,\gamma=10}$	=	=	=	=	=	=	=	=	=	=	=	x

	mutagenesis (ver. 2)											
	1	2	3	4	5	6	7	8	9	10	11	12
							k_Σ					
1 k_{lin}	x	=	=	=	=	=	=	=	=	=	=	=
2 $k_{poly,p=2}$	=	x	=	=	=	=	=	=	=	=	=	=
3 $k_{poly,p=3}$	=	=	x	=	=	=	=	=	=	=	=	=
4 $k_{RBF,\gamma=0.1}$	=	=	=	x	=	=	=	=	=	=	=	=
5 $k_{RBF,\gamma=1}$	=	=	=	=	x	=	=	=	=	=	=	=
6 $k_{RBF,\gamma=10}$	=	=	=	=	=	x	=	=	=	=	=	=
							k_Π					
7 k_{lin}	=	=	=	=	=	=	x	=	=	=	=	=
8 $k_{poly,p=2}$	=	=	=	=	=	=	=	x	=	=	=	=
9 $k_{poly,p=3}$	=	=	=	=	=	=	=	=	x	=	=	=
10 $k_{RBF,\gamma=0.1}$	=	=	=	=	=	=	=	=	=	x	=	=
11 $k_{RBF,\gamma=1}$	=	=	=	=	=	=	=	=	=	=	x	=
12 $k_{RBF,\gamma=10}$	=	=	=	=	=	=	=	=	=	=	=	x

Table B.18: Detailed results of McNemar's test of statistical significance for kNN in mutagenesis.

	FM (ver. 1)											
	1	2	3	4	5	6	7	8	9	10	11	12
	k_Σ											
1 k_{lin}	x	=	=	=	=	=	=	=	+	=	+	=
2 $k_{poly,p=2}$	=	x	=	=	=	=	=	=	+	=	+	=
3 $k_{poly,p=3}$	=	=	x	=	=	=	=	=	=	=	+	=
4 $k_{RBF,\gamma=0.1}$	=	=	=	x	=	=	=	=	=	=	+	=
5 $k_{RBF,\gamma=1}$	=	=	=	=	x	=	=	=	=	=	+	=
6 $k_{RBF,\gamma=10}$	=	=	=	=	=	x	=	=	=	=	=	=
	k_Π											
7 k_{lin}	=	=	=	=	=	=	x	=	=	=	=	=
8 $k_{poly,p=2}$	=	=	=	=	=	=	=	x	=	=	=	=
9 $k_{poly,p=3}$	-	-	=	=	=	=	=	=	x	=	=	=
10 $k_{RBF,\gamma=0.1}$	=	=	=	=	=	=	=	=	=	x	=	=
11 $k_{RBF,\gamma=1}$	-	-	-	-	-	=	=	=	=	=	x	=
12 $k_{RBF,\gamma=10}$	=	=	=	=	=	=	=	=	=	=	=	x

	FM (ver. 2)											
	1	2	3	4	5	6	7	8	9	10	11	12
	k_Σ											
1 k_{lin}	x	=	=	=	=	=	=	=	=	=	=	=
2 $k_{poly,p=2}$	=	x	=	=	=	=	=	=	=	=	=	=
3 $k_{poly,p=3}$	=	=	x	=	=	+	=	=	=	=	=	=
4 $k_{RBF,\gamma=0.1}$	=	=	=	x	=	=	=	=	=	=	=	=
5 $k_{RBF,\gamma=1}$	=	=	=	=	x	=	=	=	=	=	=	=
6 $k_{RBF,\gamma=10}$	=	=	-	=	=	x	=	=	=	=	=	=
	k_Π											
7 k_{lin}	=	=	=	=	=	=	x	=	=	=	=	=
8 $k_{poly,p=2}$	=	=	=	=	=	=	=	x	=	=	=	=
9 $k_{poly,p=3}$	=	=	=	=	=	=	=	=	x	=	=	=
10 $k_{RBF,\gamma=0.1}$	=	=	=	=	=	=	=	=	=	x	=	=
11 $k_{RBF,\gamma=1}$	=	=	=	=	=	=	=	=	=	=	x	=
12 $k_{RBF,\gamma=10}$	=	=	=	=	=	=	=	=	=	=	=	x

Table B.19: Detailed results of McNemar's test of statistical significance for kNN in FM.

	FR (ver. 1)											
	1	2	3	4	5	6	7	8	9	10	11	12
	k_Σ											
1 k_{lin}	x	=	=	=	=	=	=	=	=	=	=	=
2 $k_{poly,p=2}$	=	x	=	=	=	=	=	=	=	=	=	=
3 $k_{poly,p=3}$	=	=	x	=	=	=	=	=	=	=	=	=
4 $k_{RBF,\gamma=0.1}$	=	=	=	x	=	=	=	=	=	=	=	=
5 $k_{RBF,\gamma=1}$	=	=	=	=	x	=	=	=	=	=	=	=
6 $k_{RBF,\gamma=10}$	=	=	=	=	=	x	=	=	=	=	=	=
	k_Π											
7 k_{lin}	=	=	=	=	=	=	x	=	=	=	=	=
8 $k_{poly,p=2}$	=	=	=	=	=	=	=	x	=	=	=	=
9 $k_{poly,p=3}$	=	=	=	=	=	=	=	=	x	=	=	=
10 $k_{RBF,\gamma=0.1}$	=	=	=	=	=	=	=	=	=	x	=	=
11 $k_{RBF,\gamma=1}$	=	=	=	=	=	=	=	=	=	=	x	=
12 $k_{RBF,\gamma=10}$	=	=	=	=	=	=	=	=	=	=	=	x

	FR (ver. 2)											
	1	2	3	4	5	6	7	8	9	10	11	12
	k_Σ											
1 k_{lin}	x	=	=	=	=	=	=	=	=	=	=	=
2 $k_{poly,p=2}$	=	x	=	=	=	=	=	=	=	=	=	=
3 $k_{poly,p=3}$	=	=	x	=	=	=	=	=	=	=	=	=
4 $k_{RBF,\gamma=0.1}$	=	=	=	x	=	=	=	=	=	=	=	-
5 $k_{RBF,\gamma=1}$	=	=	=	=	x	=	=	=	=	=	=	=
6 $k_{RBF,\gamma=10}$	=	=	=	=	=	x	=	=	=	=	=	=
	k_Π											
7 k_{lin}	=	=	=	=	=	=	x	=	=	=	=	=
8 $k_{poly,p=2}$	=	=	=	=	=	=	=	x	=	=	=	=
9 $k_{poly,p=3}$	=	=	=	=	=	=	=	=	x	=	=	+
10 $k_{RBF,\gamma=0.1}$	=	=	=	=	=	=	=	=	=	x	=	=
11 $k_{RBF,\gamma=1}$	=	=	=	=	=	=	=	=	=	=	x	=
12 $k_{RBF,\gamma=10}$	=	=	=	=	=	=	=	=	-	=	=	x

Table B.20: Detailed results of McNemar's test of statistical significance for kNN in FR.

	MM (ver. 1)											
	1	2	3	4	5	6	7	8	9	10	11	12
							k_Σ					
1 k_{lin}	x	=	=	=	=	=	+	=	=	=	=	=
2 $k_{poly,p=2}$	=	x	=	=	=	=	+	=	=	=	=	=
3 $k_{poly,p=3}$	=	=	x	=	=	=	=	=	=	=	=	=
4 $k_{RBF,\gamma=0.1}$	=	=	=	x	=	=	=	=	=	=	=	=
5 $k_{RBF,\gamma=1}$	=	=	=	=	x	=	=	=	=	=	=	=
6 $k_{RBF,\gamma=10}$	=	=	=	=	=	x	=	=	=	=	=	=
							k_Π					
7 k_{lin}	-	-	=	=	=	=	x	=	=	=	=	=
8 $k_{poly,p=2}$	=	=	=	=	=	=	=	x	=	=	=	=
9 $k_{poly,p=3}$	=	=	=	=	=	=	=	=	x	=	=	=
10 $k_{RBF,\gamma=0.1}$	=	=	=	=	=	=	=	=	=	x	=	=
11 $k_{RBF,\gamma=1}$	=	=	=	=	=	=	=	=	=	=	x	=
12 $k_{RBF,\gamma=10}$	=	=	=	=	=	=	=	=	=	=	=	x

	MM (ver. 2)											
	1	2	3	4	5	6	7	8	9	10	11	12
							k_Σ					
1 k_{lin}	x	=	=	=	=	=	=	=	=	=	=	=
2 $k_{poly,p=2}$	=	x	=	=	+	+	=	+	+	+	+	+
3 $k_{poly,p=3}$	=	=	x	=	+	=	=	=	=	=	=	=
4 $k_{RBF,\gamma=0.1}$	=	=	=	x	+	+	=	=	+	=	+	=
5 $k_{RBF,\gamma=1}$	=	-	-	-	x	=	=	=	=	=	=	=
6 $k_{RBF,\gamma=10}$	=	-	=	-	=	x	=	=	=	=	=	=
							k_Π					
7 k_{lin}	=	=	=	=	=	=	x	=	=	=	=	=
8 $k_{poly,p=2}$	=	-	=	=	=	=	=	x	=	=	=	=
9 $k_{poly,p=3}$	=	-	=	-	=	=	=	=	x	=	=	=
10 $k_{RBF,\gamma=0.1}$	=	-	=	=	=	=	=	=	=	x	=	=
11 $k_{RBF,\gamma=1}$	=	-	=	-	=	=	=	=	=	=	x	=
12 $k_{RBF,\gamma=10}$	=	-	=	=	=	=	=	=	=	=	=	x

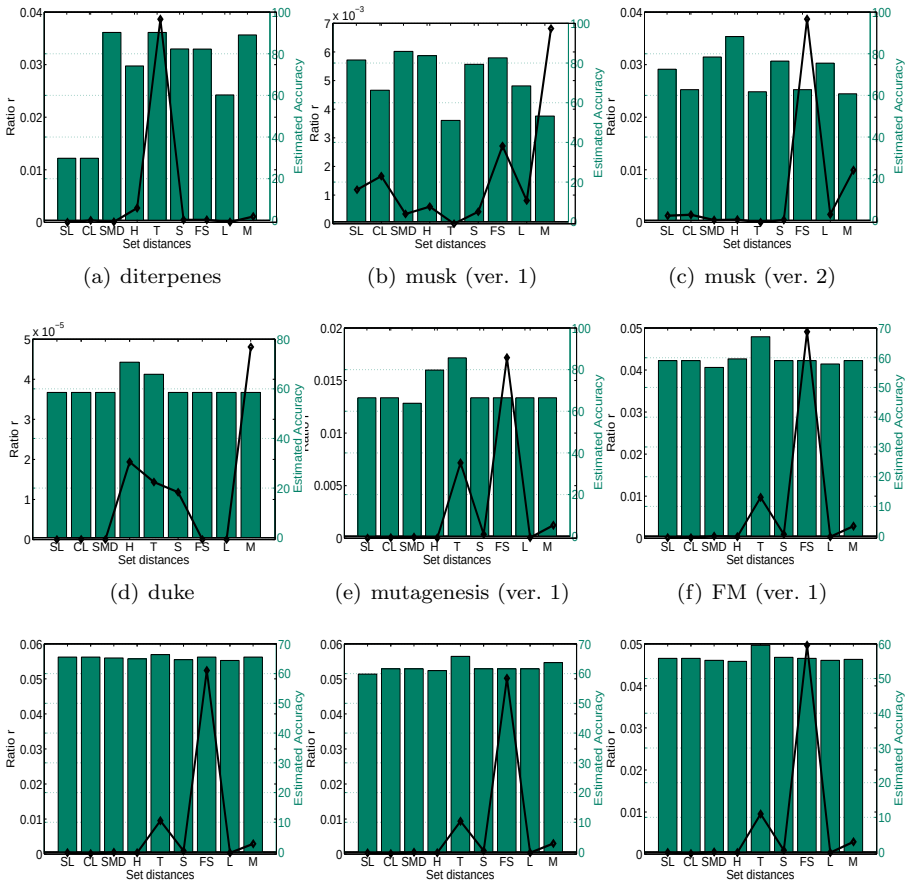
Table B.21: Detailed results of McNemar's test of statistical significance for kNN in MM.

	MR (ver. 1)											
	1	2	3	4	5	6	7	8	9	10	11	12
							k_Σ					
1 k_{lin}	x	=	=	=	=	=	=	=	=	=	=	-
2 $k_{poly,p=2}$	=	x	=	=	=	=	=	=	=	=	=	=
3 $k_{poly,p=3}$	=	=	x	=	=	=	=	=	=	=	=	-
4 $k_{RBF,\gamma=0.1}$	=	=	=	x	=	=	=	=	=	=	=	-
5 $k_{RBF,\gamma=1}$	=	=	=	=	x	=	=	=	=	=	=	=
6 $k_{RBF,\gamma=10}$	=	=	=	=	=	x	=	=	=	=	=	-
							k_Π					
7 k_{lin}	=	=	=	=	=	=	x	=	=	=	=	=
8 $k_{poly,p=2}$	=	=	=	=	=	=	=	x	=	=	=	=
9 $k_{poly,p=3}$	=	=	=	=	=	=	=	=	x	=	=	=
10 $k_{RBF,\gamma=0.1}$	=	=	=	=	=	=	=	=	=	x	=	=
11 $k_{RBF,\gamma=1}$	=	=	=	=	=	=	=	=	=	=	x	=
12 $k_{RBF,\gamma=10}$	+	=	+	+	=	+	=	=	=	=	=	x

	MR (ver. 2)											
	1	2	3	4	5	6	7	8	9	10	11	12
							k_Σ					
1 k_{lin}	x	=	=	+	=	=	=	=	=	=	=	=
2 $k_{poly,p=2}$	=	x	=	=	=	=	-	-	=	=	-	-
3 $k_{poly,p=3}$	=	=	x	=	=	=	-	-	=	=	-	=
4 $k_{RBF,\gamma=0.1}$	-	=	=	x	=	=	-	-	-	-	-	-
5 $k_{RBF,\gamma=1}$	=	=	=	=	x	=	-	-	=	=	-	-
6 $k_{RBF,\gamma=10}$	=	=	=	=	=	x	-	-	=	=	-	-
							k_Π					
7 k_{lin}	=	+	+	+	+	+	x	=	=	=	=	=
8 $k_{poly,p=2}$	=	+	+	+	+	+	=	x	=	=	=	=
9 $k_{poly,p=3}$	=	=	=	+	=	=	=	=	x	=	=	=
10 $k_{RBF,\gamma=0.1}$	=	=	=	+	=	=	=	=	=	x	=	=
11 $k_{RBF,\gamma=1}$	=	+	+	+	+	+	=	=	=	=	x	=
12 $k_{RBF,\gamma=10}$	=	+	=	+	+	+	=	=	=	=	=	x

Table B.22: Detailed results of McNemar's test of statistical significance for kNN in MR.

B.3 Spectras of Gram Matrices

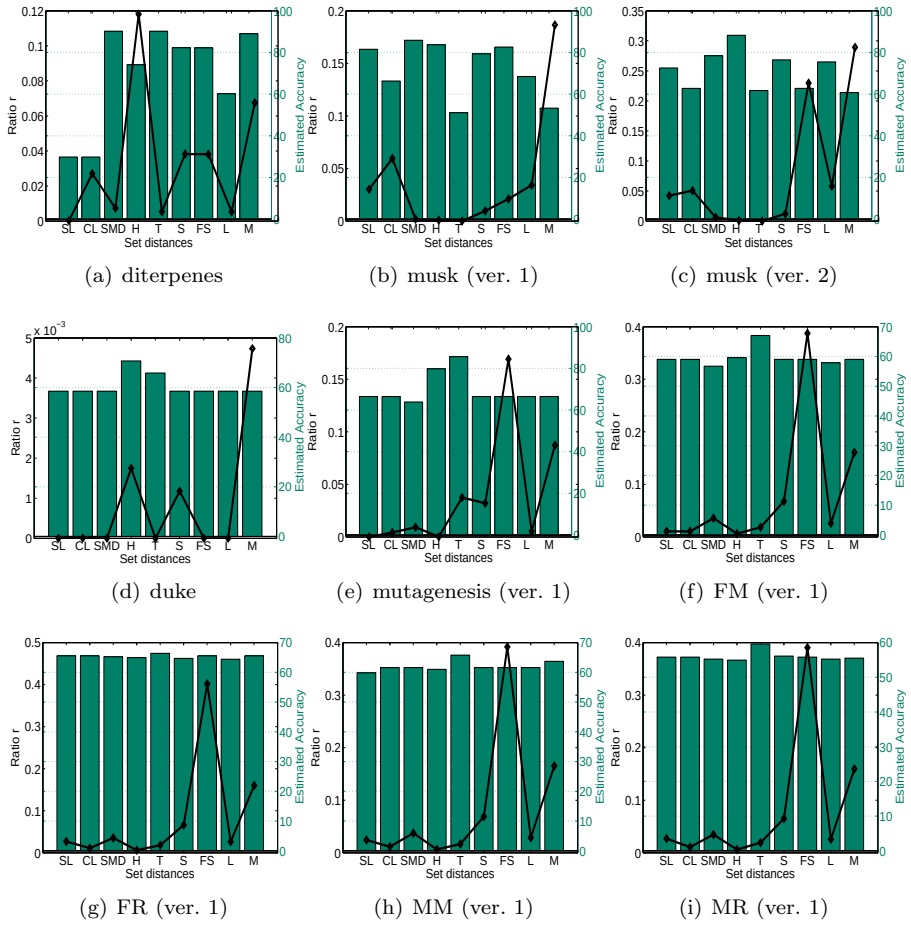


(a) diterpenes (b) musk (ver. 1) (c) musk (ver. 2)

(d) duke (e) mutagenesis (ver. 1) (f) FM (ver. 1)

(g) FR (ver. 1) (h) MM (ver. 1) (i) MR (ver. 1)

Figure B.1: Spectra of Gram matrices of set distance substitution kernels. The solid line denotes the r ratio from Equation 4.35 (left y-axis) and bars denote the estimated accuracies (right y-axis). The γ parameter is fixed to 0.1.



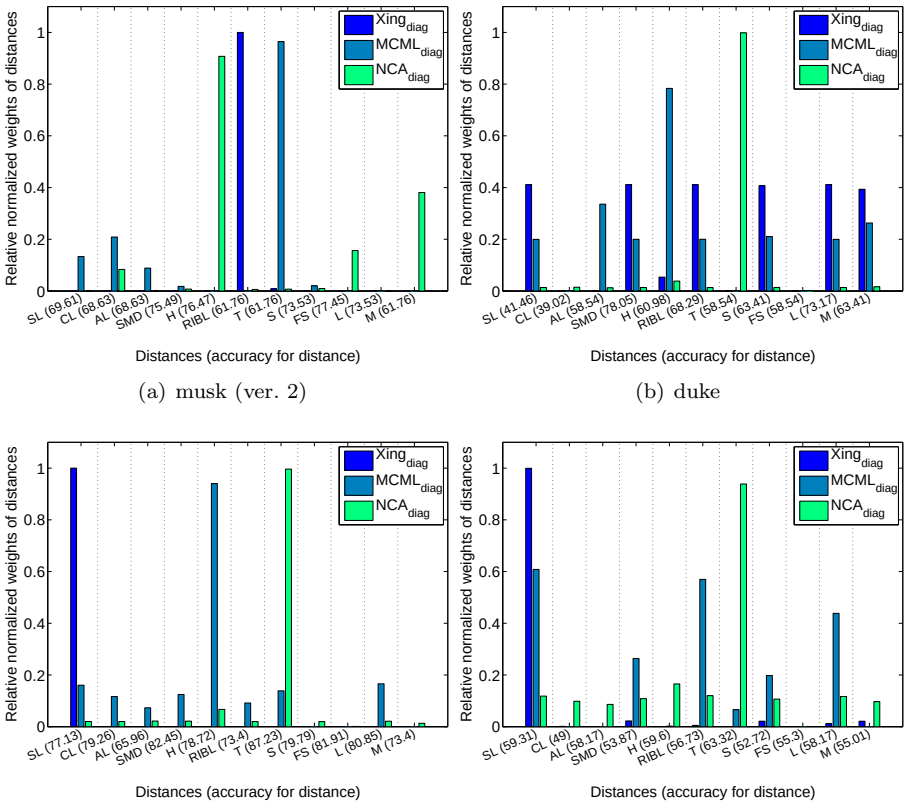
(a) diterpenes (b) musk (ver. 1) (c) musk (ver. 2)

(d) duke (e) mutagenesis (ver. 1) (f) FM (ver. 1)

(g) FR (ver. 1) (h) MM (ver. 1) (i) MR (ver. 1)

Figure B.2: Spectra of Gram matrices of set distance substitution kernels. The solid line denotes the r ratio from Equation 4.35 (left y-axis) and bars denote the estimated accuracies (right y-axis). The γ parameter is fixed to 10.

B.4 Visualizations of distance combinations



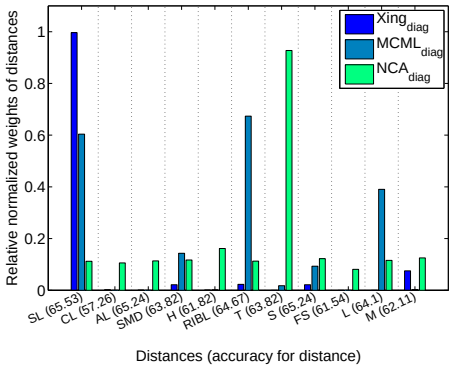
(a) musk (ver. 2)

(b) duke

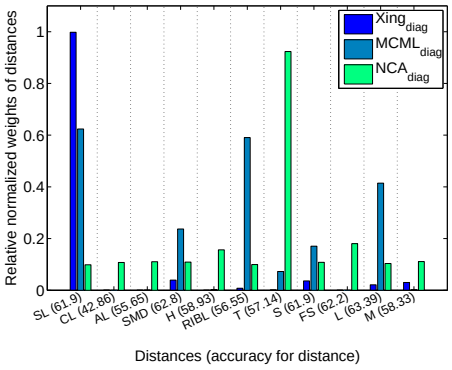
(c) mutagenesis (ver. 1)

(d) FM (ver. 1)

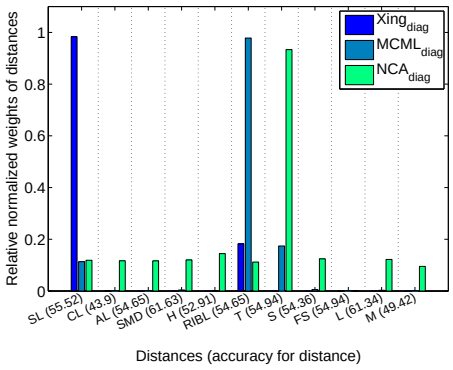
Figure B.3: Relative importance of the different set distance measures, computed by $\text{METHOD}_{\text{diag}}$, in musk (ver. 2), duke, mutagenesis (ver. 1) and FM (ver. 1) datasets. The weights are computed on the full training set. For each set distance measure we also give the corresponding performance of kNN in parenthesis.



(a) FR (ver. 1)

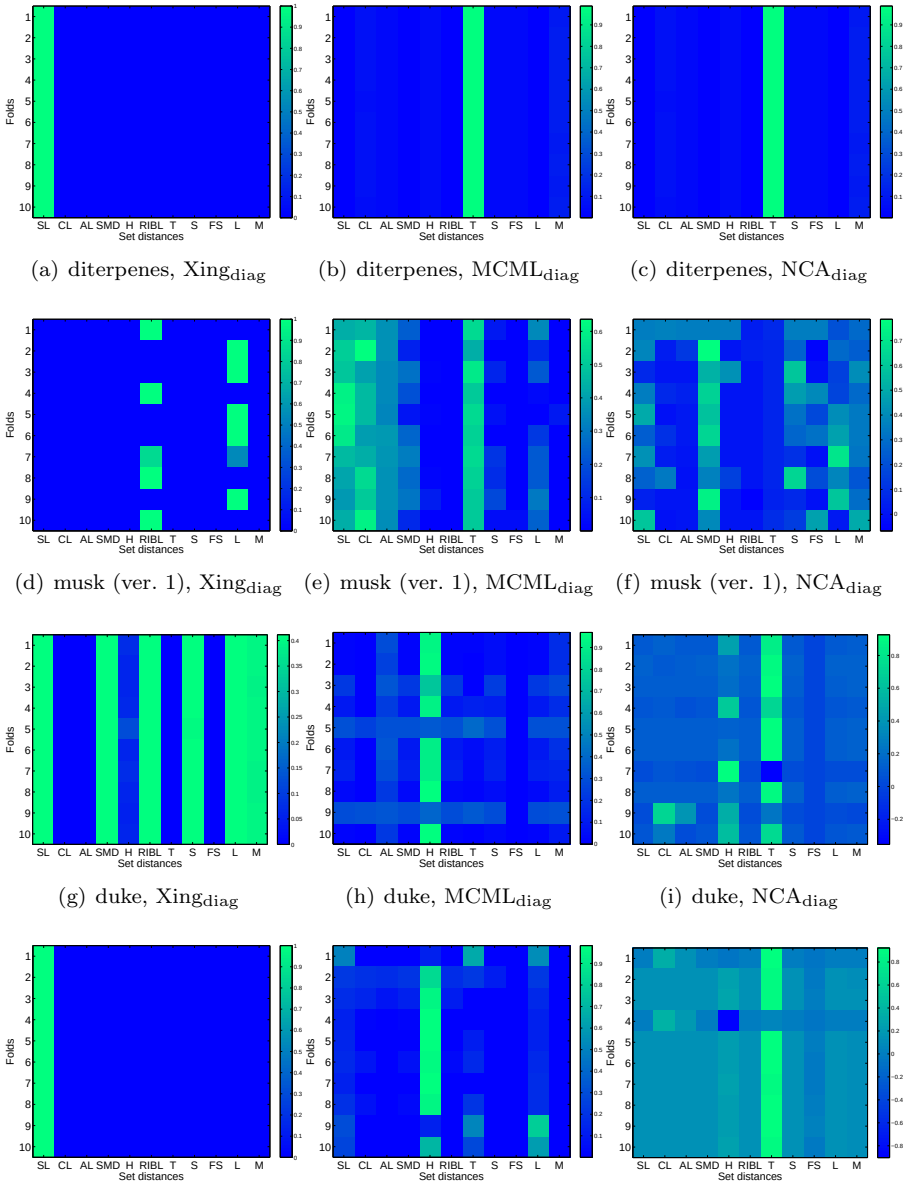


(b) MM (ver. 1)



(c) MR (ver. 1)

Figure B.4: Relative importance of the different set distance measures, computed by $\text{METHOD}_{\text{diag}}$, in FR (ver. 1), MM (ver. 1) and MR (ver. 1) datasets. The weights are computed on the full training set. For each set distance measure we also give the corresponding performance of kNN in parenthesis.



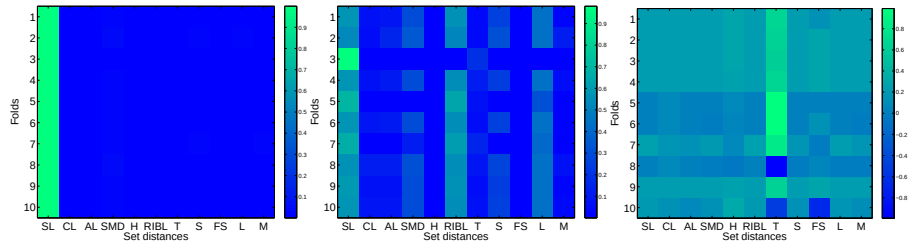
(a) diterpenes, $\text{Xing}_{\text{diag}}$ (b) diterpenes, $\text{MCML}_{\text{diag}}$ (c) diterpenes, NCA_{diag}

(d) musk (ver. 1), $\text{Xing}_{\text{diag}}$ (e) musk (ver. 1), $\text{MCML}_{\text{diag}}$ (f) musk (ver. 1), NCA_{diag}

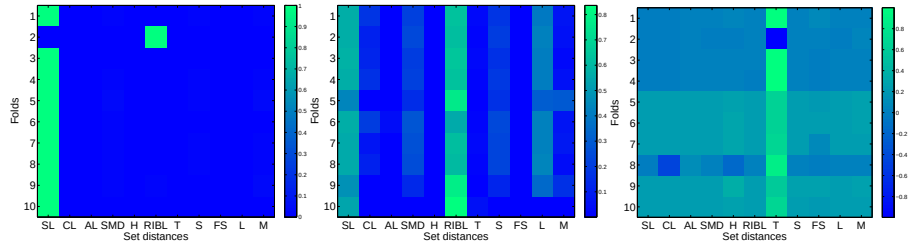
(g) duke, $\text{Xing}_{\text{diag}}$ (h) duke, $\text{MCML}_{\text{diag}}$ (i) duke, NCA_{diag}

(j) muta (ver. 1), $\text{Xing}_{\text{diag}}$ (k) muta (ver. 1), $\text{MCML}_{\text{diag}}$ (l) muta (ver. 1), NCA_{diag}

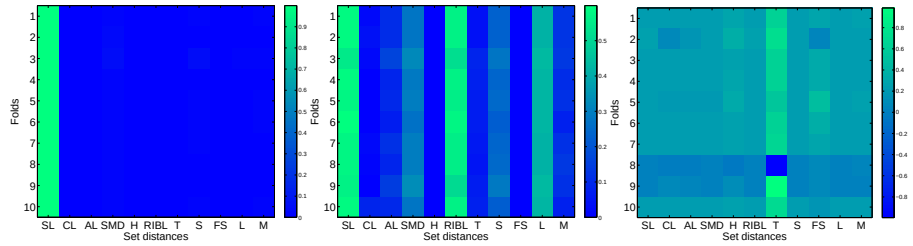
Figure B.5: Stability of the distance combinations techniques for diterpenes, musk (ver. 1), duke and mutagenesis (ver. 1). Weights in each of the 10 folds (i.e. rows) are normalized by a Frobenius norm of $\boldsymbol{A}$ ($\boldsymbol{W}$).



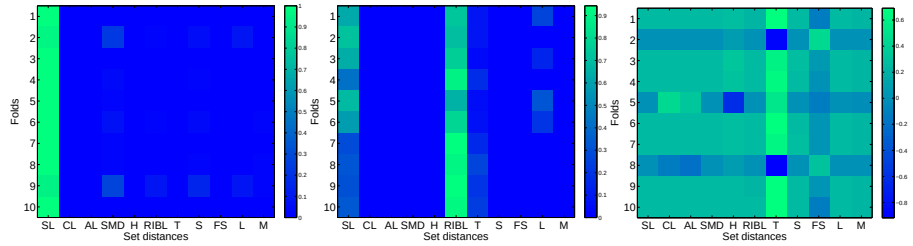
(a) FM (ver. 1), $\text{Xing}_{\text{diag}}$ (b) FM (ver. 1), $\text{MCML}_{\text{diag}}$ (c) FM (ver. 1), NCA_{diag}



(d) FR (ver. 1), $\text{Xing}_{\text{diag}}$ (e) FR (ver. 1), $\text{MCML}_{\text{diag}}$ (f) FR (ver. 1), NCA_{diag}

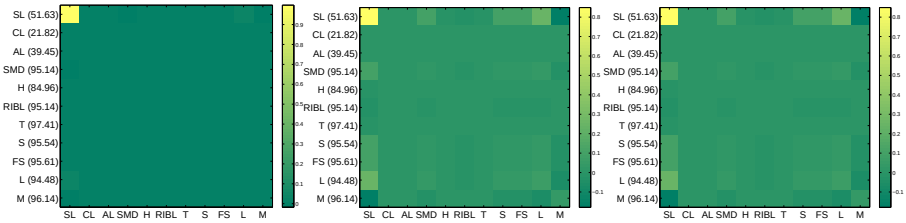


(g) MM (ver. 1), $\text{Xing}_{\text{diag}}$ (h) MM (ver. 1), $\text{MCML}_{\text{diag}}$ (i) MM (ver. 1), NCA_{diag}



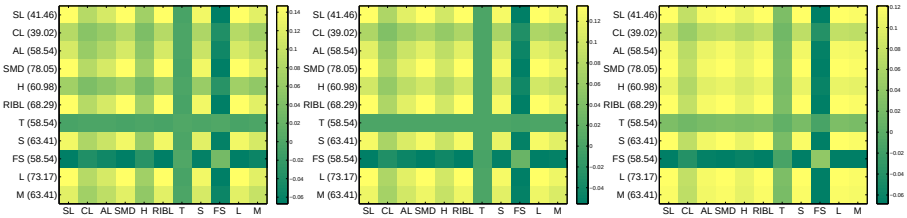
(j) MR (ver. 1), $\text{Xing}_{\text{diag}}$ (k) MR (ver. 1), $\text{MCML}_{\text{diag}}$ (l) MR (ver. 1), NCA_{diag}

Figure B.6: Stability of the distance combinations techniques in FM (ver. 1), FR (ver. 1), MM (ver. 1) and MR (ver. 1). Weights in each of the 10 folds (i.e. rows) are normalized by a Frobenius norm of $\boldsymbol{A}$ ($\boldsymbol{W}$).

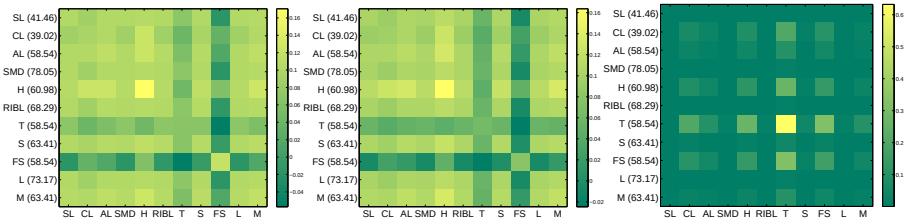


(a) $\text{Xing}_{\text{full}}$, $\lambda = 0$ (b) $\text{Xing}_{\text{full}}$, $\lambda = 1$ (c) $\text{Xing}_{\text{full}}$, $\lambda = 10$

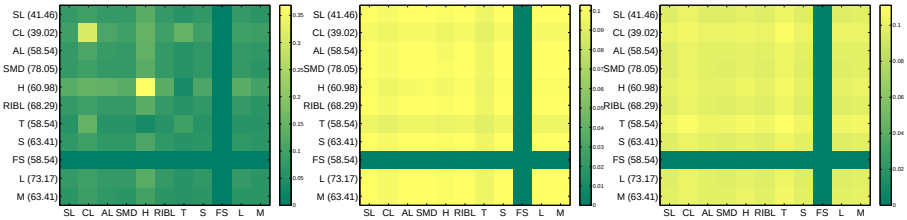
Figure B.7: Relative importance of the different set distance measures as these are computed by $\text{Xing}_{\text{full}}$, for different values of λ. The weights are computed in the diterpenes dataset and on the full training set.



(a) $\text{Xing}_{\text{full}}$, $\lambda = 0$ (b) $\text{Xing}_{\text{full}}$, $\lambda = 1$ (c) $\text{Xing}_{\text{full}}$, $\lambda = 10$

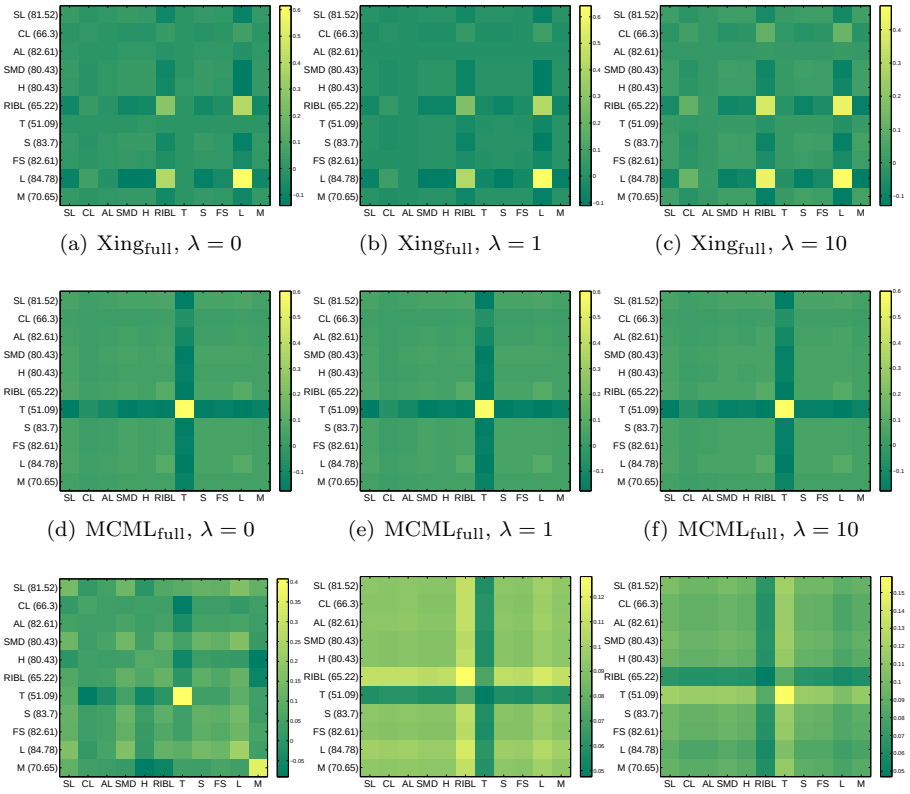


(d) $\text{MCML}_{\text{full}}$, $\lambda = 0$ (e) $\text{MCML}_{\text{full}}$, $\lambda = 1$ (f) $\text{MCML}_{\text{full}}$, $\lambda = 10$



(g) NCA_{full}, $\lambda = 0$ (h) NCA_{full}, $\lambda = 1$ (i) NCA_{full}, $\lambda = 10$

Figure B.8: Relative importance of the different set distance measures as these are computed by $\text{METHOD}_{\text{full}}$, for different values of λ. The weights are computed in the duke dataset and on the full training set.

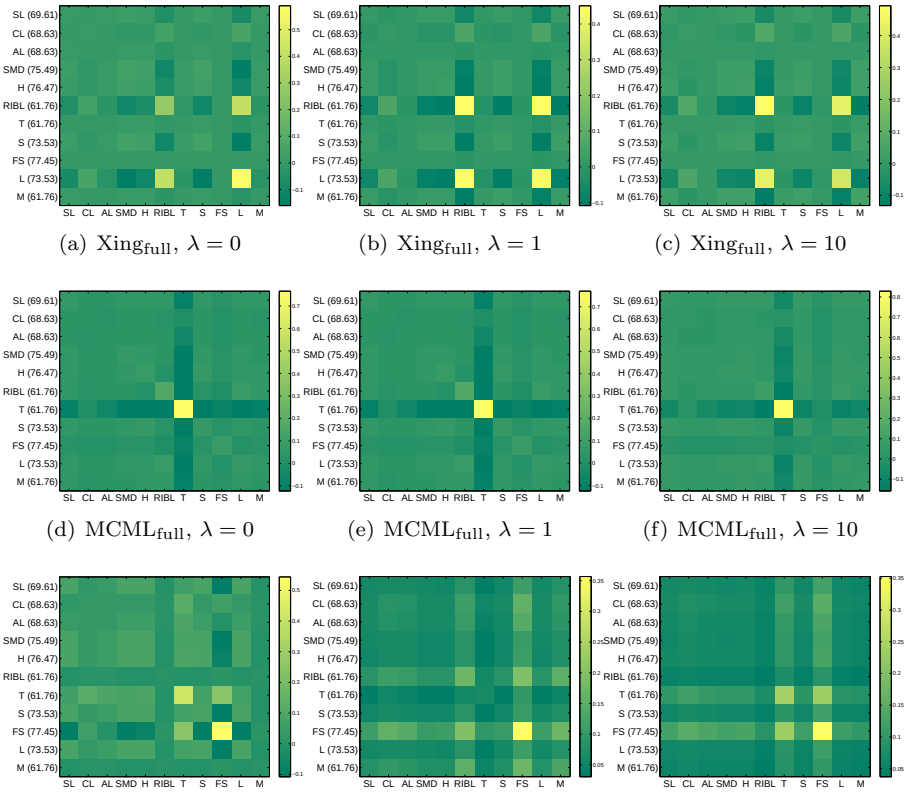


(a) $\text{Xing}_{\text{full}}$, $\lambda = 0$ (b) $\text{Xing}_{\text{full}}$, $\lambda = 1$ (c) $\text{Xing}_{\text{full}}$, $\lambda = 10$

(d) $\text{MCML}_{\text{full}}$, $\lambda = 0$ (e) $\text{MCML}_{\text{full}}$, $\lambda = 1$ (f) $\text{MCML}_{\text{full}}$, $\lambda = 10$

(g) NCA_{full}, $\lambda = 0$ (h) NCA_{full}, $\lambda = 1$ (i) NCA_{full}, $\lambda = 10$

Figure B.9: Relative importance of the different set distance measures as these are computed by $\text{METHOD}_{\text{full}}$, for different values of λ. The weights are computed in musk (ver. 1) and on the full training set.

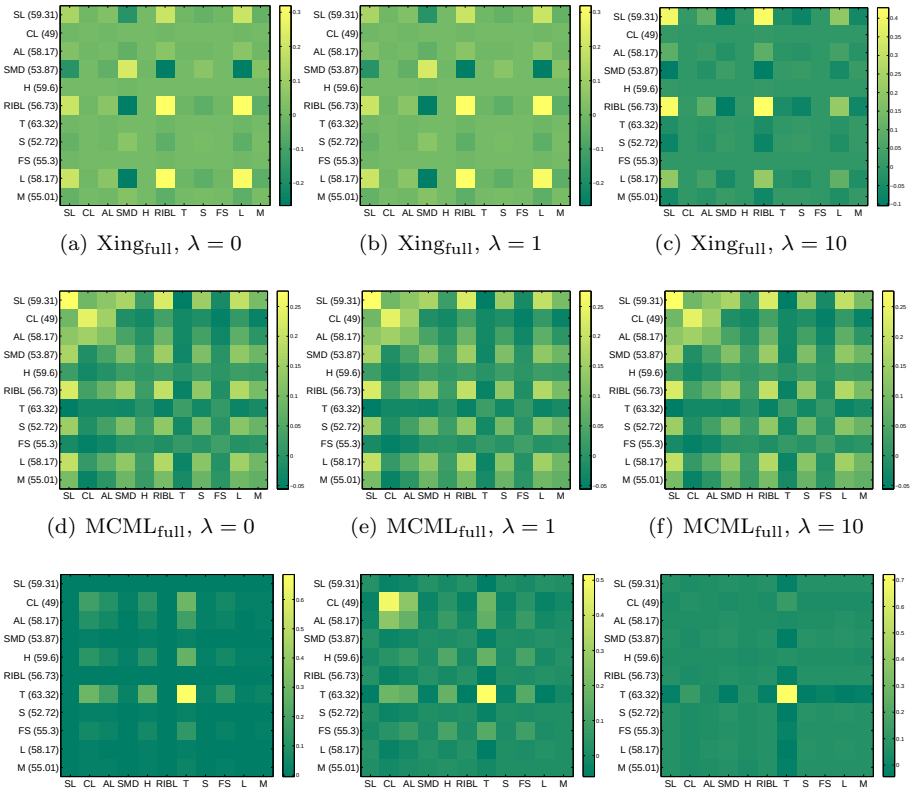


(a) $\text{Xing}_{\text{full}}$, $\lambda = 0$ (b) $\text{Xing}_{\text{full}}$, $\lambda = 1$ (c) $\text{Xing}_{\text{full}}$, $\lambda = 10$

(d) $\text{MCML}_{\text{full}}$, $\lambda = 0$ (e) $\text{MCML}_{\text{full}}$, $\lambda = 1$ (f) $\text{MCML}_{\text{full}}$, $\lambda = 10$

(g) NCA_{full}, $\lambda = 0$ (h) NCA_{full}, $\lambda = 1$ (i) NCA_{full}, $\lambda = 10$

Figure B.10: Relative importance of the different set distance measures as these are computed by $\text{METHOD}_{\text{full}}$, for different values of λ. The weights are computed in musk (ver. 2) and on the full training set.

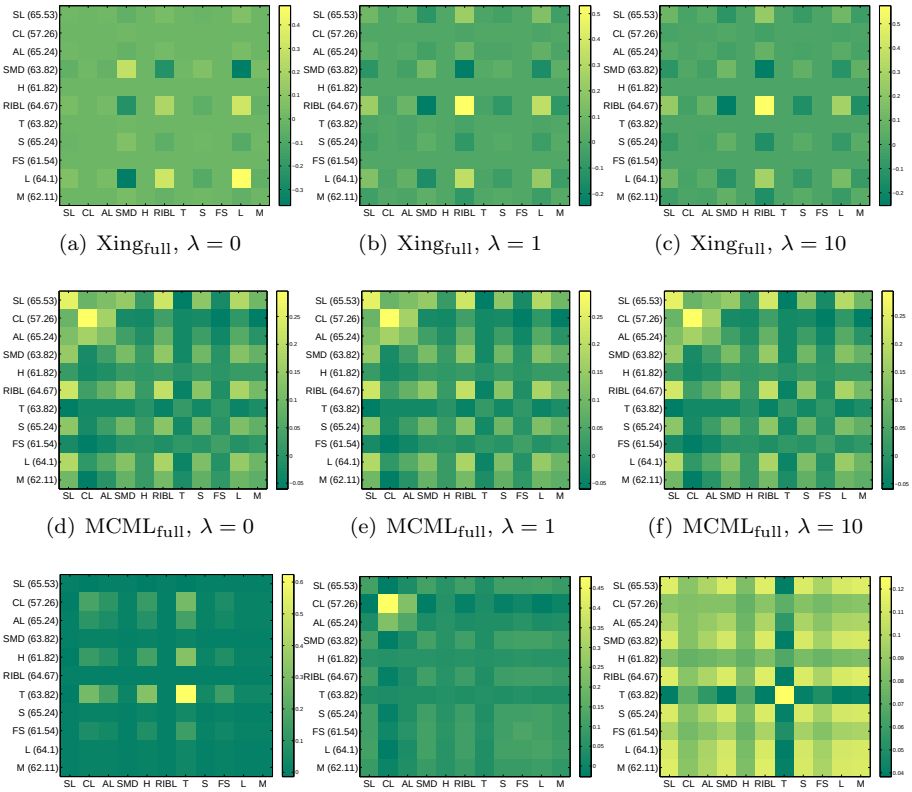


(a) $\text{Xing}_{\text{full}}$, $\lambda = 0$ (b) $\text{Xing}_{\text{full}}$, $\lambda = 1$ (c) $\text{Xing}_{\text{full}}$, $\lambda = 10$

(d) $\text{MCML}_{\text{full}}$, $\lambda = 0$ (e) $\text{MCML}_{\text{full}}$, $\lambda = 1$ (f) $\text{MCML}_{\text{full}}$, $\lambda = 10$

(g) NCA_{full}, $\lambda = 0$ (h) NCA_{full}, $\lambda = 1$ (i) NCA_{full}, $\lambda = 10$

Figure B.11: Relative importance of the different set distance measures as these are computed by $\text{METHOD}_{\text{full}}$, for different values of λ. The weights are computed in FM (ver. 1) and on the full training set.

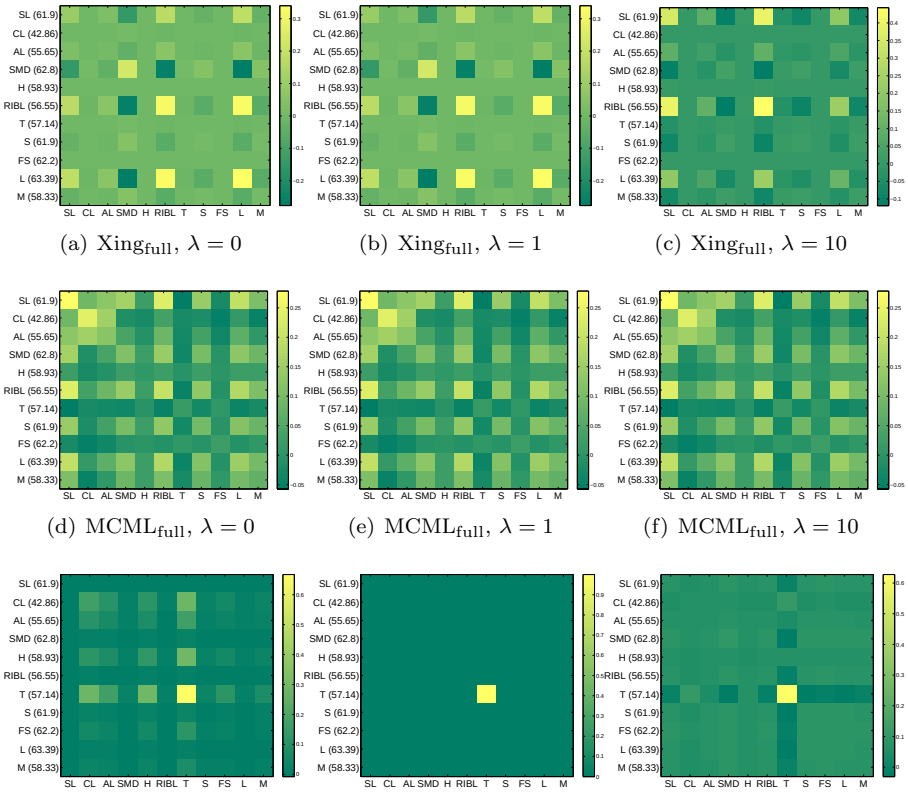


(a) $\text{Xing}_{\text{full}}$, $\lambda = 0$ (b) $\text{Xing}_{\text{full}}$, $\lambda = 1$ (c) $\text{Xing}_{\text{full}}$, $\lambda = 10$

(d) $\text{MCML}_{\text{full}}$, $\lambda = 0$ (e) $\text{MCML}_{\text{full}}$, $\lambda = 1$ (f) $\text{MCML}_{\text{full}}$, $\lambda = 10$

(g) NCA_{full}, $\lambda = 0$ (h) NCA_{full}, $\lambda = 1$ (i) NCA_{full}, $\lambda = 10$

Figure B.12: Relative importance of the different set distance measures as these are computed by $\text{METHOD}_{\text{full}}$, for different values of λ. The weights are computed in FR (ver. 1) and on the full training set.

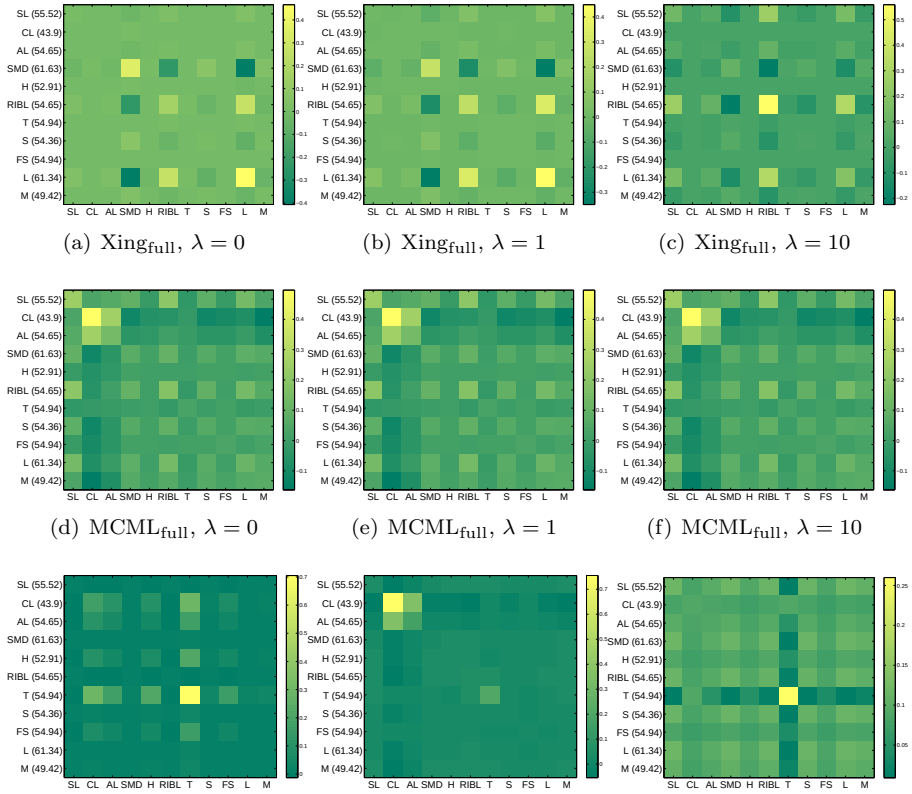


(a) $\text{Xing}_{\text{full}}$, $\lambda = 0$ (b) $\text{Xing}_{\text{full}}$, $\lambda = 1$ (c) $\text{Xing}_{\text{full}}$, $\lambda = 10$

(d) $\text{MCML}_{\text{full}}$, $\lambda = 0$ (e) $\text{MCML}_{\text{full}}$, $\lambda = 1$ (f) $\text{MCML}_{\text{full}}$, $\lambda = 10$

(g) NCA_{full}, $\lambda = 0$ (h) NCA_{full}, $\lambda = 1$ (i) NCA_{full}, $\lambda = 10$

Figure B.13: Relative importance of the different set distance measures as these are computed by $\text{METHOD}_{\text{full}}$, for different values of λ. The weights are computed in MM (ver. 1) and on the full training set.

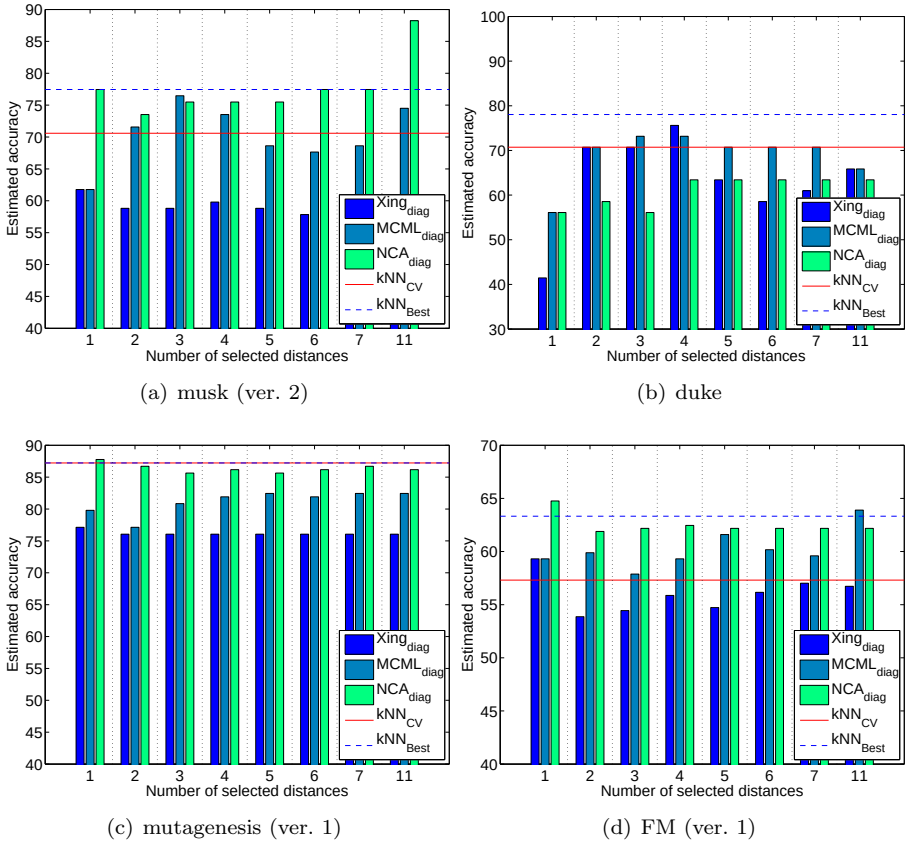


(a) $\text{Xing}_{\text{full}}$, $\lambda = 0$ (b) $\text{Xing}_{\text{full}}$, $\lambda = 1$ (c) $\text{Xing}_{\text{full}}$, $\lambda = 10$

(d) $\text{MCML}_{\text{full}}$, $\lambda = 0$ (e) $\text{MCML}_{\text{full}}$, $\lambda = 1$ (f) $\text{MCML}_{\text{full}}$, $\lambda = 10$

(g) NCA_{full}, $\lambda = 0$ (h) NCA_{full}, $\lambda = 1$ (i) NCA_{full}, $\lambda = 10$

Figure B.14: Relative importance of the different set distance measures as these are computed by $\text{METHOD}_{\text{full}}$, for different values of λ. The weights are computed in MR (ver. 1) and on the full training set.



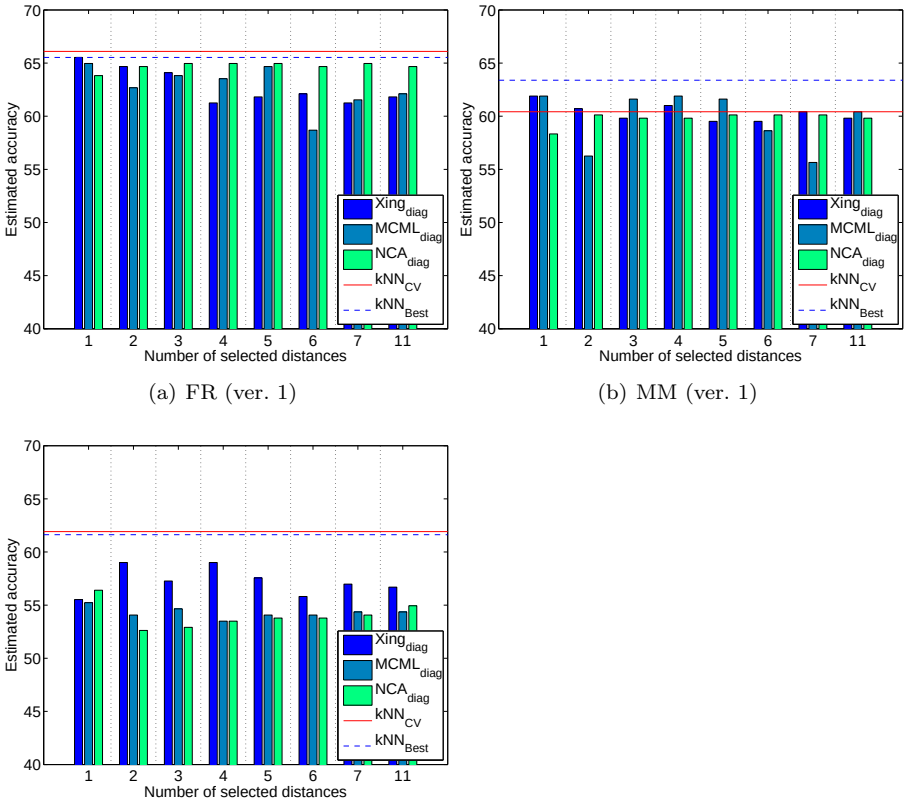
(a) musk (ver. 2)

(b) duke

(c) mutagenesis (ver. 1)

(d) FM (ver. 1)

Figure B.15: The estimated accuracy in musk (ver. 2), duke, mutagenesis (ver. 1) and FM (ver. 1) for $\text{Xing}_{\text{diag}}$, $\text{MCML}_{\text{diag}}$ and NCA_{diag} (using kNN) where $l = 1, \ldots, 7, 11$ top ranked distance measures (according to the assigned coefficients) are combined for computing the actual distance. Additionally, the performances of kNN_{Best} and kNN_{CV} are given.



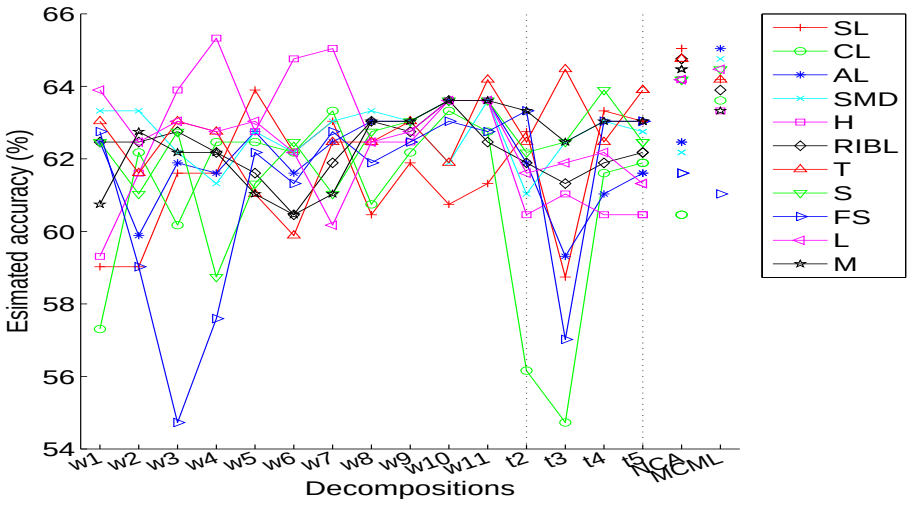
(a) FR (ver. 1)

(b) MM (ver. 1)

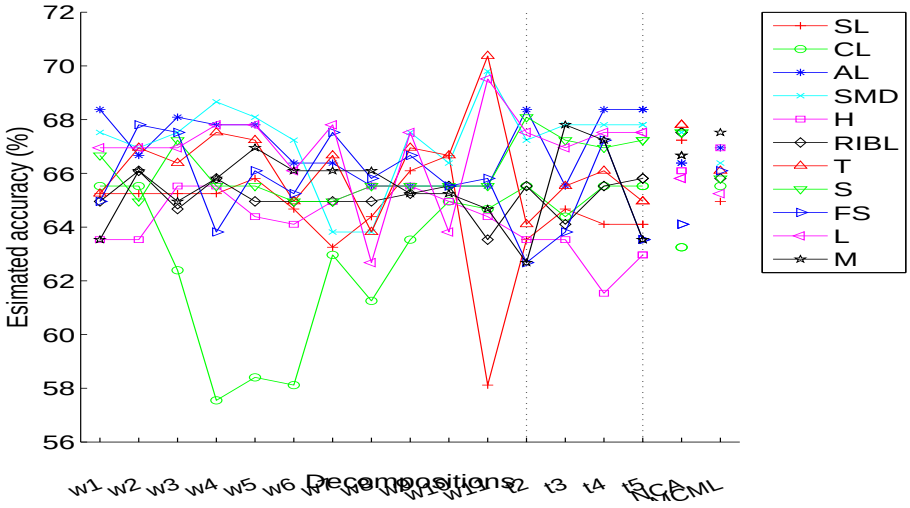
(c) MR (ver. 1)

Figure B.16: The estimated accuracy in FR (ver. 1), MM (ver. 1) and MR (ver. 1) for $\text{Xing}_{\text{diag}}$, $\text{MCML}_{\text{diag}}$ and NCA_{diag} (using kNN) where $l = 1, \ldots, 7, 11$ top ranked distance measures (according to the assigned coefficients) are combined for computing the actual distance. Additionally, the performances of kNN_{Best} and kNN_{CV} are given.

B.5 Visualizations of representation combinations

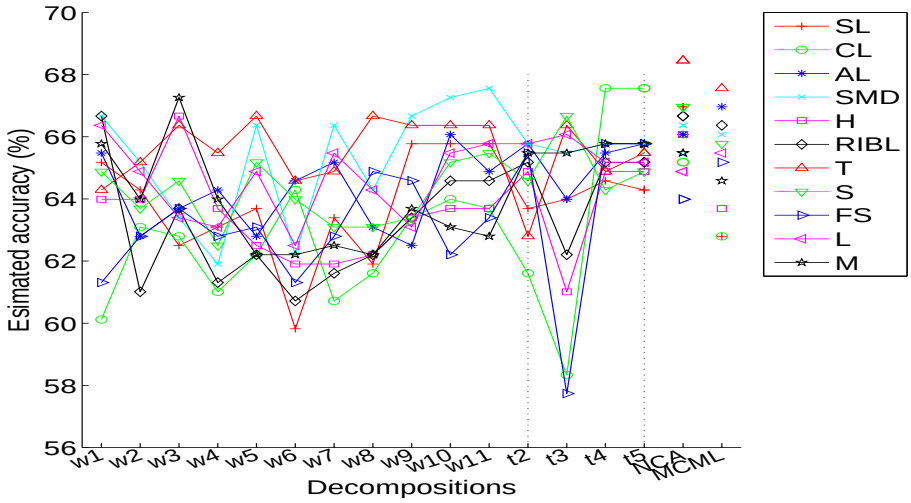


(a) FM

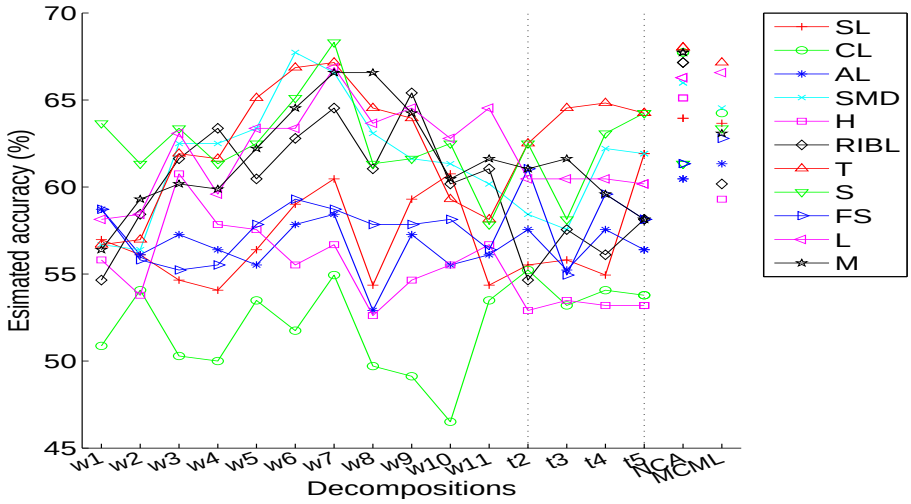


(b) FR

Figure B.17: Estimated accuracy of different kernels in the proximity space ($k_{P,d_{set}}$) vs. different decompositions for different set distance measures in the FM and FR datasets (wl denotes walks of length l whereas th denotes trees of height h). The last values in the plot denote the performance of $k_{Pd_{tuple,\boldsymbol{A}}}$ (and $k_{Pd_{tuple,\boldsymbol{W}}}$) where the different decompositions are combined.



(a) MM



(b) MR

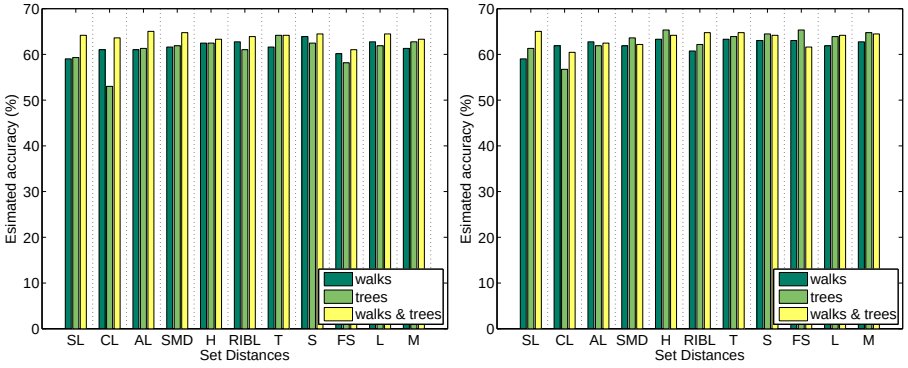
Figure B.18: Estimated accuracy of different kernels in the proximity space ($k_{P,d_{set}}$) vs. different decompositions for different set distance measures in the MM and MR datasets (wl denotes walks of length l whereas th denotes trees of height h). The last values in the plot denote the performance of $k_{Pd_{tuple,\boldsymbol{A}}}$ (and $k_{Pd_{tuple,\boldsymbol{W}}}$) where the different decompositions are combined.

	MCML			NCA		
	+	=	-	+	=	-
Mutagenesis	120	44	1	68	92	5
FM	17	148	0	7	158	0
FR	6	146	22	5	147	22
MM	12	151	2	12	153	0
MR	55	110	0	81	83	1

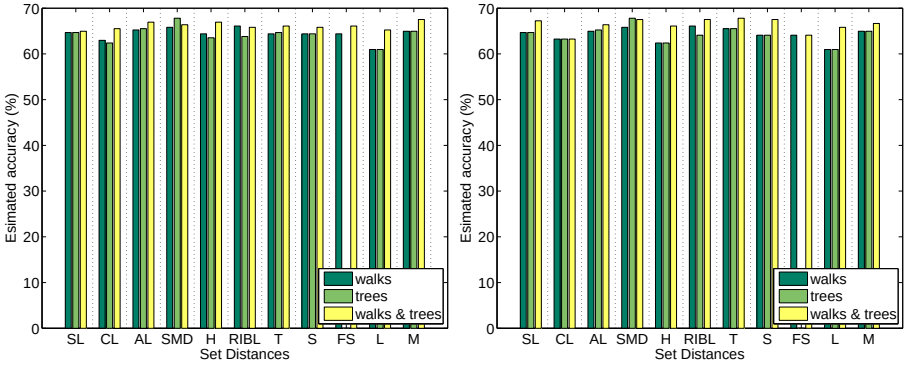
Table B.23: Significance test results comparing single decompositions vs. combination of decompositions. The values in the + and - columns denote the number of cases when MCML (NCA) are significantly better and worse than the single decompositions, respectively; the values in the = column correspond to the numbers of cases when statistically significant differences were not observed. The total number of comparisons for MCML (or NCA) is 165 (i.e. 11 walk decompositions x 11 set distances + 4 tree decompositions x 11 set distances).

	MCML			NCA		
	+	=	-	+	=	-
Mutagenesis	2	9	0	4	7	0
FM	1	10	0	1	10	0
FR	1	10	0	3	8	0
MM	3	8	0	3	8	0
MR	3	8	0	5	6	0

Table B.24: Significance test results comparing *heterogeneous* decompositions (i.e. into both walks and tress) vs. *homogeneous* decompositions (i.e. either into walks or trees). The values in the + and - columns denote the number of cases when the heterogeneous decompositions are significantly better and worse than the both homogeneous decompositions, respectively; the values in the = column correspond to the numbers of cases when the performances of the heterogeneous and both homogenious decompositions are statistically equivalent. The total number of comparisons for MCML (or NCA) is 11 (i.e. total number of set distances).

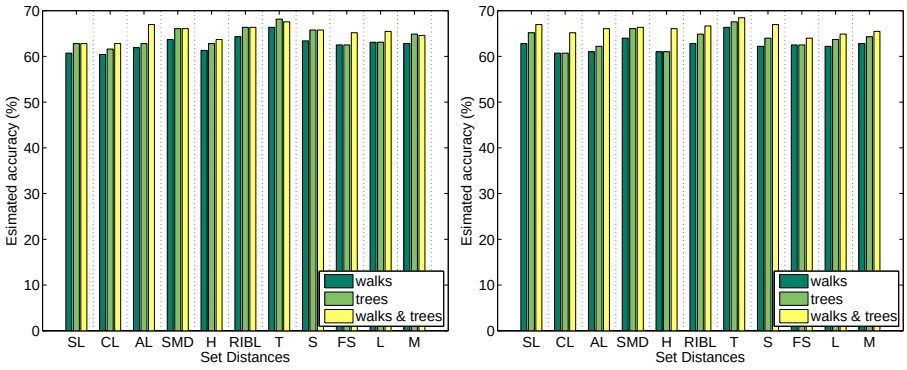


(a) FM, $\text{MCML}_{\text{full}}$ (b) FM, NCA_{full}



(c) FR, $\text{MCML}_{\text{full}}$ (d) FR, NCA_{full}

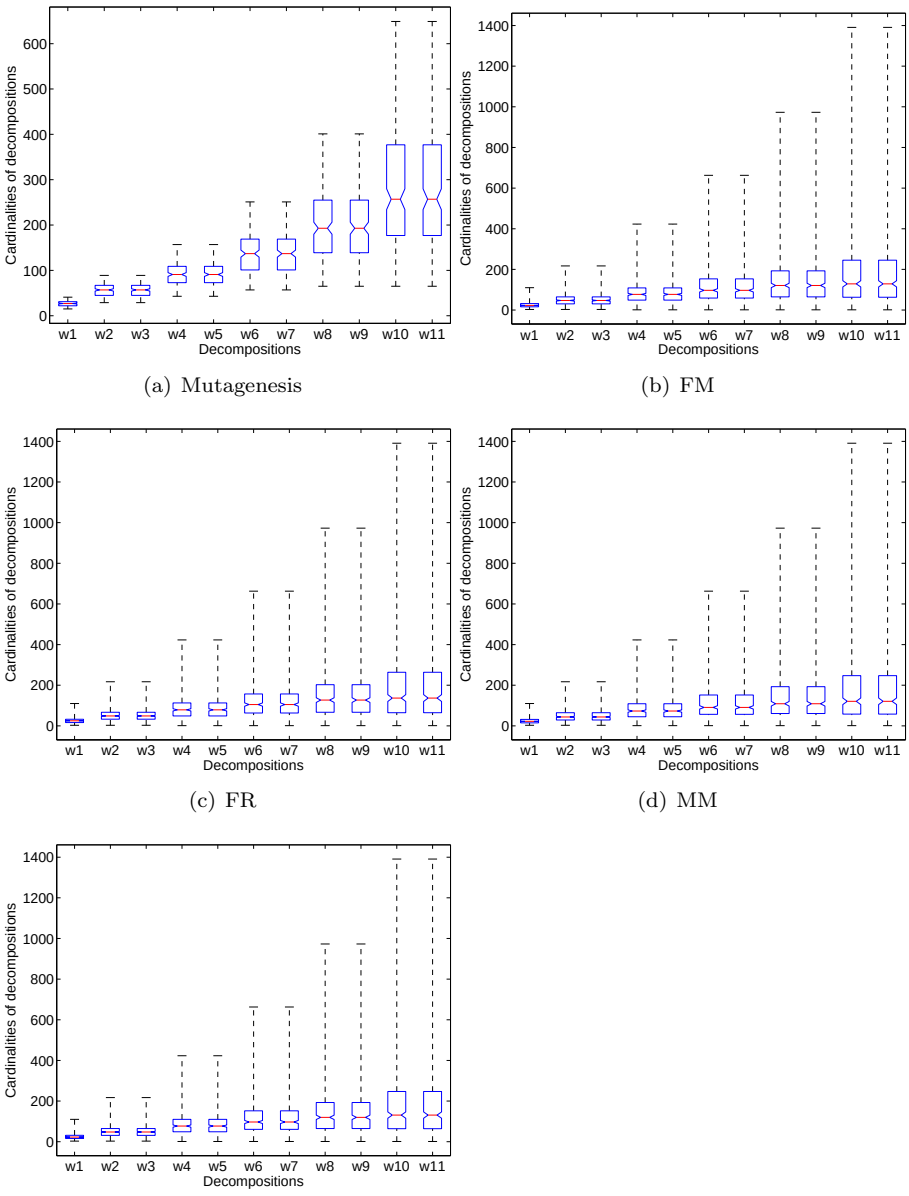
Figure B.19: Estimated accuracy of different kernels in the proximity space, $k_{Pd_{tuple,\boldsymbol{A}}}$ (for $\text{MCML}_{\text{full}}$) and $k_{Pd_{tuple,\boldsymbol{W}}}$ (for NCA_{full}), combining only walks, only trees and both walks and trees vs. different set distance measures in the FM and FR datasets.



(a) MM, $\text{MCML}_{\text{full}}$ (b) MM, NCA_{full}

(c) MR, $\text{MCML}_{\text{full}}$ (d) MR, NCA_{full}

Figure B.20: Estimated accuracy of different kernels in the proximity space, $k_{Pd_{tuple,\boldsymbol{A}}}$ (for $\text{MCML}_{\text{full}}$) and $k_{Pd_{tuple,\boldsymbol{W}}}$ (for NCA_{full}), combining only walks, only trees and both walks and trees vs. different set distance measures in the MM and MR datasets.



(a) Mutagenesis

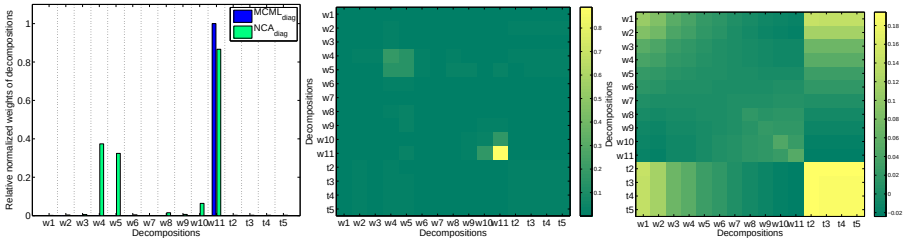
(b) FM

(c) FR

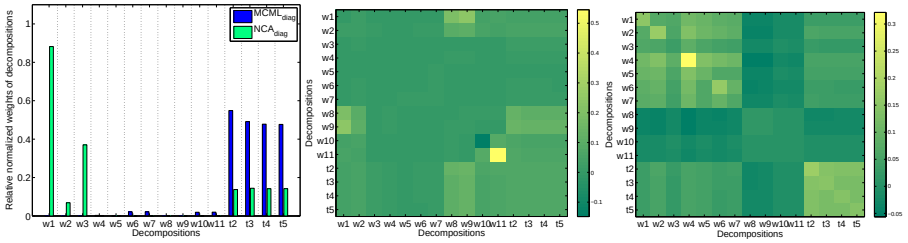
(d) MM

(e) MR

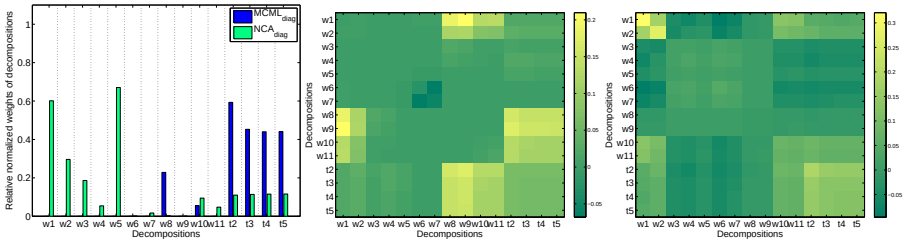
Figure B.21: Various statistics represented in the form of boxplots on the cardinalities for decompositions into walks for the graph datasets (wl denotes decompositions into walks of length l). The boxes have lines at the lower quartile, median, and upper quartile values. The whiskers extend from each end of the box to the most extreme values in the data.



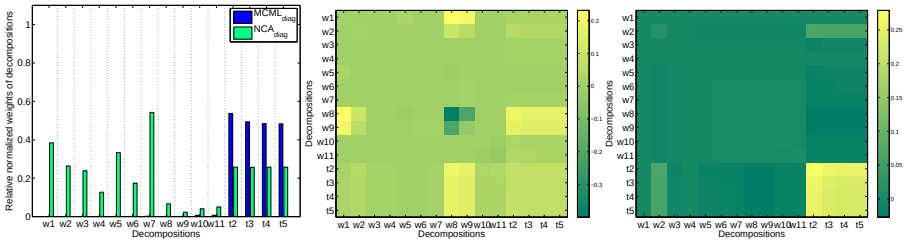
(a) muta, NCA_{diag}, $\text{MCML}_{\text{diag}}$ (b) muta, NCA_{full}, $\lambda = 0$ (c) muta, $\text{MCML}_{\text{full}}$, $\lambda = 0$



(d) FR, NCA_{diag}, $\text{MCML}_{\text{diag}}$ (e) FR, NCA_{full}, $\lambda = 0$ (f) FR, $\text{MCML}_{\text{full}}$, $\lambda = 0$

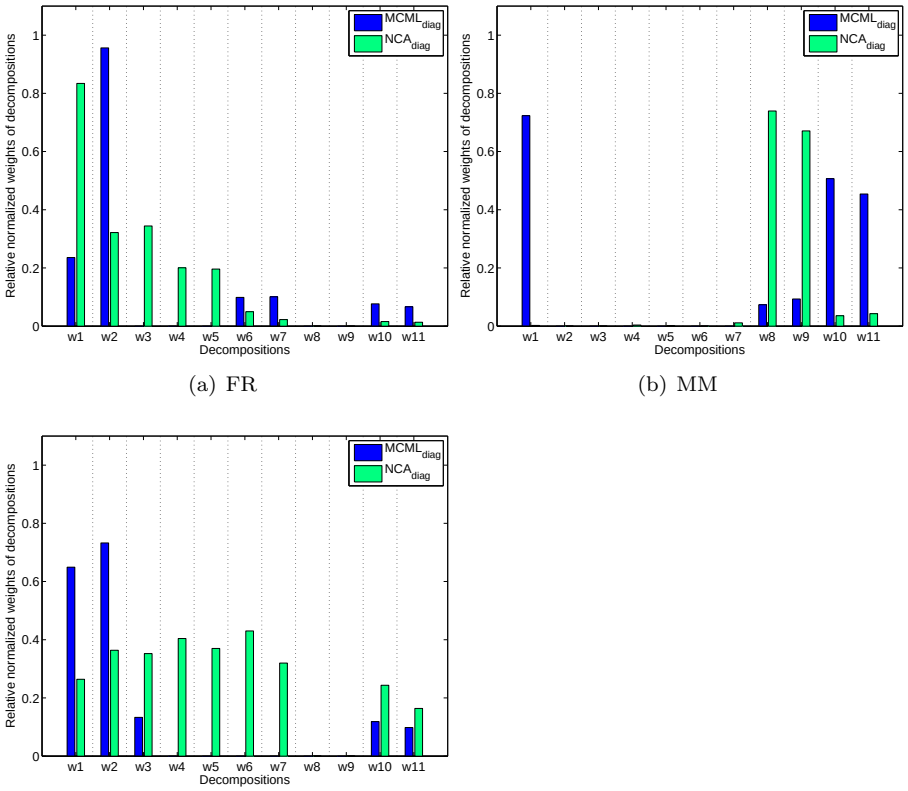


(g) MM, NCA_{diag}, $\text{MCML}_{\text{diag}}$ (h) MM, NCA_{full}, $\lambda = 0$ (i) MM, $\text{MCML}_{\text{full}}$, $\lambda = 0$



(j) MR, NCA_{diag}, $\text{MCML}_{\text{diag}}$ (k) MR, NCA_{full}, $\lambda = 0$ (l) MR, $\text{MCML}_{\text{full}}$, $\lambda = 0$

Figure B.22: Relative importance of different decompositions (mutagenesis, FR, MM and MR dataset) for the d_{SMD} set distance measure both for diagonal and full matrices $\boldsymbol{A}$ (for MCML) or $\boldsymbol{W}^T\boldsymbol{W} = \boldsymbol{A}$ (for NCA). wl denotes walks of length l whereas th denotes trees of height h. It is represented as normalized weights $\boldsymbol{A}$. Weights are normalized by a Frobenius norm of $\boldsymbol{A}$.

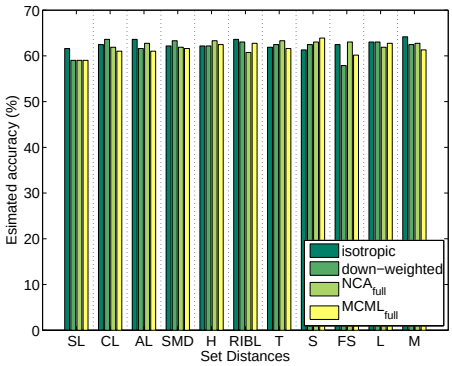


(a) FR

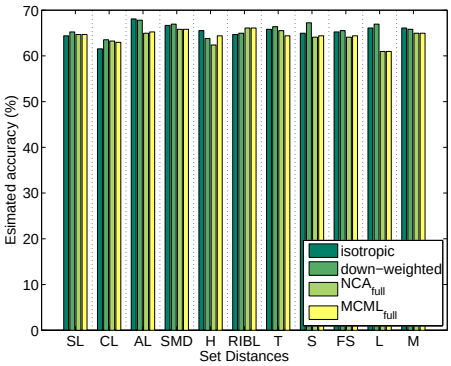
(b) MM

(c) MR

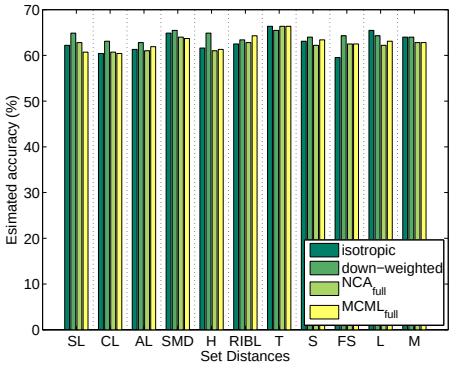
Figure B.23: Relative importance of different decompositions (FR, MM and MR datasets) for the d_{AL} set distance (wl denotes walks of length l).



(a) FM

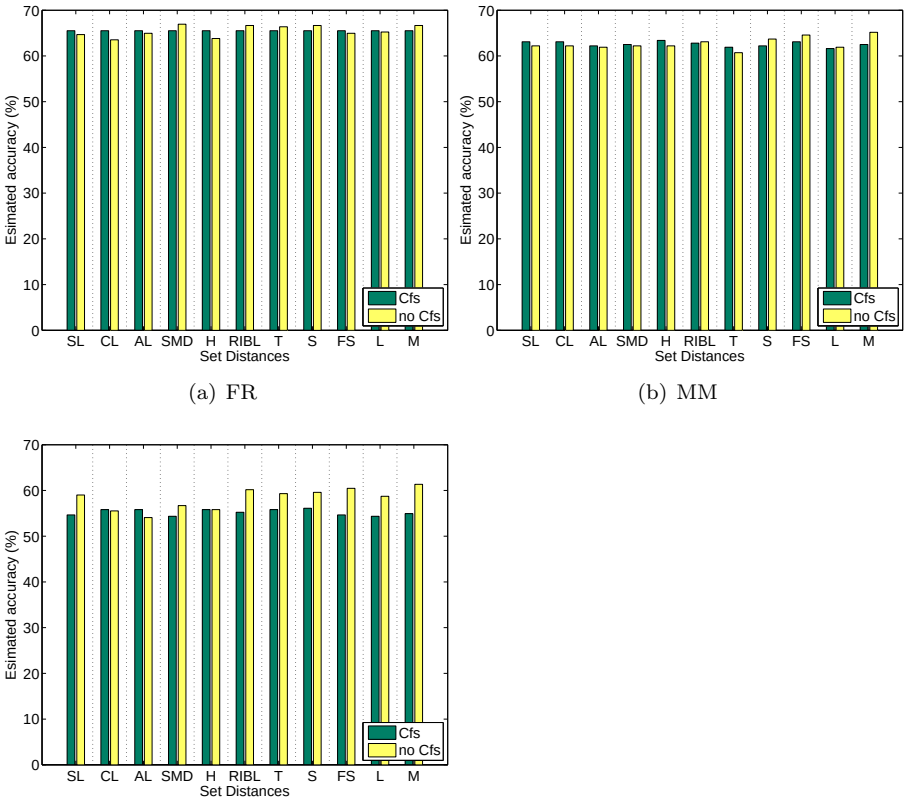


(b) FR



(c) MM

Figure B.24: Performances (using SVM) of the various weighting schemes for FM, FR and MM datasets.

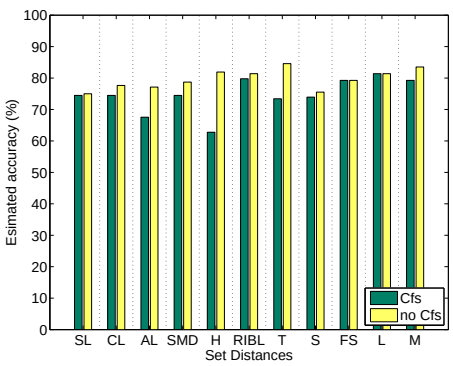


(a) FR

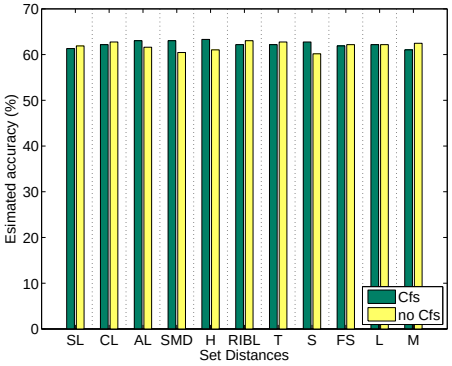
(b) MM

(c) MR

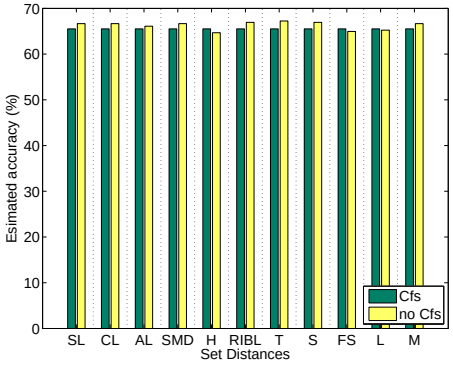
Figure B.25: Accuracy of $k_{Pd_{tuple,\boldsymbol{A}}}$ (using SVM) in the FR, MM, MR datasets and for different set distance measures. The results are obtained using $\text{MCML}_{\text{diag}}$ where feature selection in the proximity space is performed using the CFS method (the first bar). For comparison we also report the accuracy of the $k_{Pd_{tuple,\boldsymbol{A}}}$ on the full set of prototypes (the second bar).



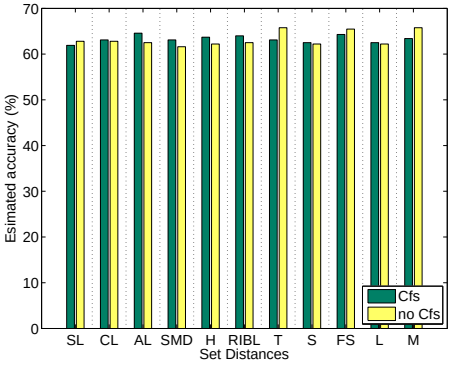
(a) mutagenesis



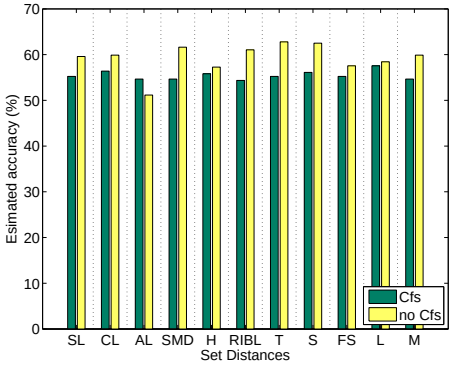
(b) FM



(c) FR

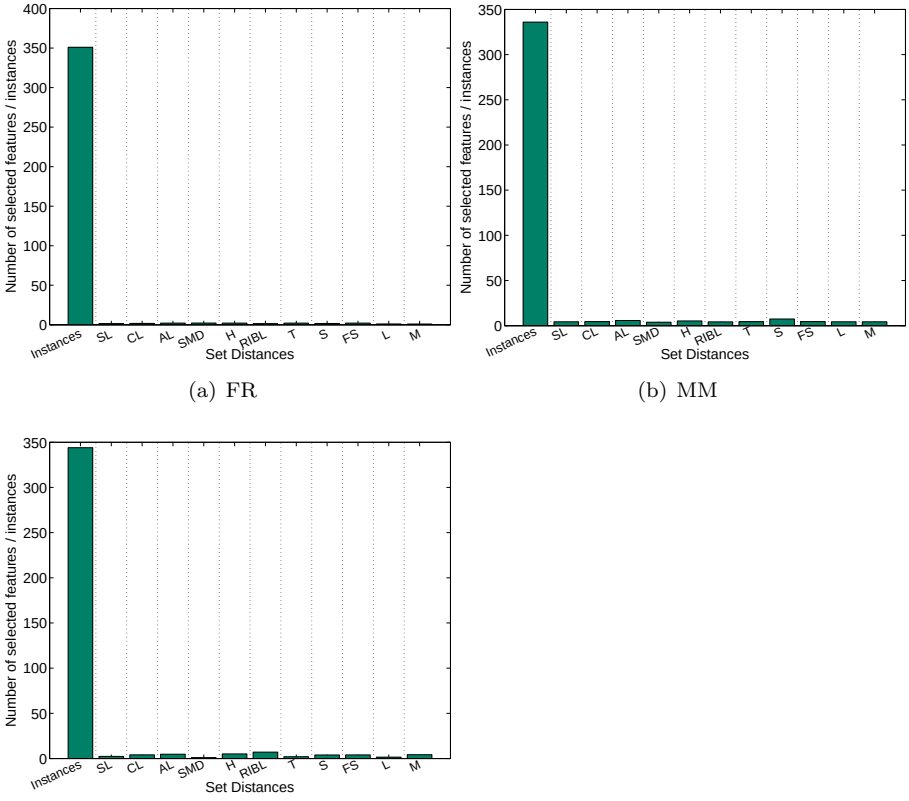


(d) MM



(e) MR

Figure B.26: Accuracy of $k_{Pd_{tuple,\boldsymbol{A}}}$ (using SVM) for different set distance measures. The results are obtained using $\text{MCML}_{\text{diag}}$ where feature selection in the proximity space is performed using the CFS method (the first bar). For comparison we also report the accuracy of the $k_{Pd_{tuple,\boldsymbol{A}}}$ on the full set of prototypes (the second bar).

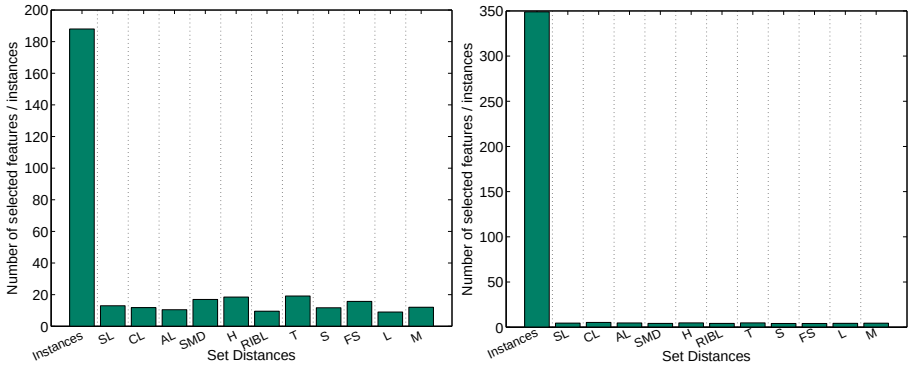


(a) FR

(b) MM

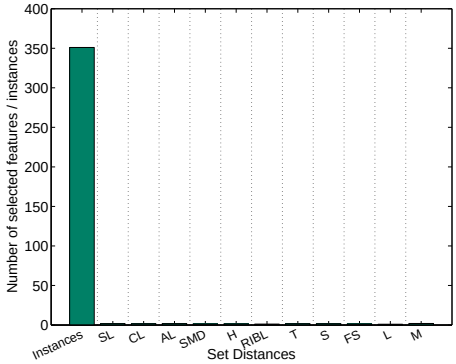
(c) MR

Figure B.27: Number of selected features (prototypes) by the CFS algorithm (using $\text{MCML}_{\text{diag}}$) for the FR, MM and MR datasets and for different set distance measures. The first bar denotes the number of instances in the training set.

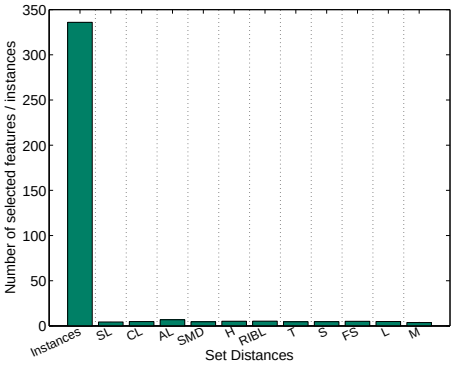


(a) mutagenesis

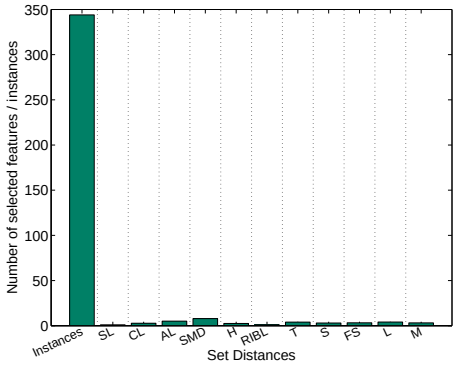
(b) FM



(c) FR



(d) MM



(e) MR

Figure B.28: Number of selected features (prototypes) by the CFS algorithm (using $\mathrm{NCA}_{\mathrm{diag}}$) for different set distance measures. The first bar denotes the number of instances in the training set.

	MCML						NCA					
	isotropic			down-weighted			isotropic			down-weighted		
	+	=	-	+	=	-	+	=	-	+	=	-
Mutagenesis	4	6	1	1	10	0	5	5	1	2	0	9
FM	1	10	0	0	11	0	0	11	0	0	10	1
FR	2	9	0	3	8	0	3	8	0	2	9	0
MM	0	11	0	0	11	0	0	11	0	0	11	0
MR	0	3	8	0	6	5	0	11	0	0	11	0

Table B.25: Significance test results comparing simple weighting schemes (i.e. *isotropic* and *down-weighted*) vs. the adaptive ones (i.e. MCML and NCA). The values in the + and - columns denote the number of cases when simple weighting schemes were significantly better and worse than MCML (or NCA), respectively; the values in the = column correspond to the numbers of cases when the performances of the simple weighting schemes and the adaptive ones were statistically equivalent. The number of comparisons is 11 (i.e. total number of set distances).

Appendix C

List of Notation

Sets:

$\mathbb{R}$	the real numbers
$\mathbb{R}^+$	the non-negative real numbers
$\mathbb{R}_0^+$	the positive real numbers
$\mathbb{N}$	the natural numbers (without zero)
$\mathbb{R}^{n\times n}$	$n \times n$ real matrices
$\mathcal{X}, X, A, \ldots$	general sets
$\lvert X\rvert$	cardinality of set X

Vectors and Matrices:

$\boldsymbol{x}$	boldface is used (column) vectors
$\boldsymbol{A}$	boldface is used to matrices
x_i	i-th element of vector $\boldsymbol{x}$
A_{ij}	element of $\boldsymbol{A}$ which is in i-th row and j-th column
$\boldsymbol{x}^T, \boldsymbol{A}^T$	transpose of $\boldsymbol{x}^T$ and $\boldsymbol{A}^T$
$\boldsymbol{I}$	square identity matrix (1's are on the diagonal and off-diagonal elements are 0's)
$diag(a_1, \ldots, a_n)$	matrix whose diagonal elements are $a_1, a_2, \ldots, a_n$ (off-diagonal elements are 0's)
$\boldsymbol{K}$	kernel (Gram) matrix
$\lVert\boldsymbol{x}\rVert$	Euclidean norm of vector $\boldsymbol{x}$
$\lVert\boldsymbol{A}\rVert_{\mathcal{F}}$	Frobenius norm of matrix $\boldsymbol{A}$
$\langle\cdot,\cdot\rangle$	inner (scalar) product
$\mathbf{1}$	vector of length consisting of the all 1s
$\boldsymbol{K}$	Gram (kernel) matrix

Objects:

ℓ	a list (Page 17)
$\ell[k]$	k element of ℓ (Page 17)
$\lvert\ell\rvert$	length of ℓ (Page 17)

$\boldsymbol{i}$	sequence of indices (Page 17)
$\mathcal{A}$	alignment (Page 60)
$c(\mathcal{A})$	cost of alignment $\mathcal{A}$ (Page 60)
$-$	a gap element (Page 60)
G	a graph (Page 28)
$\mathcal{V}$	set of vertices of a graph (Page 28)
$\mathcal{E}$	set of edges of a graph (Page 28)
$\mathcal{L}_\mathcal{V}$	set of vertex labels (Page 29)
$\mathcal{L}_\mathcal{E}$	set of edge labels (Page 29)
$lab(\cdot)$	label of a vertex or and edge (Page 29)
W	a walk (in a graph) (Page 29)
T	a tree (Page 29)
$root(T)$	root of tree (Page 30)
$\delta(\cdot)$	neighborhood of a node or an edge (Page 30)

Distances:

d_{num}	distance on numbers (Page 49)
d_δ	distance on sybolic objects (Page 49)
$d_{tuple,p}$	Minkowski distance measure on tuples (Page 61)
$d_{tuple,\boldsymbol{A}}$	Mahalanobis distance measure on tuples (Page 50)
$d_{tuple,\boldsymbol{W}}$	Mahalanobis distance measure on tuples (Page 51)
d_{set}	general distance measure on sets (Page 51)
d_{SL}	Single Linkage distance measure on sets (Page 51)
d_{CL}	Complete Linkage set distance measure (Page 52)
d_{AL}	Average Linkage set distance measure (Page 52)
d_{SMD}	Sum of Minimum Distances set distance measure (Page 52)
d_H	Hausdorff set distance measure (Page 53)
d_{RIBL}	RIBL set distance measure (Page 53)
d_S	Surjections set distance measure (Page 55)
d_{FS}	Fair Surjections set distance measure (Page 55)
d_L	Linkings set distance measure (Page 55)
d_M	Matchings set distance measure (Page 55)
d_T	Tanimoto set distance measure (Page 57)
d_{edit}	(normalized) edit distance measure on lists (Page 62)
d_{tree}	distance measure on trees (Page 63)

Kernels:

k_{norm}	kernel normalized in feature space (Page 94)
k_{num}	kernel on numbers (Page 95)
k_δ	kernel on sybolic objects (Page 95)
k_Σ	direct sum kernel (Page 96)
k_Π	tensor product kernel (Page 96)
$k_\Re$	$\Re$-Convolution kernel (Page 97)

k_{lin}	linear kernel (Page 98)
k_{poly}	polynomial kernel (Page 98)
k_{RBF}	Gaussian RBF kernel (Page 99)
k_{CP}	Cross Product kernel (Page 99)
k_{SL}	Single Linkage set kernel (Page 102)
k_{CL}	Complete Linkage set kernel (Page 103)
k_{SMD}	Sum of Maximum Kernels set kernel (Page 103)
k_H	Hausdorff set kernel (Page 103)
k_{RIBL}	RIBL set kernel (Page 104)
k_S	Surjections set kernel (Page 104)
k_{FS}	Fair Surjections set kernel (Page 104)
k_L	Linkings set kernel (Page 104)
k_M	Matchings set kernel (Page 105)
$k_\cap$	Intersection set kernel (Page 105)
k_{DS}	Distance Substitution kernel (Page 105)
k_{CS}	Contiguous Sublist kernel (Page 108)
k_{LCS}	Longest Common Sublist kernel (Page 109)
k_{tree}	tree kernel (Page 110)

Various:

iff	if and only if
f_{norm}	normalization function for Cross Product kernel (Page 100)
$r(\boldsymbol{K})$	ratio of negative to positive eigenvalues of $\boldsymbol{K}$ (Page 113)
$\mathrm{SVM_P}$	SVM with proximity kernels (Page 121)
$\mathrm{SVM_{DS}}$	SVM with Distance Substitution kernels (Page 121)
$\mathrm{SVM_{SP}}$	SVM with kernels directly based on specific pairs (Page 121)