Antti Airola

Kernel-Based Ranking

Methods for Learning and Performance Estimation

Turku Centre *for* Computer Science

TUCS Dissertations
No 144, December 2011

Kernel-Based Ranking

Methods for Learning and Performance Estimation

Antti Airola

To be presented, with the permission of the Faculty of Mathematics and Natural Sciences of the University of Turku, for public criticism in Auditorium Beta on December 12th, 2011, at 12 noon.

University of Turku
Department of Information Technology
Joukahaisenkatu 3-5, 20520 Turku

2011

Supervisors

Professor Tapio Salakoski
Department of Information Technology
University of Turku
Finland

Adjunct Professor Tapio Pahikkala
Department of Information Technology
University of Turku
Finland

Reviewers

Professor Eyke Hüllermeier
Department of Mathematics and Computer Science
University of Marburg
Germany

Adjunct Professor Juho Rousu
Department of Computer Science
University of Helsinki
Finland

Opponent

Adjunct Professor Timo Honkela
Department of Information and Computer Science
Aalto University
Finland

ISBN 978-952-12-2674-8
ISSN 1239-1883

Abstract

Machine learning provides tools for automated construction of predictive models in data intensive areas of engineering and science. The family of regularized kernel methods have in the recent years become one of the mainstream approaches to machine learning, due to a number of advantages the methods share. The approach provides theoretically well-founded solutions to the problems of under- and overfitting, allows learning from structured data, and has been empirically demonstrated to yield high predictive performance on a wide range of application domains. Historically, the problems of classification and regression have gained the majority of attention in the field. In this thesis we focus on another type of learning problem, that of learning to rank.

In learning to rank, the aim is from a set of past observations to learn a ranking function that can order new objects according to how well they match some underlying criterion of goodness. As an important special case of the setting, we can recover the bipartite ranking problem, corresponding to maximizing the area under the ROC curve (AUC) in binary classification. Ranking applications appear in a large variety of settings, examples encountered in this thesis include document retrieval in web search, recommender systems, information extraction and automated parsing of natural language. We consider the pairwise approach to learning to rank, where ranking models are learned by minimizing the expected probability of ranking any two randomly drawn test examples incorrectly. The development of computationally efficient kernel methods, based on this approach, has in the past proven to be challenging. Moreover, it is not clear what techniques for estimating the predictive performance of learned models are the most reliable in the ranking setting, and how the techniques can be implemented efficiently.

The contributions of this thesis are as follows. First, we develop RankRLS, a computationally efficient kernel method for learning to rank, that is based on minimizing a regularized pairwise least-squares loss. In addition to training methods, we introduce a variety of algorithms for tasks such as model selection, multi-output learning, and cross-validation, based on computational shortcuts from matrix algebra. Second, we improve the

fastest known training method for the linear version of the RankSVM algorithm, which is one of the most well established methods for learning to rank. Third, we study the combination of the empirical kernel map and reduced set approximation, which allows the large-scale training of kernel machines using linear solvers, and propose computationally efficient solutions to cross-validation when using the approach. Next, we explore the problem of reliable cross-validation when using AUC as a performance criterion, through an extensive simulation study. We demonstrate that the proposed leave-pair-out cross-validation approach leads to more reliable performance estimation than commonly used alternative approaches. Finally, we present a case study on applying machine learning to information extraction from biomedical literature, which combines several of the approaches considered in the thesis. The thesis is divided into two parts. Part I provides the background for the research work and summarizes the most central results, Part II consists of the five original research articles that are the main contribution of this thesis.

Acknowledgements

First, I would like to thank my supervisor Tapio Salakoski, who has over the years provided me the support necessary for pursuing my thesis work, always given good advice when needed, and allowed me the freedom to find my own path. The research group he has founded is something to be proud of, and it has been a privilege to work as part of it. I owe a great debt also to my supervisor Tapio Pahikkala, who has had a large influence in shaping me as a researcher. Without our close collaboration on machine learning research this thesis would have looked quite different, I would like to thank him for all he has taught me and for sharing a vision. Special thanks go also to Filip Ginter and Sampo Pyysalo, who helped me in settling in the group and getting started in my research, and taught me much of what (little) I know about BioNLP.

During my thesis work I have had the pleasure of collaborating with many excellent researchers in our group. I would like to thank especially Jorma Boberg, Jouni Järvinen, Jari Björne, Juho Heimonen, Pekka Naula, Sebastian Okser, Hanna Suominen and Evgeni Tsivtsivadze. Jorma and Jouni as senior researchers offered me guidance early in my research work. With Jari and Juho we have had a fruitful collaboration in the area of biomedical text mining, and many interesting discussions over the years. It has been great to pass the torch on to Pekka and Sebastian, and see all the things they have done, partly based on our previous research. Last but certainly not least both Hanna and Evgeni deserve many thanks for our close research collaboration in machine learning, on the infamous project Adarctic, and on their great company whether we were wading through the snow to math classes, just hanging around in the TUCS coffee room, or traveling abroad together.

I would like to thank Sanna Salanterä, Heljä Lundgrén-Laine, Riitta Danielsson-Ojala and the others for our collaboration on Louhi and IKITIK. Thanks go to Tero Aittokallio for introducing me to the world of genome-wide association studies. Also, international collaborations have shaped my research during the thesis work, and helped to widen my perspective on how research is done in different places around the world. Thanks for this go to Willem Waegeman and Bernard De Baets from the University of Ghent, and

Fabian Gieseke and Oliver Kramer from the University of Oldenburg, who have all directly contributed to my research.

Further, my thanks go to the Turku Centre for Computer Science (TUCS) and the Department of Information Technology in University of Turku, and the entire staff working there. My work has been possible only due to the support offered by these institutions. Also, I would like to thank Nokia Foundation for the financial support they have provided.

I am grateful to the reviewers of this thesis, Professor Eyke Hüllermeier and Adjunct Professor Juho Rousu, both for their excellent criticisms and encouragement. Further, I sincerely thank Adjunct Professor Timo Honkela for agreeing to act as my opponent.

Finally, I am deeply grateful to my parents for their unwavering support, love and encouragement over the years.

List of original publications

I Pahikkala, T., Tsivtsivadze, E., Airola, A., Järvinen, J., and Boberg, J. (2009). An efficient algorithm for learning to rank from preference graphs. *Machine Learning*, 75(1):129–165.

II Airola, A., Pahikkala, T., and Salakoski, T. (2011). Training linear ranking SVMs in linearithmic time using red-black trees. *Pattern Recognition Letters*, 32(9):1328–1336.

III Airola, A., Pahikkala, T., and Salakoski, T. (2011). On learning and cross-validation with decomposed Nyström approximation of kernel matrix. *Neural Processing Letters*, 33(1):17–30.

IV Airola, A., Pahikkala, T., Waegeman, W., De Baets, B., and Salakoski, T. (2011). An experimental comparison of cross-validation techniques for estimating the area under the ROC curve. *Computational Statistics & Data Analysis*, 55(4):1828–1844.

V Airola, A., Pyysalo, S., Björne, J., Pahikkala, T., Ginter, F., and Salakoski, T. (2008). All-paths graph kernel for protein-protein interaction extraction with evaluation of cross-corpus learning. *BMC Bioinformatics*, 9 Suppl 11.

List of related publications not included in the thesis

Co-authored scientific publications

- Airola, A., Pahikkala, T., Boberg, J., and Salakoski, T. (2010). Applying permutation tests for assessing the statistical significance of wrapper based feature selection. In Draghici, S., Khoshgoftaar, T. M., Palade, V., Pedrycz, W., Wani, M. A., and Zhu, X., editors, *Proceedings of The Ninth International Conference on Machine Learning and Applications (ICMLA 2010)*, pages 989–994. IEEE.

- Airola, A., Pahikkala, T., and Salakoski, T. (2011). An improved training algorithm for the linear ranking support vector machine. In Honkela, T., Duch, W., Girolami, M., and Kaski, S., editors, *Proceedings of the 21st International Conference on Artificial Neural Networks (ICANN 2011)*, volume 6791 of *Lecture Notes in Computer Science*, pages 134–141. Springer.

- Airola, A., Pahikkala, T., and Salakoski, T. (2010). Large scale training methods for linear RankRLS. In Hüllermeier, E. and Fürnkranz, J., editors, *Proceedings of the ECML/PKDD-Workshop on Preference Learning (PL-10)*.

- Airola, A., Pahikkala, T., Waegeman, W., De Baets, B., and Salakoski, T. (2010). A comparison of AUC estimators in small-sample studies. In Džeroski, S., Geurts, P., and Rousu, J., editors, *Proceedings of the third International Workshop on Machine Learning in Systems Biology*, volume 8 of *JMLR Workshop and Conference Proceedings*, pages 3–13. Journal of Machine Learning Research.

- Airola, A., Pyysalo, S., Björne, J., Pahikkala, T., Ginter, F., and Salakoski, T. (2008). A graph kernel for protein-protein interaction extraction. In *Proceedings of the Workshop on Current Trends in*

Biomedical Natural Language Processing (BioNLP'08), pages 1–9. Association for Computational Linguistics.

- Björne, J., Airola, A., Pahikkala, T., and Salakoski, T. (2011). Drug-drug interaction extraction from biomedical texts with SVM and RLS classifiers. In Segura-Bedmar, I., Martinez, P., and Sanchez-Cisneros, D., editors, *Proceedings of the SEPLN 2011 Workshop on First Challenge Task on Drug-Drug Interaction Extraction (DDIExtraction 2011)*, volume 761 of *CEUR Workshop Proceedings*, pages 35–42. CEUR-WS.org.

- Björne, J., Heimonen, J., Ginter, F., Airola, A., Pahikkala, T., and Salakoski, T. (2009). Extracting complex biological events with rich graph-based feature sets. In *Proceedings of the BioNLP'09 Shared Task on Event Extraction*, pages 10–18.

- Björne, J., Heimonen, J., Ginter, F., Airola, A., Pahikkala, T., and Salakoski, T. (2011). Extracting contextualized complex biological events with rich graph-based feature sets. *Computational Intelligence*. To appear.

- Gieseke, F., Kramer, O., Airola, A., and Pahikkala, T. (2011). Speedy local search for semi-supervised regularized least-squares. In Bach, J., Edelkamp, S., editors, *Proceedings of the 34th Annual German Conference on Artificial Intelligence (KI 2011)*, Lecture Notes in Computer Science, pages 87–98. Springer.

- Naula, P., Pahikkala, T., Airola, A., and Salakoski, T. (2011). Greedy regularized least-squares for multi-task learning. In Li, B., Zhu, X., and Yang, Q., editors, *Proceedings of the Fifth International ICDM Workshop on Mining Multiple Information Sources*. IEEE. To appear.

- Naula, P., Pahikkala, T., Airola, A., and Salakoski, T. (2011). Learning multi-label predictors under sparsity budget. In Kofod-Petersen, A., Heintz, F., and Langseth, H., editors, *Eleventh Scandinavian Conference on Artificial Intelligence (SCAI 2011)*, volume 227 of *Frontiers in Artificial Intelligence and Applications*, pages 30–39. IOS Press.

- Okser, S., Pahikkala, T., Airola, A., Aittokallio, T., and Salakoski, T. (2011). Fast and parallelized greedy forward selection of genetic variants in genome-wide association studies. In *IEEE International Workshop on Genomic Signal Processing and Statistics (GENSIPS 2011)*. IEEE. To appear.

- Pahikkala, T., Airola, A., Boberg, J., and Salakoski, T. (2008). Exact and efficient leave-pair-out cross-validation for ranking RLS. In

Honkela, T., Pöllä, M., Paukkeri, M.-S., and Simula, O., editors, *Proceedings of the 2nd International and Interdisciplinary Conference on Adaptive Knowledge Representation and Reasoning (AKRR 2008)*, pages 1–8. Helsinki University of Technology.

- Pahikkala, T., Airola, A., Naula, P., and Salakoski, T. (2010). Greedy RankRLS: a linear time algorithm for learning sparse ranking models. In Gabrilovich, E., Smola, A. J., and Tishby, N., editors, *SIGIR 2010 Workshop on Feature Generation and Selection for Information Retrieval*, pages 11–18. ACM.

- Pahikkala, T., Airola, A., and Salakoski, T. (2010). Feature selection for regularized least-squares: New computational short-cuts and fast algorithmic implementations. In Kaski, S., Miller, D. J., Oja, E., and Honkela, A., editors, *Proceedings of the Twentieth IEEE International Workshop on Machine Learning for Signal Processing (MLSP 2010)*, pages 295–300. IEEE.

- Pahikkala, T., Airola, A., and Salakoski, T. (2010). Speeding up greedy forward selection for regularized least-squares. In Draghici, S., Khoshgoftaar, T. M., Palade, V., Pedrycz, W., Wani, M. A., and Zhu, X., editors, *Proceedings of The Ninth International Conference on Machine Learning and Applications (ICMLA 2010)*, pages 325–330. IEEE.

- Pahikkala, T., Airola, A., Suominen, H., Boberg, J., and Salakoski, T. (2008). Efficient AUC maximization with regularized least-squares. In Holst, A., Kreuger, P., and Funk, P., editors, *Tenth Scandinavian Conference on Artificial Intelligence, SCAI 2008*, volume 173 of *Frontiers in Artificial Intelligence and Applications*, pages 12–19. IOS Press, Amsterdam, Netherlands.

- Pahikkala, T., Airola, A., Xu, T. C., Liljeberg, P., Tenhunen, H., and Salakoski, T. (2011). A parallel online regularized least-squares machine learning algorithm for future multi-core processors. In *Proceedings of the 1st International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS 2011)*, pages 590–599. SciTePress.

- Pahikkala, T., Tsivtsivadze, E., Airola, A., Boberg, J., and Salakoski, T. (2007). Learning to rank with pairwise regularized least-squares. In Joachims, T., Li, H., Liu, T.-Y., and Zhai, C., editors, *SIGIR 2007 Workshop on Learning to Rank for Information Retrieval*, pages 27–33.

- Pahikkala, T., Waegeman, W., Airola, A., Salakoski, T., and De Baets, B. (2010). Conditional ranking on relational data. In Balcázar, J. L., Bonchi, F., Gionis, A., and Sebag, M., editors, *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML PKDD 2010, Barcelona, Spain, September 20-24, 2010, Proceedings, Part II*, volume 6322 of *Lecture Notes in Computer Science*, pages 499–514. Springer.

- Pyysalo, S., Airola, A., Heimonen, J., Björne, J., Ginter, F., and Salakoski, T. (2008). Comparative analysis of five protein-protein interaction corpora. *BMC Bioinformatics*, 9 Suppl 3.

- Suominen, H., Ginter, F., Pyysalo, S., Airola, A., Pahikkala, T., Salanterä, S., and Salakoski, T. (2008). Machine learning to automate the assignment of diagnosis codes to free-text radiology reports: a method description. In Hauskrecht, M., Schuurmans, D., and Szepesvari, C., editors, *Proceedings of the ICML/UAI/COLT Workshop on Machine Learning in Health Care Applications*.

- Tsivtsivadze, E., Pahikkala, T., Airola, A., Boberg, J., and Salakoski, T. (2008). A sparse regularized least-squares preference learning algorithm. In Holst, A., Kreuger, P., and Funk, P., editors, *Tenth Scandinavian Conference on Artificial Intelligence, SCAI 2008*, volume 173 of *Frontiers in Artificial Intelligence and Applications*, pages 76–83, Amsterdam, Netherlands. IOS Press.

- Waegeman, W., Pahikkala, T., Airola, A., Salakoski, T., and De Baets, B. (2012). Learning valued relations from data. In Melo-Pinto, P., Couto, P., Serodio, C., Fodor, J., and De Baets, B., editors, *Proceedings of the EUROFUSE 2011 Workshop on Fuzzy Methods for Knowledge-Based Systems*, volume 107 of *Advances in Intelligent and Soft Computing*, pages 257–268. Springer.

Co-edited conference proceedings

- Pahikkala, T., Väyrynen, J., Kortela, J., and Airola, A., editors (2010). *Proceedings of the 14th Finnish Artificial Intelligence Conference, STeP 2010*, number 25 in Publications of the Finnish Artificial Intelligence Society, Espoo, Finland. Aalto-Print.

Contents

Chapter 1

Introduction

1.1 Machine learning

Machine learning is the scientific discipline whose goal is the development of algorithms that allow computer programs to learn from experience. The field of machine learning is located at the intersection of the fields of computer science, statistics and mathematics. From computer science it incorporates the necessary tools for implementing the learning algorithms and data structures, and the analysis techniques for reasoning about the tractability and efficiency of computation. Traditional statistics offers the basis for considering what conclusions can be inferred from finite data samples, and with what confidence. Mathematics serves as the foundation for both computer science and statistics, and also offers many additional tools for machine learning. For more in depth discussion about the fundamental characteristics, history and the central research questions of the field, see e.g. Vapnik (1995); Mitchell (2006); Hastie et al. (2009); Langley (2011).

The applications of machine learning are varied. Increasingly, methods from the field are having impact both on the development of engineering applications and on other fields of research. This has been made both necessary and possible by the unprecented quantities of data now available in electronic format. Advances both in the theory of machine learning, and the amount of available computational resources, allow the leveraging of this data for automatic development of predictive models. This has the potential to produce substantial advances in data intensive fields.

For engineering applications, machine learning methods allow automated construction of complex programs. For many tasks it is very difficult or even impossible to write down a set of explicit rules that could be implemented as a computer program. Further, it may be the case that an application needs to adapt to its environment, meaning that one cannot in advance specify the exact desired behavior of the system. In such settings one may collect ex-

amples of desired behavior, and use them to automatically train a machine learning method to perform the task. Examples of applications, where this approach is having major impact include model predictive control in industrial systems (Nrgaard et al., 2000), natural language processing (Manning and Schütze, 1999), optical character recognition (LeCun et al., 1998), recommender systems (Koren et al., 2009), robotics (Argall et al., 2009), self-adapting search engines (Joachims and Radlinski, 2007), and speech recognition (Gong, 1995).

In data driven research, machine learning methods allow extraction of (scientific) knowledge. This process is often called *data mining*, or *knowledge discovery from data* (Fayyad et al., 1996). In fields such as bioinformatics the complexity and amount of data far exceeds the capability of human beings to process and make use of, thus automatic methods for detecting regularities are needed. Discovered models may be interesting as such as descriptors of some characteristics of the process generating the data, or be used to predict or simulate future behavior. Examples of prominent applications, where machine learning methods are used for data analysis include astronomical data analysis (Ball et al., 2006), biomedical text mining (Zweigenbaum et al., 2007), marketing analysis (Berry and Linoff, 2004) and identification of genetic variants that contribute to diseases (Okser et al., 2010). It has been suggested that a new scientific methodology driven by data-intensive problems is now emerging (Hey et al., 2009). Some interesting recent developments include automated inference of natural laws from experimental data (Schmidt and Lipson, 2009) and the prototype robot scientist Adam, which represents an attempt towards automating not only data analysis, but also hypothesis generation and experimentation (King et al., 2009).

Finally, a commonly made assumption is that machine learning is essential for development of artificial intelligence. Already Turing (1950) envisioned programs with human-like capabilities to general problem solving and proposed, that the ability to learn would be an essential characteristic for such a program. When measured against such expectations, the outcomes of the last 60 years of artificial intelligence research have proven to be quite disappointing. Still, the human brain continues to serve as an inspiration for machine learning (Smale et al., 2010), and new computational models for allowing the development of complex artificial intelligences are being explored (see e.g. Bengio and LeCun (2007); Legg and Hutter (2007)). The hope remains that with advances in fields such as machine learning, coupled with increasing computational power, the development of artificial intelligences capable of general problem solving will become possible.

1.2 Research overview

The research carried out as part of this thesis work was motivated by challenges encountered, when applying machine learning methods in a number of application projects. The main focus of these projects was the automated analysis of biomedical text (Ginter, 2007; Pahikkala, 2008; Pyysalo, 2008; Suominen, 2009; Tsivtsivadze, 2009). The excellent performance of the so-called *regularized kernel methods* in the text domain (see e.g. Joachims (2002a); Pahikkala (2008)), combined with a number of other advantages the methods share, lead us to specifically consider this family of learning methods. The problem of *ranking* was soon identified as one major challenge in the application projects. This was both due to its connection to *area under the ROC curve* (AUC) analysis (Agarwal et al., 2005), as well as due to more general ranking tasks encountered in areas such as medical decision making (Suominen et al., 2006), and automated parsing of text (Tsivtsivadze et al., 2005). From methodological point of view, we encountered two major challenges. One one hand, it was necessary to develop computationally efficient methods for solving the learning tasks, and at the other hand reliable and efficient methods for evaluating the generalization performance of the learned models were needed. Next, we sketch the background for these research problems, and then proceed to formalize the research objectives of this thesis.

1.2.1 Problem setting

The main focus of this thesis is on the development and analysis of general domain-independent methods for machine learning and performance evaluation. We limit our considerations to the *supervised learning* setting, where a program is trained to make predictions by supplying it with *training data* consisting of *inputs* and *labels*. Based on these it tries to produce a general hypothesis about the dependency between the inputs and the labels, such that would allow it to accurately predict the labels of future observations for which only the inputs are known. Historically, the two most commonly studied supervised learning settings are *classification*, where the labels encode class memberships, and *regression*, where the labels are real valued. In this thesis, we will study in detail a third type of supervised learning task, that of *learning to rank*.

In learning to rank, the aim is to learn from the training data a ranking function, that is able to order sets of objects according to how well they match some underlying ranking criterion. The information about this match is often encoded via *utility scores*, where a higher utility score indicates a higher rank than a lower one. Alternatively, *pairwise preferences* may be used directly to encode information about relative order between objects.

Given a set of training examples consisting of inputs, and either utility scores or pairwise comparisons, one aims to learn a ranking function that is used to predict rankings for new data. A common approach to learning ranking functions is to minimize the probability of a mistake when predicting, which of two new randomly drawn examples should be ranked higher (Cohen et al., 1998). The bipartite ranking problem where only two possible utility values exist, which corresponds to maximizing AUC performance measure in binary classification (Bamber, 1975; Cortes and Mohri, 2004; Agarwal et al., 2005), provides an important special case of this general ranking setting.

The family of regularized kernel methods embodies one of the mainstream approaches to machine learning (Schölkopf and Smola, 2002; Shawe-Taylor and Cristianini, 2004). In supervised learning, these methods typically allow the use of structured data and non-linear modeling, and offer principled ways to deal with both the underfitting and overfitting phenomena, while still leading to convex optimization problems, where globally optimal solutions can be found. Widely used kernel methods include the *support vector machine* (SVM) (Vapnik, 1995, 1998; Drucker et al., 1997) and the *regularized least-squares* (RLS) (Poggio and Girosi, 1990; Poggio and Smale, 2003) algorithms for both classification and regression. Previously, the scalability of learning has remained an issue, when adapting kernel methods to ranking. Straightforward adaptations of existing learning methods, such as the standard ranking support vector machine (RankSVM) training method described in Herbrich et al. (1999); Joachims (2002b) lead to solving optimization problems whose size may depend quadratically rather than linearly on the size of the training set. For linear RankSVM more efficient training methods are known (Joachims, 2006), but these are limited to settings where the number of possible ranks in the data can be assumed a small constant. Thus more efficient training algorithms are needed to allow better applicability of kernel based methods for learning to rank

Performance evaluation is a central task in almost any area of machine learning. Here, the goal is to estimate the expected predictive accuracy of learned models on future data unseen during the training phase, as measured by some performance measure. This may be required for example in order to decide whether a developed system can be taken into use, for comparing learning algorithms or for model selection (Dietterich, 1998). A technique known as *cross-validation* is commonly applied to estimate learner performance. However, in many settings achieving reliable results with cross-validation can be challenging. The assumption that all the training samples are sampled independently of each other is routinely broken in many applications, leading to biased estimates (see e.g. Pahikkala et al. (2006a); Sætre et al. (2007)). The use of multivariate performance measures, such as the AUC, can lead to problems when predictions made on different rounds of cross-validation are combined together (Parker et al., 2007; Forman and

Scholz, 2010). Finally, the use of reduced set approximations (Smola and Schölkopf, 2000; Quiñonero-Candela and Rasmussen, 2005) for scaling kernel methods to large datasets can result in unexpected complications when doing cross-validation (Pahikkala et al., 2009). The computational costs of cross-validation are also of concern, as straightforward implementations of the procedure require re-training a learning algorithm a large number of times.

1.2.2 Research objectives

The main objectives of this thesis are to develop computationally efficient methods for learning accurate ranking models, and reliable and efficient strategies for performance estimation.

Efficient learning methods. We aim to develop novel algorithms for learning ranking functions. By minimizing an approximation of the pairwise ranking error, the goal is to learn predictors with ranking accuracy that is competitive with state-of-the art in ranking methods. We prefer general methods that are able to learn both from utility scores, and from pairwise preferences. The methods are developed within the regularized kernel method framework. The reduced set approximation corresponding to the Nyström approximation of the kernel matrix is considered as a way to scale kernel methods for large data sets, and adapt linear solvers for training non-linear models.

Computational and memory efficiency are key aspects addressed in designing the methods. The developed algorithms are to be such that they will not explicitly form the data or kernel matrices corresponding to all the pairwise preferences in the training data. Rather, the preferences are modeled only implicitly, using computational shortcuts based on matrix algebra, sorting operations and advanced data structures.

Performance estimation by cross-validation. We study the properties of different cross-validation methods, with special focus on performance estimation for ranking problems. The considered approaches included the leave-pair-out method, the leave-query-out method, and both the pooling and averaging approaches. Since straightforward implementations of these methods would be computationally very costly, developing computational shortcuts is central for making the methods in practice useful. We produce an experimental study on AUC-estimation by cross-validation to better understand the properties of the leave-pair-out method, as well as the pooling and averaging methods. Further, we study the effects of combining the reduced set approximation and cross-validation, and derive computationally efficient algorithms for reliable performance estimation in this setting.

Application study. The task of protein-protein interaction extraction from scientific literature provides a case study in the use of kernel methods

and reduced set approximation, AUC-estimation, and the practical importance of efficient and reliable cross-validation algorithms.

1.3 Organization of the thesis

The main contribution of this thesis are the five original research articles, that together form the second half of this thesis. The first half of the thesis (Chapters 1−6) provides an introduction, where the background of the research work is described in detail, and the main contributions of the papers, and their connections to the overall research questions are described. Chapter 2 provides an overview of regularized kernel methods. In Chapter 3 we formalize the general ranking problem and discuss related work, proceeding in Chapter 4 to introduce the novel RankRLS algorithm, as well as the proposed extensions to the RankSVM method. Chapter 5 provides an overview of a number of challenges in performance evaluation via cross-validation, as well as our proposed solutions. Chapter 6 concludes the first part of the thesis, provides an overview of the second part, and discusses possible future directions of research.

Chapter 2

Regularized kernel methods

In this section we introduce the family of regularized kernel methods. We present an abstract definition of the supervised learning problem, and show how the problem can be solved in the regularized risk minimization framework. The introduced approach consists of selecting a prediction function which balances the tradeoff between how well the function fits to training data, and the complexity of the selected function. The considered prediction functions are linear, with respect to a feature space into which the data is mapped to. The kernel trick is introduced as a means to implicitly construct rich enough feature spaces, that allow one to adequetly model the learned concept using a linear function, even though the concept may be non-linear with respect to the original input space. In addition, the approach allows learning from structured data. For a detailed overview of regularized risk minimization and kernel based learning we refer to Schölkopf and Smola (2002); Shawe-Taylor and Cristianini (2004).

We use the following type of notation. $[m]$ denotes the index set $\{1 \dots m\}$. A bold lowercase letter $\mathbf{v} \in \mathbb{R}^n$ denotes a column vector of length n, whose i:th entry is given by v_i. The set of all $m \times n$ matrices with real coefficients is denoted by $\mathbb{R}^{m \times n}$. Given a matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$, $\mathbf{M}_{i,j}$ denotes its element in the i:th row and j:th column. Given two index sets $\mathcal{R} \subseteq [m]$ and $\mathcal{S} \subseteq [n]$, $\mathbf{M}_{\mathcal{R}\mathcal{S}}$ is the submatrix containing the rows and columns indexed by $\mathcal{R}$ and $\mathcal{S}$, respectively, and $\mathbf{M}_{\mathcal{R}}$ is the shorthand notation for $\mathbf{M}_{\mathcal{R}[n]}$. By $\mathbf{M}^{\mathrm{T}}$ we denote the transpose of $\mathbf{M}$, by $\mathbf{M}^{-1}$ its inverse if such exists, and by $\mathbf{M}^{+}$ its pseudoinverse (see e.g. Meyer (2000)).

2.1 Problem setting

The goal of supervised learning is to infer from a finite data sample a predictive model, that generalizes to new data that may not have been observed during the training phase. The data consists of inputs and labels, with in-

puts each being associated with some feature representation. The predictive model aims to capture a relationship that would allow predicting labels as a function of the features for new data for which the labels are unknown. We call the sample from which the model is inferred the training set. The most commonly made assumption is that the training data is sampled according to the same distribution as the new data on which predictions are made on.

For inferring a model we need a learning algorithm, that takes as input a training set, and outputs a prediction function. Following Shawe-Taylor and Cristianini (2004), we set three major design requirements for such algorithms. First, a learning algorithm should be computationally efficient, meaning polynomial scaling in running time and memory usage. In practice for a method to scale to reasonable training set sizes on modern computers it should have at most cubic running time scaling and quadratic memory usage, as well as efficient methods for model selection and performance estimation. For large scale learning problems, where data is abundant, close to linear scaling is preferred. Second, a learning algorithm should be robust, meaning the ability to tolerate noisy data measurements and learn relations that are probabilistic in nature, rather than exact. Finally, we require statistical stability meaning that the algorithm is likely to identify models that capture properties of the true underlying source generating the data, rather than just properties of a particular training set. Next, we formalize our problem setting.

Let the *input space* $\mathcal{X}$, and *output space* $\mathcal{Y}$ be sets. We are supplied with a *training set* Z containing inputs, and associated label information, defined as $Z = (X, Y) \in \mathcal{X}^m \times \mathcal{Y}$. By $X = (x_1, \ldots, x_m) \in \mathcal{X}^m$ we denote the set of m inputs belonging to the training set. By $Y \in \mathcal{Y}$ we denote a structured object containing the *label* information associated with X. For example, in *binary classification*, we may define $\mathcal{Y} = \{-1, 1\}^m$, where 1 denotes the so-called *positive*, and -1 the *negative class*, whereas in *regression* we may define $\mathcal{Y} = \mathbb{R}^m$. In these settings, there is exactly one label per each input. The reason we however use more general definition for the output space is that in ranking problems $\mathcal{Y}$ has often complex structure. The labels may have dependencies between them, and in some settings they are associated with pairs of inputs rather than with individual inputs (see Chapter 3).

A *learning algorithm*

$$\mathcal{A} : \bigcup_{m \in \mathbb{N}} \mathcal{X}^m \times \mathcal{Y} \to \mathcal{H}, \quad Z \mapsto f$$

takes as input a finite training set Z, and outputs a *prediction function* $f : \mathcal{X} \to \mathbb{R}$, which aims to model the dependency between the inputs and the labels. By $\mathcal{H}$ we denote some as of yet undefined *hypothesis space*, from which f is chosen. We overload our notation as follows. Let $X \in \mathcal{X}^m$, $m \in \mathbb{N}$ be a sequence of inputs. Then by $f(X) \in \mathbb{R}^m$ we denote the vector

of predictions for this sample. Regardless of the label structure, the learned function predicts a single real valued output per input. The setting matches regression where the labels are also real-valued, and in binary classification a threshold can be set to decide whether the prediction denotes positive or negative class. It also turns out that using such a scoring function is also sufficient for representing ranking models, even if the labels are not supplied directly as real valued scores during training (see Chapter 3).

A *loss function*

$$l : \bigcup_{m \in \mathbb{N}} \mathbb{R}^m \times \mathcal{Y} \mapsto [0, \infty)$$

measures how well the predicted labels and true labels for a data set match[1]. The exact form of a suitable loss function depends on the problem. The goal of learning is to find a prediction function that incurs minimal average loss on future data. This can be achieved by minimizing the *expected risk*, also known as the *generalization error*, defined as

$$R(f) = \mathrm{E}_{(\overline{X},\overline{Y})\sim D}[l(f(\overline{X}), \overline{Y})],$$

where E denotes expected value, D is the probability distribution that generated our training data, and $(\overline{X}, \overline{Y})$ is a randomly sampled data set.

In practice we never have access to the true underlying distribution, and are rather limited to using the *empirical risk*

$$\hat{R}(f) = l(f(X), Y), \tag{2.1}$$

which is simply the loss computed on the training set.

Should we choose a too simple hypothesis space, we may encounter the problem of *underfitting*, where none of the functions in $\mathcal{H}$ can accurately model the true underlying relation. As a historical example, a main criticism against the classic perceptron algorithm was that due to its restriction to models that are linear with respect to the input space, it was unable to learn non-linear relations such as XOR (Minsky and Papert, 1969). On the other hand, considering too complex models leads to loss of statistical stability, as the empirical risk no longer provides reliable information about the true generalization error of the considered functions. For example, using polynomial interpolation, one can fit a training set of size m using a $m-1$:th degree polynomial and a single feature, in order to perfectly regress any real valued labels for the set[2]. It is however highly unlikely that such a model

[1] In contrast to the more typical approach of defining losses as a sum over comparisons of individual predictions and labels (see e.g. Vapnik (1995)), we use a more general formulation, since ranking based losses do not typically admit such a representation. A similar type of formalization was used in Joachims (2002b); Lan et al. (2009).

[2] Assuming the feature does not take on differing values for training data points having equal labels

would perform well on data not part of the training set. The phenomenon where model is fitted to random characteristics of the training set that do not generalize to new data is referred as *overfitting*. To find balance between underfitting and overfitting, we need to find ways to control the structure and complexity of the considered hypothesis space. In regularized kernel methods these goals are achieved by using *kernel functions* that give the learning methods sufficient expressive power to avoid underfitting, and by using *regularization* which penalizes complex functions in order to avoid overfitting.

2.2 Kernel feature spaces

First, we recall some basic concepts that are used to define kernel induced feature spaces. While the considered concepts can also be generalized to complex numbers, we restrict our considerations to the field of real numbers.

Definition 1 (Inner product space)**.** *An inner product space $\mathcal{V}$ is a vector space over $\mathbb{R}$ with a map $\langle \cdot, \cdot \rangle : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$ known as the inner product, which is symmetric, linear in each argument and satisfies*

$$\langle x, x \rangle \geq 0, \text{ with } \langle x, x \rangle = 0 \text{ iff } x = 0$$

The inner product directly induces a *norm* $\|x\| = \sqrt{\langle x, x \rangle}$ and a *distance metric* $d(x, x') = \|x - x'\| = \sqrt{\langle x, x \rangle - 2\langle x, x' \rangle + \langle x' x' \rangle}$ on $\mathcal{V}$.

We recall that a *Cauchy sequence* $x_1, x_2 \ldots$ in $\mathcal{V}$ denotes a sequence of elements of $\mathcal{V}$, for which for every positive real number ϵ, we can find such $N \in \mathbb{N}$ that $d(x_i, x_j) < \epsilon$ for all $i, j > N$.

Definition 2 (Hilbert space)**.** *A Hilbert space $\mathcal{F}$ is an inner product space that is complete, with respect to the norm defined by the inner product, meaning that $\mathcal{F}$ contains the limit point of every Cauchy sequence in $\mathcal{F}$.*

For any mapping

$$\Phi : \mathcal{X} \to \mathcal{F},$$

a function

$$k(x, x') = \langle \Phi(x), \Phi(x') \rangle$$

which computes the inner product of mapped data points is called a *kernel function*. We call the Hilbert space $\mathcal{F}$, where the data is mapped to, a *feature space*. This mapping is not unique, in the sense that for a given kernel one may define $\Phi_1 \neq \Phi_2$ such that $k(x, x') = \langle \Phi_1(x), \Phi_1(x') \rangle = \langle \Phi_2(x), \Phi_2(x') \rangle$ for all choices of x and x'. In some ceases we may not be able to explicitly compute the feature map at all. However, the kernel function allows us access to inner products between the mapped data points, which is already

sufficient information for developing powerful learning algorithms. Thus provided that the kernel function is computationally efficient to evaluate, our learning algorithm can operate in high dimensional feature spaces, without explicitly mapping the data. As a consequence, linear algorithms that can be fully formulated in terms of dot products can be made nonlinear by replacing the dot product with a non-linear kernel function. An algorithm which is linear with respect to $\mathcal{F}$ will then be nonlinear with respect to $\mathcal{X}$. Further, kernel functions allow learning from structured data. The input space $\mathcal{X}$ can consist of complex objects such as pictures, DNA-sequences or graphs, as long as a suitable kernel function can be defined between such objects. Informally, a good kernel function should aim to capture the notion of similarity, producing the higher values the more similar the function arguments are with respect to each other.

Formally, a function $k(x, x')$, which is either continuous or has a countable domain, corresponds to an inner product $\langle \Phi(x), \Phi(x') \rangle$ in a feature space if and only if it is *finitely positive semi-definite*, defined as $\sum_{i,j=1}^{m} \alpha_i \alpha_j k(x_i, x_j) \geq 0$ for any $m \in \mathbb{N}$, where $x_k \in \mathcal{X}$ and $\alpha_k \in \mathbb{R}$ $1 \leq k \leq m$. Equivalently, for a given input sequence X, a kernel function gives rise to a symmetric matrix

$$\mathbf{K} = \begin{pmatrix} k(x_1, x_1) & \cdots & k(x_1, x_m) \\ \vdots & \ddots & \vdots \\ k(x_m, x_1) & \cdots & k(x_m, x_m) \end{pmatrix},$$

known as the *kernel matrix* of X, which is *positive semi-definite*, meaning that $\mathbf{A}^{\mathrm{T}}\mathbf{K}\mathbf{A} \geq 0$ for all $\mathbf{A} \in \mathbb{R}^m$.

Next, we consider a general approach to constructing a feature space corresponding to any given kernel function. Following Schölkopf and Smola (2002) we consider the following mapping into a function space:

$$\Phi : \mathcal{X} \to \mathbb{R}^{\mathcal{X}}, \quad x \mapsto k(\cdot, x),$$

where $\mathbb{R}^{\mathcal{X}}$ is the space of functions mapping $\mathcal{X}$ into $\mathbb{R}$, and $k(\cdot, x)$ is a function that computes the kernel evaluation between the argument and x.

First, we construct a vector space $\mathcal{V}$ as:

$$\mathcal{V} = \left\{ f \in \mathbb{R}^{\mathcal{X}} \;\middle|\; f(\cdot) = \sum_{i=1}^{l} \alpha_i k(\cdot, x_i), \alpha_i \in \mathbb{R}, x_i \in \mathcal{X} \right\},$$

defining addition as $f, g \in \mathcal{V} \Rightarrow (f + g)(x) = f(x) + g(x)$.

Next, we turn the vector space into an inner product space. Let $f(\cdot) = \sum_{i=1}^{l} \alpha_i k(\cdot, x_i)$ and $g(\cdot) = \sum_{i=1}^{n} \beta_i k(\cdot, x'_i)$ be two elements in $\mathcal{V}$. Then we can define an inner product as

$$\langle f, g \rangle = \sum_{i=1}^{l} \sum_{j=1}^{n} \alpha_i \beta_j k(x_i, x'_j).$$

It can be shown (Schölkopf and Smola, 2002) that this mapping indeed satisfies all the properties required of an inner product, and induces the following norm:

$$\|f\| = \sqrt{\sum_{i,j=1}^{l} \alpha_i \alpha_j k(x_i, x_j)}$$

Finally, (Schölkopf and Smola, 2002) note that in order to simplify the mathematical analysis of the properties of the feature space, it is useful to turn it into a Hilbert space. This is achieved by adding to it the limit points of all the series that are convergent in the norm, resulting in the *reproducing kernel Hilbert space* (RKHS) of functions, defined as

$$\mathcal{F} = \left\{ f \in \mathbb{R}^{\mathcal{X}} \;\middle|\; f(\cdot) = \sum_{i=1}^{\infty} \alpha_i k(\cdot, x_i), \alpha_i \in \mathbb{R}, x_i \in \mathcal{X}, \|f\| < \infty \right\}.$$

2.3 Regularized risk minimization

Next, we present the framework of kernel-based regularized risk minimization. The kernel functions introduced earlier allow us to represent rich enough hypotheses to allow underfitting. Conversely, a mechanism known as regularization is introduced for avoiding the problem of overfitting. We assume that our hypothesis space consists of the set of linear functions in a kernel induced feature space, where the kernel has been a priori chosen according to its suitability for the application domain in question.

A large class of machine learning algorithms (see e.g. Evgeniou et al. (2000) for discussion) are based on minimizing the *regularized risk functional*

$$\mathcal{A}(Z) = \operatorname*{argmin}_{f \in \mathcal{F}} J(f),$$

where

$$J(f) = \hat{R}(f) + \lambda \|f\|^2. \tag{2.2}$$

The first term in (2.2) is the empirical risk (2.1) measuring how well the considered hypothesis f fits the training data, as measured by a convex loss function l. The second term called the *regularizer* measures the complexity of the hypothesis with the RKHS norm, and $\lambda > 0$ is the regularization parameter. We require the convexity of the loss function in order to guarantee that we can always find a globally optimal solution to the minimization problem.

We define $k(x, X) = (k(x, x_1), \ldots, k(x, x_m)) \in (\mathbb{R}^m)^{\mathrm{T}}$, where $x \in \mathcal{X}$. According to the *representer theorem* (Schölkopf et al., 2001) any minimizer

of (2.2) admits a representation of the form

$$f(x) = \sum_{i=1}^{m} \alpha_i k(x, x_i) = k(x, X)\mathbf{a}, \tag{2.3}$$

where $\alpha_i \in \mathbb{R}$, $\mathbf{a} = (\alpha_1, \ldots, \alpha_m)^{\mathrm{T}} \in \mathbb{R}^m$, This representation allows us to express the solution solely in terms of kernel evaluations between the argument and the training examples. We call this the *dual formulation* and $\mathbf{a}$ the *dual solution*.

The minimizer (2.3) can be re-written as

$$\begin{aligned} f(x) &= \sum_{i=1}^{m} \alpha_i \langle \Phi(x), \Phi(x_i) \rangle \\ &= \langle \Phi(x), \sum_{i=1}^{m} \alpha_i \Phi(x_i) \rangle \\ &= \langle \Phi(x), \mathbf{w} \rangle. \end{aligned}$$

We call this the *primal formulation*. Let us assume that the feature representation Φ corresponding to the kernel function is finite dimensional. For any kernel function and any finite set of data points, such a feature representation can be for example constructed using the *empirical kernel map*, as discussed in the next section. Then, we can form a data matrix $\Phi(X) = (\Phi(x_1) \ldots \Phi(x_m))$, and the primal solution becomes the column vector $\mathbf{w} = \Phi(X)\mathbf{a}$. Conversely, given the primal solution $\mathbf{w}$, a dual solution can be recovered by solving the linear system $\Phi(X)\mathbf{a} = \mathbf{w}$, where $\mathbf{a}$ is unknown. The consistency of this system of linear equations is guaranteed by the representer theorem. Thus $\mathbf{a}$ can be recovered using for example the Moore-Penrose pseudoinverse of $\Phi(X)$ (Meyer, 2000).

Depending on whether we solve the primal problem directly in the feature space, or its dual counterpart, we can re-formulate (2.2) either as

$$J(\mathbf{w}) = l(\Phi(X)^{\mathrm{T}}\mathbf{w}, Y) + \lambda \mathbf{w}^{\mathrm{T}}\mathbf{w}, \tag{2.4}$$

or as

$$J(\mathbf{a}) = l(\mathbf{K}\mathbf{a}, Y) + \lambda \mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{a}. \tag{2.5}$$

The primal problem (2.4) may be solved whenever we can efficiently form the feature representations of the training examples directly. This is most straightforward when using the linear kernel, but approaches such as explicitly modeling low-order polynomial features (Chang et al., 2010), decomposing the kernel matrix (Tsuda, 1999; Schölkopf et al., 1999; Harmeling et al., 2002), or using randomized features for shift invariant kernels (Rahimi and Recht, 2007) have also been proposed. Efficient training algorithms whose

training complexity is often only linear in the number of training examples, and the dimensionality of the feature space, have been introduced for solving the primal problem (see e.g. Joachims (2006); Smola et al. (2007); Shwartz et al. (2007)).

Whenever we can efficiently access only the kernel evaluations between training points, dual algorithms are applied for solving (2.5). Not considering the cost of kernel evaluations, the training costs of fastest dual learning algorithms typically vary between $O(m^2)$ and $O(m^3)$ (see e.g. Bottou and Lin (2007)).

In this thesis we introduce both dual and primal learning algorithms. The dual algorithms allow efficient use of kernels, which may be necessary for learning non-linear concepts, or for learning from structured data. The primal algorithms allow scaling the methods to larger data sets whenever the linear kernel is sufficient for achieving good prediction accuracy, or the feature map can otherwise efficiently be explicitly constructed. The introduced algorithms are described in more detail in Chapter 4 of the thesis, as well as in Papers I, II and III.

2.4 Empirical kernel map

The representer theorem guarantees that in regularized risk minimization with quadratic regularizer, the optimal solution can be expressed as a linear combination of the images of the training data. Thus, during training it is enough to consider the subspace of the feature space spanned by the images of the training data. It turns out that there is a simple technique to construct a feature representation, that corresponds to the kernel function for members of this subspace.

Following Schölkopf et al. (1999), we note that for a finite set of inputs X, one can construct a feature representation corresponding to a given kernel function k using the *empirical kernel map*

$$\Phi_X : \mathcal{X} \to \mathbb{R}^r,$$

where

$$\Phi_X(x) = (\mathbf{K}^{1/2})^+(k(x, X))^{\mathrm{T}}, \tag{2.6}$$

and r is larger or equal to the rank of $\mathbf{K}$. $\mathbf{K}^{1/2}$ denotes a symmetric decomposition of $\mathbf{K}$, meaning that $\mathbf{K}^{1/2}(\mathbf{K}^{1/2})^{\mathrm{T}} = \mathbf{K}$. In practice such decomposition can be constructed from the eigen-decomposition of $\mathbf{K}$. We recall the following property of the Moore-Penrose pseudoinverse $\mathbf{M}^+$ of any given matrix $\mathbf{M}$: $\mathbf{M} = (\mathbf{M}^{\mathrm{T}})^+\mathbf{M}^{\mathrm{T}}\mathbf{M}$.

For the set of inputs X that were used to define the mapping, the feature

representations are defined as

$$\begin{aligned}\Phi_X(X) &= (\mathbf{K}^{1/2})^{+}\mathbf{K} \\ &= (\mathbf{K}^{1/2})^{+}\mathbf{K}^{1/2}(\mathbf{K}^{1/2})^{\mathrm{T}} \\ &= (\mathbf{K}^{1/2})^{\mathrm{T}}.\end{aligned}$$

Since $\Phi_X(X)^{\mathrm{T}}\Phi_X(X) = \mathbf{K}$, the empirical kernel map allows us to construct a feature representation that corresponds to the kernel function k for all the training data. Algorithms that depend only on properties that are determined by inner products between the training examples can thus be trained using the features generated by the empirical kernel map. For example, Tsuda (1999) proposed using the empirical kernel map for generating feature representations in order to train support vector machines. A downside of the approach is the $O(m^3)$ cost required to decompose a full rank kernel matrix (Golub and Loan, 1989).

Finally, we note that Schölkopf et al. (1999) gives also an alternative definition of empirical kernel map as

$$\Phi_X(x) = (k(x, X))^{\mathrm{T}},$$

where the map simply corresponds to using kernel evaluations directly as features. Using such feature representation when training support vector machines was considered in more detail for example in Schölkopf et al. (2002). However, we do not consider this form of empirical kernel map further in this thesis.

2.5 Reduced set approximation

The kernel matrix can form a computational bottleneck for kernel based methods, since computing the matrix and storing its values results in at least quadratic time and memory complexities. Further, computing predictions according to (2.3) requires loading the whole training set to memory, and computing kernel evaluations between each training input and the new input. *Reduced set approximation* (see e.g. Poggio and Girosi (1990); Smola and Schölkopf (2000); Rifkin et al. (2003); Quiñonero-Candela and Rasmussen (2005)), also known as *sparse approximation* or the *subset of regressors* method is a standard approach to reducing the computational costs of training and prediction for kernel methods.

Instead of using all the m training examples to represent the learned hypothesis, as in (2.3), we limit ourselves to considering their subset indexed by $\mathcal{R} \subseteq [m]$, where $|\mathcal{R}| << m$. We call this subset the *basis vectors*. Now we consider a solution that allows only the basis vectors to have nonzero

coefficient, that is,

$$f(x) = \sum_{i \in \mathcal{R}} \beta_i k(x, x_i) = k(x, X_{\mathcal{R}})B, \tag{2.7}$$

where $X_{\mathcal{R}}$ is the subsequence of training inputs indexed by $\mathcal{R}$, and $B \in \mathbb{R}^{|\mathcal{R}|}$ is a column vector containing the $|\mathcal{R}|$ coefficients in the reduced set solution. A number of different selection schemes for choosing the set $\mathcal{R}$ have been proposed in the literature (see e.g. Smola and Schölkopf (2000); Rifkin et al. (2003); Quiñonero-Candela and Rasmussen (2005); Zhang et al. (2008); Kumar et al. (2009)). We restrict our considerations to simple uniform sampling without replacement, as it has been noted in Rifkin et al. (2003); Kumar et al. (2009) to be both computationally highly efficient, and in terms of prediction performance competitive with more sophisticated approaches.

Quiñonero-Candela and Rasmussen (2005) noted that using the reduced set approximation is equivalent to using the following type of modified kernel function

$$\tilde{k}(x, x') = k(x, X_{\mathcal{R}})(\mathbf{K}_{\mathcal{R}\mathcal{R}})^{-1}k(x', X_{\mathcal{R}})^{\mathrm{T}}, \tag{2.8}$$

The kernel matrix for the training set, constructed with (2.8), becomes

$$\widetilde{\mathbf{K}} = (\mathbf{K}_{\mathcal{R}})^{\mathrm{T}}(\mathbf{K}_{\mathcal{R}\mathcal{R}})^{-1}\mathbf{K}_{\mathcal{R}}.$$

$\widetilde{\mathbf{K}}$ is sometimes known as the *Nyström approximation* of $\mathbf{K}$.

Using the Nyström approximation $\widetilde{\mathbf{K}}$ directly as an approximation of $\mathbf{K}$ by simply inserting it to the dual problem (2.5) leads to the method known as the *Nyström method* (Williams and Seeger, 2001). Rifkin et al. (2003) criticized this approach, because it leads to using different kernel functions during training and prediction. However, in can be shown that the dual solution obtained by training a regularized kernel method using the Nyström approximation can be transformed to an equivalent prediction function that corresponds to the reduced set approximation (2.7) based on the original kernel function. This was shown in Rifkin et al. (2003) for the case of regularized least-squares regression, a more general formulation of the result is presented in Paper III. Rifkin et al. (2003) also empirically demonstrated, that the reduced set approach outperforms the naive Nyström method in predictive performance.

Combining the empirical kernel map and the Nyström approximation, we can create computationally efficiently a feature representation that allows training reduced set approximations of kernel methods in the primal. The decomposition strategy, and the mapping from a learned primal solution to a reduced set solution of type (2.7) are described in Paper III. The cost of performing the mapping is $O(m|\mathcal{R}|^2)$, which results in a $m \times |\mathcal{R}|$ sized data matrix. Further, Paper III introduces an efficient algorithm for removing basis vectors from the mapping, without having to fully re-compute

it. This technique has applications in cross-validation, as also discussed in Section 5.6.

Section 3.5 in Paper I formalizes the reduced set approximation for the introduced RankRLS algorithm. In Paper V a reduced set approximation method is implemented in order to scale the learner to the considered data set sizes, in the experiments on protein-protein interaction extraction from scientific literature.

2.6 Kernel functions

The choice of a suitable kernel function is an application dependent problem, and there exists a large body of literature on kernels for different application domains (see e.g. Shawe-Taylor and Cristianini (2004) and references therein). Next we present some of the most widely used kernels for vectorial data, and introduce a family of kernels for learning from graph structured data.

First, let us assume that $\mathcal{X} = \mathbb{R}^n$, and that $x, x' \in \mathcal{X}$. Commonly used kernel functions that have a closed form solution include the following:

The *linear kernel* is defined as

$$k(x, x') = \langle x, x' \rangle,$$

meaning that it is simply the dot product between two real-valued vectors. Most kernel methods are generalizations of originally linear methods, and become equivalent to their linear counterparts when the linear kernel is used.

The *polynomial kernel* is defined as

$$k(x, x') = (\langle x, x' \rangle + c)^d, \tag{2.9}$$

where $d \in \mathbb{N}$ and $c \geq 0$ are parameters.

The *Gaussian kernel*, also known as the *radial basis function* (RBF) *kernel* can be defined as

$$k(x, x') = e^{-\gamma \|x - x'\|^2}, \tag{2.10}$$

where $\gamma \in \mathbb{R}^+$ is a parameter.

Further, a large number of kernels have been introduced for non-vectorial data (see e.g. Shawe-Taylor and Cristianini (2004)). These include for example set kernels, string kernels, graph kernels and semantic kernels for textual data. Often these kernels do not have closed form expressions for computing them, but rather the kernel evaluations rely on recursive relations solved by dynamic programming techniques.

Gärtner et al. (2003) introduced a number of kernels for graph data. Let us assume, that our data points are directed weighted graphs, where each

node has a number of binary labels. We consider the setting, where each feature corresponds to a pair of labels that are connected in the graph. Let V be the set of vertices in the graph and $\mathcal{L}$ be the set of possible labels vertices can have. We represent the graph with an adjacency matrix $\mathbf{A} \in \mathbb{R}^{|V|\times|V|}$, whose rows and columns are indexed by the vertices, and $\mathbf{A}_{i,j}$ contains the weight of the edge connecting $v_i \in V$ and $v_j \in V$ if such an edge exists, and zero otherwise. Further, we represent the labels as a label allocation matrix $\mathbf{L} \in \mathbb{R}^{|\mathcal{L}|\times|V|}$ so that $\mathbf{L}_{i,j} = 1$ if the j-th vertex has the i-th label and $\mathbf{L}_{i,j} = 0$ otherwise.

A *walk* of length k in a graph is a sequence of k edges, where the end node of each edge must be the same as the starting node of the following edge. Thus a walk is one possible route one could traverse from the starting node of the first edge to the end node of the last edge in the sequence. If the graph contains cycles, a walk may pass through the same edge multiple times. A *path* is a walk that is allowed to contain any edge only once. The matrix power $\mathbf{A}^n_{i,j}$ contains the multiplied weights on all walks of length n connecting the vertices v_i and v_j in the graph. The feature representation, corresponding to all label pairs connected by walks of length n can be computed as $\mathbf{L}\mathbf{A}^n\mathbf{L}^{\mathrm{T}}$. The sum $\sum_{i=1}^{n} \mathbf{L}\mathbf{A}^i\mathbf{L}$ combines together the pairwise features from all walks of lengths $1 \ldots n$. Finally, let us assume that $\|\mathbf{A}\| < 1$, where $\|\mathbf{A}\|$ denotes the spectral radius, being the supremum of the absolute values of the eigenvalues of $\mathbf{A}$, holds. Then, the sum of powers of $\mathbf{A}$ converges, allowing us to compute the pairwise features for paths of all lengths as $\sum_{i=1}^{\infty} \mathbf{L}\mathbf{A}^i\mathbf{L} = \mathbf{L}((\mathbf{I} - \mathbf{A})^{-1} - \mathbf{I})\mathbf{L}^{\mathrm{T}}$, where $\mathbf{I}$ denotes the identity matrix. This is a special case of a more general formulation presented in Gärtner et al. (2003).

To adapt the graph kernel framework for different applications, one needs to define suitable graph representations, labels, weightings and decide on the length of paths considered. Paper V introduces one such adaptation of the general framework for the purpose of extracting information about protein-protein interactions based on dependency graphs encoding syntactic information about sentences. The kernel is called all-paths graph kernel, though all-walks graph kernel would in fact be a more appropriate choice, as the kernel is actually based on general walks rather than just paths in the graph. Related work includes the graph representations proposed in Pahikkala et al. (2006b) for the task of parse ranking, that are used in the experiments described in Papers I and III.

Kernel functions have a number of closure properties, which allows one to define new kernels by combining known ones. For example, multiplication by a positive real number, as well as addition and multiplication of kernel functions produces valid new kernel functions. For a detailed survey of these properties, see Schölkopf and Smola (2002); Shawe-Taylor and Cristianini (2004).

2.7 Regularized least-squares

Regularized least-squares (RLS) is a regularized kernel method suitable for both regression and classification. The method is known under various aliases in the machine learning and statistics literature. For example, in the statistics literature the linear RLS is often known as *ridge regression* (Hoerl and Kennard, 1970), for which reason the kernelized RLS is sometimes called *kernel ridge regression* (Saunders et al., 1998). Inspired by neural networks and support vector machines respectively, the method has also become known as the *regularization network* (Poggio and Girosi, 1990) and as the *least-squares support vector machine* (Suykens and Vandewalle, 1999). A Bayesian interpretation of the method results in *Gaussian process regression* (Williams and Rasmussen, 1996). The term RLS has been used for example by Poggio and Smale (2003); Rifkin et al. (2003); Cesa-Bianchi (2007), and it is by this name we refer to the method from now on.

First, we assume that for a training set of size m, the output space is defined as $\mathcal{Y} = \mathbb{R}^m$, and the labels in the training set are stored in the column vector $\mathbf{y} \in \mathbb{R}^m$. By inserting the *squared loss*, defined as

$$l(f(X), \mathbf{y}) = \sum_{i=1}^{m} (f(x_i) - y_i)^2, \tag{2.11}$$

into the regularized risk functional (2.2) we derive the objective function for the RLS

$$J(f) = \sum_{i=1}^{m} (f(x_i) - y_i)^2 + \lambda \|f\|^2. \tag{2.12}$$

This can be equivalently re-written in matrix notation either in the primal, or in the dual form.

The primal form of (2.12) is

$$J(\mathbf{w}) = (\Phi(X)^{\mathrm{T}}\mathbf{w} - \mathbf{y})^{\mathrm{T}}(\Phi(X)^{\mathrm{T}}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^{\mathrm{T}}\mathbf{w}.$$

Since this equation is convex with respect to $\mathbf{w}$, we can find a global minimizer by setting the gradient equal to 0. The gradient of this equation is

$$\nabla J(\mathbf{w}) = -2\Phi(X)\mathbf{y} + 2(\Phi(X)\Phi(X)^{\mathrm{T}} + \lambda \mathbf{I})\mathbf{w},$$

and setting it to 0 results in the system of linear equations

$$(\Phi(X)\Phi(X)^{\mathrm{T}} + \lambda \mathbf{I})\mathbf{w} = \Phi(X)\mathbf{y}. \tag{2.13}$$

Since the matrix on the left side of the equation is positive definite, an unique solution to the equation exists.

The dual form of (2.12) is

$$J(\mathbf{a}) = (\mathbf{K}\mathbf{a} - \mathbf{y})^{\mathrm{T}}(\mathbf{K}\mathbf{a} - \mathbf{y}) + \lambda \mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{a}.$$

Again, the equation is convex with respect to $\mathbf{a}$, the gradient of this equation is

$$\nabla J(\mathbf{a}) = 2\mathbf{K}(-\mathbf{y} + (\mathbf{K} + \lambda \mathbf{I})\mathbf{a}),$$

and we can recover a dual solution by solving the linear system

$$(\mathbf{K} + \lambda \mathbf{I})\mathbf{a} = \mathbf{y}. \tag{2.14}$$

The property that both the primal and dual solutions can be found by solving systems of linear equations yields a number of computational benefits. When solving the system only once, Cholesky decomposition based methods can be used to solve the system. The computational cost of solving (2.13) is $O(n^3 + n^2 m)$, assuming that the feature space is $\mathbb{R}^n$. For the dual form (2.14) the cost is $O(m^3)$, not taking into account the cost of constructing the kernel matrix which varies according to the kernel function.

The RLS is naturally suited to solving regression problems, but it can also be used to solve binary classification problems by regressing the class labels. In this context its use can be criticized due to the two-sided nature of the loss function; examining (2.11) it can be seen, that the loss unnecessarily penalizes correct class predictions whose magnitude is too large. However, it has been empirically observed that RLS tends to perform as well in classification as methods based on one sided loss-functions (Suykens and Vandewalle, 1999; Rifkin, 2002; Poggio and Smale, 2003; Zhang and Peng, 2004).

Perhaps one of the main benefits of using the RLS loss is that much of the computations done in solving the linear systems of equations can be re-used, when doing small modifications to the system. First, we consider changing the value of the regularization parameter λ. Since one often does not know the suitable value for the parameter in advance, it is usually necessary to solve the RLS problem many times for different values of λ in order to find a good value for the parameter. By computing the eigen-decomposition of $\mathbf{K}$, or the "economy size" singular value decomposition of $\Phi(X)$, one may re-solve the system for several different values of λ with no growth in asymptotic cost, as discussed in Rifkin and Lippert (2007).

Additionally, computational shortcuts can be used to update the solutions $\mathbf{w}$ or $\mathbf{a}$, when removing data points from the training set. Such short-cuts can be used to derive efficient methods for cross-validation, where repeatedly some data points are left out of training set for the purpose of evaluating learner performance. An efficient leave-one-out algorithm for RLS is a classical result (Vapnik, 1979). This result has been recently extended to repeated hold-out and cross-validation with arbitrary sized, possibly overlapping holdout sets in Pahikkala et al. (2006a) and An et al.

(2007), independently of each other. When using the linear kernel such computational short-cuts may analogously be extended to feature selection, where the model is updated to reflect the removal or addition of features in the training set (Pahikkala et al., 2010b). Similar type of shortcuts allow efficient online learning (see e.g. Pahikkala et al. (2011)).

Finally, conjugate gradient optimization (Shewchuk, 1994) based training methods allow scaling linear RLS training to very large and high-dimensional but sparse data sets (Rifkin, 2002).

The RankRLS algorithm (see Chapter 4 as well as Paper I), can be considered as an extension of RLS, where regression is done on the level of pairs of data points. The RLS method is used in the experimental evaluations in Papers I, III, IV, and V

2.8 Support vector machines

The (soft margin) *support vector machine* (SVM) algorithm (Cortes and Vapnik, 1995; Vapnik, 1995) is among the most popular approaches to binary classification. We assume that $\mathcal{Y} = \{-1, 1\}^m$, and that the training labels are stored in a vector $\mathbf{y} \in \{-1, 1\}^m$.

The hinge loss is defined as

$$l(f(X), \mathbf{y}) = \sum_{i=1}^{m} \max(1 - y_i f(x_i), 0). \tag{2.15}$$

By inserting this into the regularized risk functional (2.2), we recover the SVM objective function, defined as

$$J(f) = \sum_{i=1}^{m} \max(1 - y_i f(x_i), 0) + \lambda \|f\|^2.$$

The hinge loss provides a convex approximation of the non-convex $0-1$ loss that computes the fraction of misclassifications on binary classification tasks. The SVM algorithm is known to have good performance on binary classification tasks, and it is one of the most widely used kernel based learning methods. However, the SVM optimization problem is made more difficult than the RLS one by the fact that no closed form solution exists, and also the loss function is non-smooth, meaning that it is not differentiable everywhere. The RankSVM method (Herbrich et al., 1999; Joachims, 2002b) considered in Chapter 4 extends the SVM for ranking.

For training kernel-based SVMs a large number of dual algorithms have been proposed (see Bottou and Lin (2007) for overview). According to Bottou and Lin (2007), the methods scale between $O(m^2)$ and $O(m^3)$ in

the number of training examples, depending on the level of regularization applied.

For the linear case, the use of *subgradient* based methods has been proposed in Joachims (2006); Smola et al. (2007); Teo et al. (2010). Joachims (2006) called the approach the *cutting plane method*, the equivalence to subgradient based *bundle optimization* was shown in Smola et al. (2007); Teo et al. (2010). The concept of subgradient generalizes the notion of gradient for convex functions. Formally, for a convex risk functional $\hat{R}(f)$, $\mathbf{s}' \in \mathcal{F}$ is called a subgradient of $\hat{R}$ at f', if and only if

$$\hat{R}(f) \geq \hat{R}(f') + \langle f - f', \mathbf{s}' \rangle \ \forall f.$$

If $\hat{R}$ is differentiable at f', then the set of subgradients at that point consists of simply the gradient. The subgradient information can be used to minimize the regularized risk functional using bundle optimization methods (Smola et al., 2007; Teo et al., 2010). The main idea is to use subgradient evaluations to iteratively construct a piecewise linear lower bound approximation of the empirical risk. The optimization problem where this lower bound replaces the empirical risk in (2.2) can be solved by using quadratic programming, where the number of constraints depends on the number of computed subgradients. Theoretical and empirical analysis of the approach has shown that the method can be expected to converge to accurate solution long before the number of constraints is large enough to make the quadratic program expensive to solve, the main computational costs result rather from risk and subgradient evaluations (Joachims, 2006; Smola et al., 2007; Teo et al., 2010).

For the hinge loss, a primal formulation for the subgradient of the empirical risk at $\mathbf{w}'$ can be defined as $\mathbf{s}' = \Phi(X)\mathbf{g}$, where $\mathbf{g} \in \mathbb{R}^m$ is defined as

$$g_i = \begin{cases} 0 & \text{if } y_i \Phi(x_i)^{\mathrm{T}} \mathbf{w}' \geq 1 \\ -y_i & \text{otherwise} \end{cases} .$$

This approach has been implemented in a number of state-of-the art linear SVM solvers, including SVM$^{\text{perf}}$ (Joachims, 2006), BMRM (Teo et al., 2007) and the OCA (Franc and Sonnenburg, 2009) software packages. The methods have either linear or linearithmic scaling in the number of training examples. The method has also been used to implement state-of-the-art linear RankSVM solvers, as discussed in Chapter 4. The bundle method is described in detail in Section 3 of Paper II, where it is used to implement the introduced novel RankSVM training algorithm. The bundle method is also used to implement the basic SVM for running the experiments described in Section 5.4 in Paper IV.

Chapter 3

Learning to rank

The task of *learning to rank* refers the supervised learning problem, where the aim is to learn to order objects according to some criterion of goodness. Common applications include search engines that rank documents according to their match to user queries (Joachims and Radlinski, 2007; Liu, 2009), recommender systems (Minkov et al., 2010) and medical decision making (Suominen et al., 2006). In this section the general learning to rank setting is introduced, and important special cases such as *AUC maximization* and *ordinal regression* are related to the general setting. We adapt the widely used approach of modeling the ranking problem in terms of *pairwise preferences*. Two alternative approaches to learning to rank, the *pointwise* and the *listwise* approach, are also discussed. As pointers to relevant literature, Fürnkranz and Hüllermeier (2010) provides a compact overview of the area of pairwise preference learning, and Domshlak et al. (2011) discusses the use of pairwise preferences in a broader context in the field of AI, as well as connections to other fields where preference models are used. Liu (2009) provides an overview of the pointwise, pairwise and listwise approaches in the context of learning to rank for information retrieval.

3.1 Data

We again sample a training set $Z = (X, Y) \in \mathcal{X}^m \times \mathcal{Y}$ containing inputs representing the objects we aim to rank, and label information encoding information about how well the objects match the underlying ranking criterion. There exist two distinct settings with respect to how the label information is provided. We may learn from *scored data*, or from *pairwise comparisons*.

In a *global ranking* defined by scored data, we have a setting similar to regression, where $\mathcal{Y} = \mathbb{R}^m$ (assuming a set of m inputs), meaning that each input in a dataset is associated with a single real-valued score. In this setting, we may define our training set of m examples as $Z = (X, \mathbf{y})$, where

$X \in \mathcal{X}^m$ is the sequence of inputs, and $\mathbf{y} \in \mathbb{R}^m$ is a column vector of utility scores. In this case we may compare any two data points, denoted here as (x, y) and (x', y') as follows. If $y > y'$, then x is *preferred over* x', denoted as $x \succ x'$. If $y < y'$, the contrary is the case. Finally in case $y = y'$, we say that we are indifferent with respect to this choice, both choices are equally good.

The scored data may also be used to define a *partial order*. This is commonly the case for example in document ranking, where the inputs are pairs of user queries and documents, and the utility score reflects how well the document and query match together[1]. In this setting rankings are defined over individual queries, but it is not meaningful to compare data points related to different queries. At test time the ranking is always constructed for a single query at a time. In this setting, we adopt the term query to denote a subset of inputs over which a ranking is defined. We limit our considerations to the setting where each input belongs to one and only one query. Identical inputs are allowed to appear in different queries, but these are then considered distinct from each other rather than being the same input. The described framework is standard in the learning to rank for information retrieval literature (see e.g. Joachims (2002b); Lan et al. (2009)), and is also applicable to ranking problems in many other domains (Pahikkala et al., 2010d).

Formally, in the scoring setting with query structured data, the output space corresponding to m inputs is defined as $\mathcal{Y} = \mathbb{R}^m \times \mathcal{Q}$, where

$$\mathcal{Q} = \{\{q_1, \ldots, q_p\} | q_i \subseteq [m], q_i \cap q_j = \emptyset \text{ if } i \neq j\}$$

denotes the set of all possible query partitions for index set $[m]$. The global ranking can in this setting treated as the restricted case $\mathcal{Q} = \{[m]\}$. Pairwise preferences can be constructed as before for data points belonging to the same query, examples from different queries are said to be incomparable. In this setting, we may define a training set of m examples as $Z = (X, \mathbf{y}, Q)$, where $X \in \mathcal{X}^m$ is the sequence of inputs, $\mathbf{y} \in \mathbb{R}^m$ is a column vector of utility scores, and $Q \in \mathcal{Q}$ is the query partition of the training set.

When sampling data points from a global ranking, we may assume that the training set constitutes an i.i.d. sample from an unknown distribution over the input-utility pairs. However, with query structured data, the sampling is typically done on the level of queries, rather than on the level of individual data points. Here we may assume that the set of training queries

[1]Paper I uses the term *label ranking* to describe ranking with query-structured data. This choice of terminology reflects the idea, that the data points represent input-label pairs with joint feature representations, and are ordered according to how well the label matches the input. However, the term has been more commonly used to describe the situation where the set of labels is fixed, and the labels do not have features (Fürnkranz and Hüllermeier, 2010).

constitutes an i.i.d. sample from an unknown distribution over all the possible queries. The fact that the data points are not sampled independently of each other in the query setting can lead to complications when doing performance estimation via cross-validation, as discussed in more detail in Chapter 5.

In some applications it may be the case that we do not have access to the utility scores of the training examples, but rather are supplied only with pairwise comparisons. For example Joachims (2002a) proposed gathering such pairwise comparisons from implicit feedback given by search engine users. When supplying a user with a set of candidate web pages, the link that received a click may be considered to be preferred over those that did not receive a click. This feedback information does not directly inform us whether the offered links matched the user need well or not, only that the clicked link was a better match than the other ones that were presented.

A set of pairwise preferences can be encoded as a directed preference graph, where input points serve as vertices, and the edges encode preferences between the vertices. By an edge $e_i = (h, j)$, where $h \neq j$, we encode that x_h is preferred over x_j. Formally, for a data set of m examples, we denote the corresponding output space of preference graphs as

$$\mathcal{Y} = \bigcup_{l \in \mathbb{N}} (\{(h, j) | h \neq j, h, j \in [m]\})^l,$$

and denote a preference graph drawn from the underlying distribution as

$$E = (e_1, \ldots, e_l) \in \mathcal{Y}.$$

When supplied with utility scores, an equivalent representation as a preference graph may be constructed as follows

$$E = \{(h, j) | y_h > y_j, \exists q \in Q, h, j \in q)\}.$$

For computational reasons, it is rarely a good idea to explicitly construct the pairwise preferences when having access to utility scores. Rather, efficient algorithms are based on modeling the preference graph only implicitly.

3.2 Ranking performance

Our aim is to learn a scoring function $f : \mathcal{X} \mapsto \mathbb{R}$, for predicting utility scores. From ranking point of view it is not important to predict correct values, rather we require that when sorting objects according to predicted scores the resulting ordering agrees as well as possible with the true ranking. We assume that ties are broken randomly, when constructing rankings. Following Herbrich et al. (1999); Dekel et al. (2003), we measure this discrepancy using the *pairwise ranking error*. The error measures the probability,

that two randomly drawn objects are incorrectly ranked. Depending on the assumptions we make about the underlying data source, we may recover a variety of settings

First, let us assume a probability distribution $D_{\succ}$ over $\mathcal{X} \times \mathcal{X}$, such that $(x, x') \sim D_{\succ}$, denotes a pairwise preference $x \succ x'$. Then, the expected ranking loss is defined

$$\mathrm{E}_{(x,x')\sim D_{\succ}}[H(f(x') - f(x))], \tag{3.1}$$

where H is the Heaviside step function defined as

$$H(a) = \begin{cases} 1, & \text{if } a > 0 \\ 1/2, & \text{if } a = 0 \\ 0, & \text{if } a < 0 \end{cases} . \tag{3.2}$$

The expected ranking loss for trivial predictors that always predict randomly, or always predict ties, is 0.5.

Let us assume that we are given a training set Z, sampled according to $D_{\succ}$. The training set contains a preference graph E of k pairwise preferences. Following Dekel et al. (2003), we give a general formulation of the ranking error, also known as disagreement error, as

$$l(f(X), E) = \frac{1}{k} \sum_{(i,j)\in E} H(f(x_j) - f(x_i)). \tag{3.3}$$

The properties of $D_{\succ}$ determine, to what extent the underlying relation produces consistent rankings, and hence to what extent it can be modeled using a scoring function. Shortly put, intransitive preference relations cannot be modeled without error using scoring functions, as intransitivity leads to preference cycles, for which no corresponding ranking exists. For example, for the game of rock-paper-scissors, at best we can capture only two of the three rules for the game by producing a ranking for the three strategies. However, in learning to rank, when sampling the data as pairs, we may often assume that the underlying relation is transitive, considering any preference cycles as the product of noise. Further, in settings where the pairwise preferences are generated from utility scores of individual examples, the transitivity of the preference relation is guaranteed. For more in-depth analysis about ranking representability of general preference learning problems, we refer to Pahikkala et al. (2010e).

Next, we consider the scoring setting, where all the inputs are associated with a real-valued utility score. Further, we assume the global ranking setting. Here, a typical assumption is (see e.g. Clémençon et al. (2005)), that there exists a probability distribution D over $\mathcal{X}\times\mathbb{R}$, from where our data is drawn as input-utility pairs. Then, $D_{\succ}$ corresponds to two independent

draws of objects from D. Without loss of generality we can again assume that the first object in a drawn pair has higher utility than the second one, with ties being broken randomly.[2] Now let us consider a training set $(X, \mathbf{y}) = Z$, consisting of an i.i.d. sample from D. The pairwise ranking error is defined as the sum

$$l(f(X), \mathbf{y}) = \frac{1}{N} \sum_{y_i > y_j} H(f(x_j) - f(x_i)), \tag{3.4}$$

As noted for example in Clémençon et al. (2005), the set Z does not constitute an i.i.d. sample from $D_{\succ}$, since there are dependencies between pairs related to same inputs. Nevertheless, theoretical results in the literature suggest that (regularized) risk minimization based on (3.4) should lead to low expected ranking loss (3.1) (Herbrich et al., 2000; Agarwal et al., 2005; Clémençon et al., 2005).

This far we have assumed, that the data is sampled independently either on the level of pairs, or on the level of individual inputs. However, in the commonly encountered query-setting the data is rather sampled on the level of queries. At test time we are offered a query consisting of data points. The goal is to produce a ranking for the query such that matches the true ranking as well as possible, as measured by a multivariate loss function l. Let D_q denote the fixed but unknown distribution over queries. The expected loss is

$$\mathrm{E}_{(X,\mathbf{y})\sim D_q}[l(f(X), \mathbf{y})],$$

where $(X, \mathbf{y})$ contains the inputs and the true utility scores for a randomly sampled query. This formulation has been proposed in literature on learning to rank for document retrieval, where data typically has query structure (Joachims, 2002b; Lan et al., 2009). Following Joachims (2002b), we may use the pairwise ranking error (3.4) as the query-wise loss, and define the empirical risk as the average pairwise ranking error over all the queries in the training set. Also other types of losses have been suggested in the literature, as discussed in Section 3.4.3.

3.3 Bipartite ranking and ordinal regression

The bipartite ranking problem is perhaps the simplest possible ranking setting. Here each input is assigned one of two possible scores. Without loss of generality we may define the set of available labels as $\{-1, 1\}$. The aim is to learn a prediction function that ranks the positive examples higher than the

[2] For technical reasons in settings where it is likely that two randomly drawn examples have exactly the same utility, we may wish to instead define $D_{\succ}$ such that the second drawn object is conditioned on having a different utility score than the first one.

negative ones. The induced preference graph has bipartite structure, meaning that all positive examples are connected with all negative examples, but examples from the same class are not connected. Hence the term bipartite ranking has been used for example in Freund et al. (2003); Agarwal et al. (2005) to describe the setting.

Clearly, any binary classification task can be treated as a bipartite ranking problem. In fact, it is well known that the bipartite ranking problem is equivalent to the task of maximizing the area under the ROC curve (AUC) binary classification performance measure (see e.g. Bamber (1975); Cortes and Mohri (2004); Agarwal et al. (2005)). Historically, AUC has been associated with analysis of radar images, (Hanley and McNeil, 1982) and medical decision making (Swets, 1988). In the recent years the measure has received substantial interest within the machine learning community (Agarwal et al., 2005; Bradley, 1997; Cortes and Mohri, 2004; Fawcett and Flach, 2005; Huang and Ling, 2005; Provost et al., 1998; Waegeman et al., 2008; Vanderlooy and Hüllermeier, 2008).

The AUC criterion corresponds to the probability, that given two randomly drawn inputs, one from the positive and one from the negative class, the classifier is able to distinguish between them correctly. Formally, let the conditional distribution of an input from $\mathcal{X}$, given that it belongs to the positive class be denoted by D_+, and given that it belongs to the negative class by D_-. Agarwal et al. (2005) define the expected ranking accuracy over all possible positive-negative example pairs as

$$\mathrm{E}_{x_+ \sim D_+ x_- \sim D_-}[H(f(x_+) - f(x_-))],$$

and derive generalization bounds which bound this quantity either as a function of AUC measured on the training data, or on independent test data.

AUC may be considered an empirical estimate of the expected ranking accuracy. It can can be computed using the following formula, also called the Wilcoxon-Mann-Whitney statistic:

$$p(f(X), Y) = \frac{1}{|X_+||X_-|} \sum_{x_i \in X_+} \sum_{x_j \in X_-} H(f(x_i) - f(x_j)), \qquad (3.5)$$

where X is a sequence of examples, and $X_+ \subset X$ and $X_- \subset X$ denote the division of the inputs to positive and negative classes according to the class labels in Y, respectively (see e.g. Cortes and Mohri (2004)). From (3.4) we can see that in bipartite ranking the AUC criterion becomes equivalent to the ranking error, since in this case AUC equals one minus ranking error.

The AUC measure has a number of beneficial properties, due to which its use in place of classification error has been recommended in evaluation of binary classification methods. These properties include its invariance to to

class-specific error costs, which are often difficult to determine, its invariance to changes in the prior class distributions, and the fact that trivial predictors such as majority voter or random predictions never lead to high AUC performance regardless of the data distribution (Bradley, 1997). Cortes and Mohri (2004) provided a detailed theoretical comparison of classification error and AUC, and showed that achieving low classification error does not guarantee high AUC.

Ordinal regression can be seen as a generalization of the AUC criterion to ranking with multiple levels of utility. Instead of two possible levels, there now exists several levels of utility, which we may encode as each data point having a label from the set $\{1, \ldots, k\}$, where $k \in \mathbb{N}$ is the number of possible levels. Ordinality means that the categories are ordered, with higher categories being preferred over the lower ones. In contrast to regression there is however no distance metric between the categories. This type of data is often gathered from human judgements (e.g. rate movies from one to five stars or restaurants on scale 1-10). A common assumption is that the number of possible categories is fairly low, since it is easier for humans to provide feedback given only a small number of choices to decide from.

There are several different perspectives to ordinal regression learning. First, one may aim at prediction time to predict the exact category a test example belongs to. This can be modeled using a prediction function f and $k-1$ thresholds $\theta_1 \ldots \theta_{k-1}$. Now the category a new example belongs to may be predicted as

$$c(x) = \begin{cases} 1, & \text{if } f(x) \leq \theta_1 \\ l, & \text{if } \theta_{l-1} < f(x) \leq \theta_l,\ l \in \{2 \ldots k-1\} \\ k, & \text{if } f(x) > \theta_{k-1} \end{cases} \tag{3.6}$$

Many approaches to ordinal regression learning fit this model, including both classical approaches introduced in statistics (see e.g. (McCullagh, 1980)), as well as recent work in machine learning (see e.g. (Frank and Hall, 2001; Crammer and Singer, 2002; Chu and Keerthi, 2007).

However, from ranking point of view requiring the fitting of thresholds is unnecessary. When making predictions it does not matter whether correct categories are assigned to test examples, only that the ranking acquired by sorting test examples by their predicted scores matches the true ranking well. Starting from the work of Herbrich et al. (1999), much of the work on learning to rank from ordinal data has concentrated on optimizing losses which guarantee good ranking performance, without the need to recover the actual categories at prediction time. The approach of Herbrich et al. (1999) was to consider the pairwise preferences induced by the ordinal categories, where each data point is preferred over those belonging to the lower categories. This naturally leads to algorithms that minimize approximations of the ranking error (3.4) in order to achieve low expected ranking loss (3.1).

This is the approach we adopt in our work on learning to rank, when learning from ordinal data. A slightly different approach was introduced in Waegeman et al. (2008) where AUC was generalized for ordinal regression into volume under the surface, which estimates the expected value for the probability of correctly ordering a k-tuple given a randomly drawn data point from each category.

The setting for learning to rank from scored data as defined in Section 3.1 can be considered more general than the ordinal regression setting considered here. First, in ordinal regression one typically assumes a global ranking over all the data points, meaning that methods introduced for ordinal regression often cannot deal with query structure in the data. Second, the assumption that the data comes from only a few ordered categories may not in general hold. Methods built upon this assumption may be unsuitable for the case where arbitrary real-valued utility scores are provided. While methods that optimize pairwise criterions typically can generalize to this setting, they may suffer from efficiency problems when the number of allowed categories is no longer restricted (see Section 4.2).

3.4 Approaches to learning to rank

A large number of approaches have been proposed for learning to rank in the last decade or so. Here we provide an overview of this body of related work. Following Liu (2009) we divide the ranking methods into three categories: the pointwise, pairwise and the listwise methods.

3.4.1 Pointwise methods

In the pointwise approach, the ranking problem is modeled by aiming at correctly predicting the utility score or category of the data points. The criterion typically leads to minimizing univariate loss functions, which for each data point calculate the discrepancy between the true label, and the predicted label. Standard classification or regression methods may be applied for learning to predict the labels. Alternatively, ordinal regression methods that take better into account the special structure of the output space have been proposed. A justification for using the pointwise methods is that predicting perfectly the correct utility scores allows one to recover optimal rankings.

As discussed in section 3.3, the bipartite ranking problem can be solved using any binary classification algorithm. For example, some of the work on document retrieval, casts the ranking task simply as classifying "relevant" and "non-relevant" query-document pairs (see e.g. Cooper et al. (1992); Nallapati (2004)). Also, in most applications where the AUC is used to measure performance, standard classification methods are applied

rather than ranking methods. More generally, the ordinal regression problem can be modeled as a multi class learning problem (Li et al., 2008). The aforementioned approaches have a number of drawbacks when applied to ranking. For bipartite ranking, both theoretical and empirical results have shown that optimizing classification error rate may not lead to good ranking performance (see e.g. Cortes and Mohri (2004); Pahikkala et al. (2008b) Paper I). Using multi-class classification ignores the ordinal structure over the classes completely (it is less wrong to predict 4 stars for a 5 star movie than to predict 2 stars). Further, the approach breaks down if the range of utility scores is not restricted to only few categories.

Frank and Hall (2001) introduced a cumulative decomposition approach, where for each category a classifier is trained to predict the probability that the true category is higher than the given category. Crammer and Singer (2002) proposed a perceptron based PRank algorithm for ordinal regression learning in an online setting. Here a predictor such as described in equation (3.6) is used, and both the current linear model and the thresholds are adjusted each time a new data point is misclassified. We are aware of two kernel methods proposed for ordinal regression learning[3]. First, Shashua and Levin (2003) introduced an SVM formulation for ordinal regression, where the selection of suitable thresholds is integrated into the standard SVM optimization problem. The work was further improved in Chu and Keerthi (2005, 2007). Second, Chu and Ghahramani (2005b) introduced a similar approach based on Gaussian processes for ordinal regression learning. A common limitation of all these algorithms is that they are not applicable if there is too large, or unrestricted number of ordinal categories available.

Finally, the ranking problem can be solved by pointwise regression of utility scores. Here methods such as (regularized) least-squares may be used for learning (see e.g. Fuhr (1989); Tsivtsivadze et al. (2005); Cossock and Zhang (2006)). In regression the ordering information is taken into account, and the approach is applicable also when the range of utility scores is unrestricted. The criterion enforced when using regression is however too strict; from ranking point of view it does not matter what the predicted scores are, as long as the ordering they produce is good.

A problem all the methods considered here share is that they can be seriously lead astray by imbalanced data. Predicting the majority class or mean of the training labels may produce low classification or regression error, but does not help in ranking. Finally, the methods ignore query structure and are not applicable when only pairwise comparisons are available. All this being said, many of the pointwise approaches are simple to efficiently implement, and can often reach as good ranking performance as those optimizing

[3]We do not include the work of Herbrich et al. (1999, 2000) here because, even though it was motivated by the problem of ordinal regression, the end result is a general pairwise ranking method.

ranking based criteria more directly as discussed in more detail at the end of this chapter.

3.4.2 Pairwise Methods

In the pairwise approach, the aim is to learn to correctly predict relative order between any two data points. A method that is able to correctly order data points can be used to construct accurate rankings. For bipartite ranking this criterion leads to AUC optimization (3.5), and in the more general case to pairwise ranking error minimization (3.3, 3.4). The approach is inspired by the notion of pairwise preferences, and in the preference learning literature the setting is sometimes known as *object ranking* (Fürnkranz and Hüllermeier, 2010).

The criterion is more closely related to final goals of learning to rank than the pointwise criteria. On one hand correct prediction of categories or scores is not enforced, what matters is how well the predictor can decide which of two data points is more preferred. On the other hand pathological solutions such as a majority voter or a mean predictor are not encouraged, since such models achieve no better pairwise performance than random. Finally, the approach allows learning directly from pairwise comparisons, and queries can be modeled by considering only preferences between data points belonging to the same query.

Tesauro (1989) proposed learning a unary scoring function from pairwise comparisons using a neural network architecture. More recently, the pairwise approach has been popularized by the introduction of a pairwise formulation of SVM, popularly known as the RankSVM (Herbrich et al., 1999, 2000). Originally, RankSVM was introduced for learning from ordinal regression data, a generalization to learning directly from pairwise comparisons was introduced in Joachims (2002a), a modification better suited to query structured data in Cao et al. (2006) and an efficient training method for the linear case in Joachims (2006). Many other classification algorithms have similarly been adapted to learning ranking functions by optimizing a pairwise criterion. These include the RankBoost (Freund et al., 2003), and RankNet (Burges et al., 2005) algorithms, as well as the RankRLS algorithm, originally introduced in Pahikkala et al. (2007), which is the subject of Paper I. The magnitude preserving ranking algorithm MPrank (Cortes et al., 2007b,a) also belongs to this category.

An alternative direction was considered in Cohen et al. (1998), where the ranking problem was modeled as a classification problem on pairs of examples. Here the aim is to learn a binary prediction function $g : \mathcal{X} \times \mathcal{X} \mapsto \{-1, 1\}$, which predicts, whether the first example is preferred over the second one, or vice versa. An advantage of this approach is that any binary classification algorithm may be used here for learning, by using joint fea-

ture representations of pairs of data points both at training and prediction time. However the approach has not been popular in learning to rank for two reasons. First, producing the predicted ranking from the pairwise comparisons of test examples is a non-trivial task. The learned model may be non-transitive, producing predictions which are not consistent with any ranking. For a proposed solution on how to construct the final ranking efficiently from pairwise classifications, see Ailon and Mohri (2008). Second, the quadratic growth in the size of the training set leads to computationally inefficient algorithms. The pairwise classification approach has shown more promise in recent work on preference learning tasks where the underlying relation has a complex structure that cannot be reduced to a linear ordering representable by a simple unary scoring function, such as cyclic preference relations (Pahikkala et al., 2010e) and conditional ranking models (Pahikkala et al., 2010d).

AUC-maximization has been studied as a topic of its own in several studies. Variants of most learning methods have been introduced for the task, including those based on boosting (Freund et al., 2003), decision trees (Ferri et al., 2002), logistic regression (Herschtal and Raskutti, 2004), RLS (Pahikkala et al., 2008b), rule sets (Fawcett, 2001), and SVMs (Rakotomamonjy, 2004; Brefeld and Scheffer, 2005; Joachims, 2005). Methods specifically designed for AUC optimization may not be suitable for other ranking tasks, though some of these methods are known to be special cases of general pairwise ranking algorithms.

A common problem for all the pairwise methods is that for straightforward implementations of the methods, the training complexity grows with respect to the number of pairwise preferences, rather than the number of training examples. For global rankings the number of preferences typically grows quadratically with training set size, while for query structured data the growth is linear in the number of queries, and quadratic in query size. These complexities may be affordable for small training sets, or small query sizes, but can become problematic on large data sets. One approach proposed for solving this problem has been approximation, where only a subset of all pairs is used (see e.g. Rakotomamonjy (2004); Brefeld and Scheffer (2005); Sculley (2009)). Other line of approach has been to develop computational shortcuts which allow the optimization of pairwise losses without explicitly iterating over the pairs (see e.g. Freund et al. (2003); Joachims (2006); Cortes et al. (2007b)). The algorithms introduced in Papers I and II are based on the latter approach (see also Chapter 4).

3.4.3 Listwise Methods

The listwise approach to learning to rank has been motivated by the multivariate performance measures used in information retrieval, such as Mean

Average Precision (MAP) (Yue et al., 2007), Normalized Discounted Cumulative Gain (NDCG) (Järvelin and Kekäläinen, 2000), and Expected Reciprocal Rank (ERR) (Chapelle et al., 2009). These measures give much greater importance on whether the first entries in the ranking are correct, than to the correctness of the entries low in the produced ranking. This choice is motivated by the fact that in many ranking applications, such as web search, the user is likely to examine only the highest ranked objects. The pointwise and pairwise approaches may not be optimal here, since they do not give any special importance on the highest ranked entries.

The listwise, or multivariate learning algorithms aim at optimizing these measures more directly. This is made challenging by the fact that the previously mentioned measures do not decompose into pointwise or pairwise evaluations. Instead, the individual atoms considered are queries, and the losses measure how well a predicted list of relevance judgements or a permutation of the query indices matches the correct ranking. Also, when viewed as functions of predicted scores, the aforementioned measures are not smooth or differentiable. A small change in a predicted score often has no effect on the produced ranking. Thus advanced optimization strategies which are able to work with full queries are needed in listwise learning. Introduced approaches include structural SVMs (Joachims, 2005; Tsochantaridis et al., 2005; Yue et al., 2007; Chapelle et al., 2007), as well as several several other approaches (Burges et al., 2007; Cao et al., 2007; Qin et al., 2008; Xia et al., 2008). The majority of these methods are based on linear models.

3.5 Conclusions

A natural question that arises considering all the presented approaches is, which one should adapt in practice. In case the training data is gathered as pairwise comparisons it is quite clear that only the pairwise methods are applicable, but for scored data one can use any of the three approaches. Previous results in the literature suggest that considering pairwise preferences should lead to higher ranking performance than using pointwise methods (see e.g. Herbrich et al. (1999); Freund et al. (2003); Burges et al. (2005); Joachims (2005)). Further, articles introducing listwise methods have consistently reported that the listwise approach allows better optimization for information retrieval specific measures than the other approaches (Yue et al., 2007; Chapelle et al., 2007; Cao et al., 2007; Qin et al., 2008; Xia et al., 2008).

As a contrary result, Chapelle and Keerthi (2010) recently demonstrated that the RankSVM method outperforms in terms of the NDCG measure a number of considered listwise based baseline methods on the widely used LETOR (Qin et al., 2010) benchmark dataset. Further, in the recent Yahoo! 2010 Learning to Rank Challenge, it was reported that most of the top

performing systems were actually based on pointwise regression of utility values (Chapelle and Chang, 2011). The possible explanations offered for this phenomenon were that the regression based approaches allowed non-linear modeling, unlike most methods optimizing ranking based losses, as well as the possibility of some previously reported improvements in the literature being the product of random chance.

The focus of this thesis is on regularized kernel methods, as this allows the use of structured data, learning non-linear models, and offers principled ways to deal with both the underfitting and overfitting phenomena. Further, we require that the considered methods are as general as possible, rather than restricted to certain settings such as only AUC maximization, though we will also study such important special cases in detail. The considered methods should achieve higher ranking performance than the pointwise approach, and be computationally efficient. These constraints motivate both the development of the RankRLS method, as well as improvements to the computational efficiency of RankSVM, both discussed in Chapter 4.

Chapter 4

Novel ranking algorithms

In this section we introduce the RankRLS algorithm developed as part of this thesis work, as well as an improved training method for the RankSVM method. Both RankRLS and RankSVM combine kernel based learning, and learning to rank by minimizing a convex approximation of the number of pairwise mistakes made by the learned model. We assume the setting described in Section 3.1, where training data containing either utility scores or pairwise preferences is provided.

4.1 RankRLS

The RankRLS method combines the regularized risk minimization problem (2.2) with a least-squares based approximation of the pairwise disagreement error (3.3). The method was originally formulated in Pahikkala et al. (2007) for the case of scored data. Tsivtsivadze et al. (2008) studied the reduced set approximation of RankRLS, Pahikkala et al. (2008b) considered the specific case of AUC-optimization, and Pahikkala et al. (2008a) presented and efficient leave-pair-out cross-validation algorithm for the method. The following sections are based on Paper I, that collects together this previous work, and further extends RankRLS to learning from pairwise comparisons, introduces new efficient algorithms, and includes extended experimental evaluation.

In addition to pairwise preferences, we may in some settings have access to *preference magnitudes*, that denote to which degree an object is preferred over another. For two input-utility pairs (x, y) and (x', y') such that $x \succ x'$, the preference magnitude can be defined as $y - y'$. In theory one can also define magnitudes for general preference relations not based on utility scores. If such information is not available and magnitudes are required, we may assume that each preference has a magnitude 1. In the following, we use E_M to denote a set of pairwise preferences augmented with preference magnitudes, meaning that each $e_i = (h, j, w_i) \in E_M$ contains a magnitude

w_i encoding the degree, to which x_h is preferred over x_j. The corresponding preference graph is thus a directed weighted graph.

The RankRLS is an example of a *magnitude preserving ranking algorithm*, a term first introduced in Cortes et al. (2007b), meaning that in addition to directions of preferences it also aims to enforce their magnitudes using a two-sided loss function. Analogously to the use of RLS regression for classification, one might expect this criterion to be unnecessarily strict in case the goal is to achieve low ranking error. Experimental results presented in Paper I however indicate, that also in this case the approach leads to predictive performance that is comparable to that achieved with one-sided pairwise ranking losses.

4.1.1 Problem formulation

We define the magnitude preserving pairwise ranking loss as

$$l(f(X), E_M) = \sum_{(h,j,w_i)\in E} (w_i - f(x_h) + f(x_j))^2. \tag{4.1}$$

This choice is most natural when using scored data since the preference magnitudes can be recovered from differences of utility scores. For scored data, (4.1) reduces to[1]

$$l(f(X), \mathbf{y}, Q) = \sum_{q\in Q} \sum_{i,j\in q} (y_i - y_j - f(x_i) + f(x_j))^2. \tag{4.2}$$

The loss function (4.2) is how the RankRLS method was originally formulated in Pahikkala et al. (2007). If the data has no query structure, but rather a single global ranking, then the variant of RankRLS that is based on (4.2) also becomes equivalent to the MPrank algorithm proposed simultaneously and independently of us in Cortes et al. (2007b,a).

In cases where magnitude information about preferences is not available, the magnitude preserving ranking loss reduces to

$$l(f(X), E) = \sum_{(h,j)\in E} (1 - f(x_h) + f(x_j))^2. \tag{4.3}$$

One may further wish to normalize the losses by dividing them by the number of preferences. Alternatively, for query-structured data, the losses incurred in each query may be normalized by the size of the query, as such an approach has been shown to be sometimes beneficial for the related RankSVM method (Cao et al., 2006). In addition to the pairwise

[1]For technical reasons also pairs having the same utility score, such that belong to the same query, are included in the loss. Empirical results in Paper I suggest that this is not detrimental to predictive performance.

squared losses presented here, Paper I also introduces an additional variant of the RankRLS loss. We do not however include it in the following considerations, as it does not improve the applicability of RankRLS beyond what the presented losses allow.

4.1.2 Solving the optimization problem

Inserting the magnitude preserving pairwise ranking loss (4.1) to the regularized risk functional (2.2) we derive the objective function for the RankRLS

$$\sum_{(h,j,w_i)\in E} (w_i - f(x_h) + f(x_j))^2 + \lambda \|f\|^2. \tag{4.4}$$

This can be equivalently re-written in matrix notation either in the primal, or in the dual form.

Let $\mathbf{M} \in \mathbb{R}^{m \times l}$ be a matrix whose rows and columns are indexed by the vertices and edges of the preference graph for the training set, and its entries are given by

$$\mathbf{M}_{h,i} = \begin{cases} w_i & \text{if } e_i = (h, j, w_i), \text{ for some } j \neq h \\ -w_i & \text{if } e_i = (j, h, w_i), \text{ for some } j \neq h \\ 0 & \text{otherwise} \end{cases}. \tag{4.5}$$

Each w_i and $-w_i$ can be replaced by 1 and -1 correspondingly, in case preference magnitudes are not available. In graph theory, this matrix is sometimes called the oriented incidence matrix of a graph (see e.g. Brualdi and Ryser (1991)).

Further, let us write $\mathbf{n} = (w_1, \ldots, w_l)^{\mathrm{T}}$. Then, we can write in matrix form the RankRLS risk (4.4) as

$$J(\mathbf{a}) = (\mathbf{n} - \mathbf{M}^{\mathrm{T}}\mathbf{K}\mathbf{a})^{\mathrm{T}}(\mathbf{n} - \mathbf{M}^{\mathrm{T}}\mathbf{K}\mathbf{a}) + \lambda \mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{a}. \tag{4.6}$$

It can be seen that dual RankRLS problem is analogous to the basic dual RLS formulation (2.14), but the regression is done on the level of preference graph edges. Analogously to RLS, we can recover a close form solution. The gradient of (4.6) is

$$\nabla J(\mathbf{a}) = 2\mathbf{K}(-\mathbf{M}\mathbf{n} + (\mathbf{M}\mathbf{M}^{\mathrm{T}}\mathbf{K} + \lambda \mathbf{I})\mathbf{a}),$$

and by setting the gradient to zero, we can find a globally optimal solution by solving the linear system

$$(\mathbf{M}\mathbf{M}^{\mathrm{T}}\mathbf{K} + \lambda \mathbf{I})\mathbf{a} = \mathbf{M}\mathbf{n}. \tag{4.7}$$

The primal form of the RankRLS risk (4.4) is

$$J(\mathbf{w}) = (\mathbf{M}^{\mathrm{T}}\Phi(X)^{\mathrm{T}}\mathbf{w} - \mathbf{n})^{\mathrm{T}}(\mathbf{M}^{\mathrm{T}}\Phi(X)^{\mathrm{T}}\mathbf{w} - \mathbf{n}) + \lambda \mathbf{w}^{\mathrm{T}}\mathbf{w}. \tag{4.8}$$

The gradient of (4.8) is

$$\nabla J(\mathbf{w}) = -2\Phi(X)\mathbf{M}\mathbf{n} + 2(\Phi(X)\mathbf{M}\mathbf{M}^{\mathrm{T}}\Phi(X)^{\mathrm{T}} + \lambda\mathbf{I})\mathbf{w},$$

and setting it to 0 results in the system of linear equations

$$(\Phi(X)\mathbf{M}\mathbf{M}^{\mathrm{T}}\Phi(X)^{\mathrm{T}} + \lambda\mathbf{I})\mathbf{w} = \Phi(X)\mathbf{M}\mathbf{n}. \tag{4.9}$$

We recall, that by m we refer to the number of training inputs, by l the number of pairwise preferences in the training set, and by n the dimensionality of the feature space. The complexity of solving the dual form (4.7) is $O(m^3+l)$, where the $O(m^3)$ term is due to the cost of solving a $m \times m$ linear system, using for example matrix inversion. Since in most applications the number of edges grows at most quadratically with the number of training examples, we may simplify the overall complexity to $O(m^3)$.

The computational complexity of solving the primal form (4.9) is $O(n^3 + \min(n^2m + m^2n + l, n^2l))$. If $n << m$ this can be much more efficient than using the dual form. Further, for the scoring setting and the magnitude preserving loss it can be shown, that using a low-rank decomposition of the graph Laplacian matrix $\mathbf{M}\mathbf{M}^{\mathrm{T}}$, the primal solution can be recovered in $O(n^3+n^2m)$ time, eliminating the dependency on the number of preferences.

Thus using basic dense linear algebra techniques based on matrix inversion or matrix factorization, RankRLS can be trained in a time that is either cubic in the number of training examples, or cubic in the dimensionality of feature space. In practice this means that on modern computers RankRLS can be trained as long as either the number of training examples, or the dimensionality of the feature space does not exceed a few thousands.

When using kernels, the reduced set approximation (see Section 2.5) can be used to scale RankRLS training beyond a few thousand training examples. This approach is described in detail in Section 3.5 of Paper I, and it can be seen as a special case of the Nyström approximation scheme studied in Paper III. Additional experimental results demonstrating that this approach can lead to increased ranking performance are presented in Tsivtsivadze et al. (2008).

Alternatively, in many central ranking applications, such as web search, the data to be ranked consists of natural language documents. In typical feature representations the dimensionality of the feature space is huge, as it may be related to the number of distinct words in a vocabulary, or to some power of this number. However, the data is sparse, meaning that the data matrix is filled mostly with zeroes. Using the linear kernel, it is possible to make use of this sparsity, avoiding explicitly constructing dense $m \times m$ or $n \times n$ matrices. Using the conjugate gradient method, the RankRLS optimization can rather be formalized in terms of sparse matrix - vector products. The basic technique is described in Paper I, more detailed analysis and further experimental results are presented in Airola et al. (2010). Let

$\overline{n}$ be the average number of non-zero features per example, and t the number of iterations that conjugate gradient optimization needs to converge. Then linear RankRLS can be trained from scored data in $O(tm\overline{n})$ time, and from pairwise preferences in $O(tm\overline{n} + tl)$. As discussed in Airola et al. (2010) bounding t is not straightforward, but in practice it was observed that even on very large datasets good solutions could be achieved within tens of iterations. In practice this allows linear RankRLS training to scale to tens to hundreds of thousands of training examples and features on sparse data in a matter of minutes.

4.1.3 Computational shortcuts

For scored data and the magnitude preserving ranking loss, Paper I introduces a number of computational shortcuts. The shortcuts are analogous to those known for the basic RLS algorithm (see Section 2.7).

First, it can be shown that after computing the eigendecomposition of the matrix $\mathbf{M}\mathbf{M}^{\mathrm{T}}\mathbf{K}$, in $O(m^3)$ time, solutions for different regularization parameter values λ can subsequently be computed in $O(m^2)$ time. This is highly advantageous, since one rarely knows in advance the suitable value for λ, rather it is typically chosen by grid searching. Similar shortcut can be used for re-training primal RankRLS for different regularization parameter values in $O(n^2)$ time, after performing $O(n^3 + n^2m)$ time operations.

Second, using the previously discussed eigendecomposition operations, we can train RankRLS efficiently for multiple ranking tasks. Assuming that instead of a single vector of output scores, we have a $m \times v$ matrix representing v distinct ranking tasks, dual RankRLS can be trained in $O(m^2v)$ time for all the tasks, assuming previously computed $O(m^3)$ cost decomposition. Similarly, primal RankRLS can be trained in $O(n^2v+mnv)$ time for v tasks, assuming previously computed $O(n^3 + n^2m)$ cost operations.

Third, based on low-rank matrix update operations, one can develop computationally efficient cross-validation algorithms for RankRLS. These methods in effect allow a trained RankRLS model with a minimal number of operations to "unlearn" the effects of a holdout set of examples, that is left out of the training set. Two such fast algorithms have been derived. First, we consider the the case, where there is a single global ranking defined over the objects. Here leave-pair-out cross-validation can be seen as a generalization of the standard leave-one-out method, for pairwise loss functions. The method was proposed by (Cortes et al., 2007a), who derived an approximate method for computing this estimate for the MPrank algorithm in $O(m^2)$ time. In Paper I, we improve this result by presenting an $O(m^2)$ method for computing the non-approximative leave-pair-out predictions for RankRLS. Further, we consider the problem of cross-validation for query-structured data, and present a $O(m^2 + m|q|^2)$ time method for

leave-query-out cross-validation, where $|q|$ is the average query size.

Recently we have introduced further computational shortcuts for RankRLS, whose detailed treatment however falls outside the scope of this thesis. In Pahikkala et al. (2010a) we derived Greedy RankRLS, a greedy forward selection algorithm for doing feature selection with RankRLS on query-structured data. The method starts from the empty feature set, and the iteratively keeps selecting the feature whose addition leads to largest decrease in leave-query-out cross-validation error. Let k be the number of selected features. Then the method can be trained in $O(kmn)$ time, by making use of leave-query-out cross-validation shortcuts, as well as additional shortcuts for fast addition of new features to the model. Further, in Pahikkala et al. (2010d) we considered the problem of conditional ranking on relational data, where the vertices of relational graphs are ranked according to how well they match any vertex on which the ranking is conditioned on. For example, when playing a game, other players can be ranked according to how likely they are to win against a given player. We derived an efficient to compute closed form solution for RankRLS for learning such pairwise ranking relations using pairwise Kronecker kernels.

4.2 RankSVM

The ranking support vector machine (RankSVM) method combines the regularized risk minimization problem (2.2) with a hinge-loss based approximation of the pairwise disagreement error (3.3). The method was proposed originally in Herbrich et al. (1999) for solving ordinal regression problems, and later adapted to learning from query structured data and pairwise preferences in Joachims (2002a).

4.2.1 Problem formulation

The pairwise hinge loss is defined as

$$l(f(X), E) = \sum_{(i,j)\in E} \max(1 - f(x_i) + f(x_j), 0).$$

The loss can be seen as an extension of the basic hinge loss (2.15) to pairwise comparisons, the loss is also very similar to the pairwise least-squares loss (4.1). For scored data, the pairwise hinge loss can be written as

$$l(f(X), \mathbf{y}, Q) = \sum_{q\in Q} \sum_{i,j\in q, y_i > y_j} \max(1 - f(x_i) + f(x_j), 0).$$

Again, we can normalize the losses by the number of pairwise preferences, or by the query size as suggested in Cao et al. (2006).

4.2.2 Solving the optimization problem

Herbrich et al. (1999); Joachims (2002a) note the close connection of the RankSVM problem to ordinary SVM classification. Let us consider a binary classification problem, where we encode each preference $x_i \succ x_j$ as a positive training example, using $\Phi(x_i) - \Phi(x_j)$, or the dual representation of this difference, as its feature vector. Likewise, we may assume the existence of a corresponding negative training example, whose feature vector is $\Phi(x_j) - \Phi(x_i)$. When computing the hinge loss (2.15) for this classification problem, for each preference $x_i \succ x_j$ we recover a term $\max(1 - 1 \cdot \langle \mathbf{w}, \Phi(x_i) - \Phi(x_j) \rangle), 0)$ for the positive, and the same term $\max(1 - (-1) \cdot \langle \mathbf{w}, \Phi(x_j) - \Phi(x_i) \rangle), 0) = \max(1 - \langle \mathbf{w}, \Phi(x_i) - \Phi(x_j) \rangle), 0)$ for the corresponding negative example. Due to the linearity of the inner product, we can decompose the prediction as $\langle \mathbf{w}, \Phi(x_i) - \Phi(x_j) \rangle = \langle \mathbf{w}, \Phi(x_i) \rangle - \langle \mathbf{w}, \Phi(x_j) \rangle$, allowing us to recover the pairwise hinge loss.

One practical implication of this result is that any standard SVM solver can be used to solve also the RankSVM problem (analogously, the same connection exists between the RLS and RankRLS methods). This approach was originally adapted in Herbrich et al. (1999). Further, the popular kernel RankSVM solver included in the SVM$^{\text{light}}$ software package (Joachims, 1998), uses a standard SVM solver trained on example pairs for training the RankSVM (Joachims, 2002a). The downside of this approach is that the computational complexity of these solvers becomes dependent not on the number of examples, but on the number of pairwise preferences. Since the number of pairwise preferences often grows quadratically with respect to training set size, adapting modern dual SVM solvers to RankSVM training can lead to $O(m^4)$ or worse scaling.

For linear RankSVM training, more efficient training algorithms have been proposed. A method suitable for scored data was introduced in Joachims (2006), our treatment of the subject is based on the formulation in Teo et al. (2010), where the approach was shown to be equivalent to subgradient optimization using a bundle method. Let us consider RankSVM training on scored data, where only a single global ranking exists. The setting can be easily generalized to the query structured setting, since the loss and subgradient can there be expressed as a sum of loss and subgradient computations for individual queries.

Joachims (2006) shows that linear RankSVM training can be done using bundle optimization. On each iteration of the method, the loss corresponding to the current solution, as well as its subgradient needs to be computed. According to analysis in Joachims (2006) the method needs $O(\frac{1}{\lambda\epsilon^2})$ steps to converge to ϵ-accurate solution, Smola et al. (2007) improves this bound to $O(\frac{1}{\lambda\epsilon})$. Thus assuming that the pairwise hinge loss and its subgradient can be computed efficiently, one can create an efficient linear RankSVM solver.

Let us define

$$c_i = |\{j : (y_i < y_j) \wedge (\mathbf{w}^{\mathrm{T}}\Phi(x_i) > \mathbf{w}^{\mathrm{T}}\Phi(x_j) - 1) \wedge (1 \leq j \leq m)\}| \quad (4.10)$$

and

$$d_i = |\{j : (y_i > y_j) \wedge (\mathbf{w}^{\mathrm{T}}\Phi(x_i) < \mathbf{w}^{\mathrm{T}}\Phi(x_j) + 1) \wedge (1 \leq j \leq m)\}|. \quad (4.11)$$

Using the frequencies (4.10) and (4.11), we recover an alternative formulation for the empirical risk.

The average pairwise hinge loss can be equivalently expressed as

$$\sum_{i=1}^{m}((c_i - d_i)\mathbf{w}^{\mathrm{T}}\Phi(x_i) + c_i). \quad (4.12)$$

Similarly, computation of (4.10) and (4.11) allows the computation of a subgradient of the empirical risk. A subgradient of the pairwise hinge loss can be expressed as

$$\sum_{i=1}^{m}(c_i - d_i)\Phi(x_i). \quad (4.13)$$

Inner product evaluations, scalar-vector multiplications and vector summations are needed to calculate (4.12) and (4.13). These take each $O(\overline{n})$ time, where $\overline{n}$ is the average number of nonzero elements in a feature vector. Provided that we know the values of c_i, d_i and N, both the loss and subgradient can thus be evaluated in $O(m\overline{n})$ time.

Joachims (2006) describes a way to calculate efficiently these frequencies, and subsequently the loss and subgradient. However, the work assumes that the range of possible utility score values is restricted to r different values, with r being quite small. The algorithm requires $O(r)$ passes through the training set, contributing a $O(rm)$ term to the overall complexity, which is $O(m\overline{n} + m\log(m) + rm)$. If the number of allowed scores is not restricted, at worst case $r = m$ with the resulting complexity $O(m\overline{n} + m^2)$, meaning quadratic behavior with respect to the training set size. Chapelle and Keerthi (2010) proposes for scored data a similar algorithm, that is however based on quasi-Newton optimization rather than the bundle method, but the approach has also at worst case quadratic iteration cost. On very large data sets, this quadratic scalability can still limit the scalability of linear RankSVM training on scored data.

Paper II presents a technique for removing this dependence on r from the complexity. The method uses balanced binary search trees to allow the iterative computation of both c_i and d_i in $O(m\log(m))$ time, allowing $O(m\overline{n} + m\log(m))$ worst case behavior for linear RankSVM training.

On large enough data sets this can make a substantial difference in training times, reducing days of training time to minutes, as demonstrated also experimentally in Paper II.

For the general case, where the data is supplied directly as pairwise preferences, Chapelle and Keerthi (2010) presents an $O(t\overline{n}m+tl)$ time aglorithm for linear RankSVM training, where the number of iterations t is assumed to be often a small constant. The method is analogous to the conjugate gradient training method for RankRLS. It is also based on the use of the oriented incidence matrix (4.5), in order to avoid explicit enumeration of the feature representations of the pairs, and has similar time complexity.

Efficient kernel RankSVM training can be achieved using the empirical kernel map corresponding to the Nyström approximation, explored especially in Paper III, in order to convert the dual RankSVM problem to the primal problem. This idea was introduced already in Section 4 in Paper I. The use of empirical kernel map for RankSVM training has subsequently been independently considered by Chapelle and Keerthi (2010). Briefly put, we can create an empirical kernel map of the data corresponding to reduced set approximation in $O(mr^2)$ time, where r is the number of basis vectors. After this, a linear RankSVM can be trained on the data in the scoring setting in $O(mr + m \log m)$ time, resulting in an overall $O(mr^2 + m \log m)$ complexity. For the general pairwise setting we recover $O(mr^2 + trm + tl)$ complexity, adapting the training method of Chapelle and Keerthi (2010). Alternatively, the bundle method may be adapted to directly training the reduced set approximation in the dual, as was done for example in Joachims and Yu (2009) for regular SVM classification.

4.3 RankRLS vs. RankSVM

The RankRLS and RankSVM methods are highly related, as both are regularized kernel methods that aim to solve the supervised ranking problem by minimizing convex pairwise loss functions. Previously, RankSVM has been shown to work well in a wide variety of applications. When considering the two methods, a natural question that arises is, in which setting should one use one or the other of the two methods. The computational properties of the methods are compared in Section 4 in Paper I, and an experimental comparison is presented in Section 5 in Paper I on four datasets from different application domains.

Considering the ranking performance, the main conclusion was, that there is very little difference in the performance of RankSVM and RankRLS. Both methods were found to outperform RLS on two of the considered problems, while on the other two problems basic RLS performed as well as the pairwise methods. The empirical results suggest that both RankSVM and

RankRLS seem to have as good as or better ranking performance as basic regression. We consider the main difference between the applicability of RankRLS and RankSVM to be computational efficiency.

For linear learning, RankRLS can be trained in a time that is linear with respect to training set size using either dense matrix algebra techniques for low-dimensional data, or the conjugate gradient method for sparse but high-dimensional data. The $O(m \log(m))$ training time needed for RankSVM training indicates slightly worse scaling, though in practice if one of the methods can be trained on some problem, then the other one is also applicable. When training a single classifier using kernels, the $O(m^3)$ time RankRLS training algorithms are more efficient, than the straightforward adaptations of kernel based SVM-solvers proposed in (Herbrich et al., 1999; Joachims, 2002a). As a point of comparison, in a runtime experiment presented in Table 1 in Paper I, it can be seen that already at 2500 training examples, the SVM$^{\text{light}}$ RankSVM solver needed over 5.5 hours for training, whereas RankRLS could be trained in 83 seconds. As discussed in Section 4.2.2, one can however apply the efficient primal RankSVM solvers, such as the one introduced in Paper II, to solve also the dual problem.

The main computational advantage of RankRLS over RankSVM results from the computational shortcuts made possible by its closed form solution. In settings where procedures such as fast selection of regularization parameter, cross-validation, multi-output learning or feature selection need to be performed, using RankRLS can be much faster than using RankSVM. This is because such procedures can be computed for the same cost as training RankRLS once, whereas RankSVM needs to be re-trained multiple times. Further, the training complexity of RankSVM, for methods both those based on bundle optimization (Joachims, 2006) or on quasi-Newton optimization (Chapelle and Keerthi, 2010), has inverse dependency on the regularization parameter λ. For small values of λ the methods may need substantial number of iterations in order to converge. While the same property holds for the conjugate gradient RankRLS, the training times for dense matrix algebra based RankRLS training methods are not affected by the choice of λ.

Chapter 5

Performance estimation

Performance estimation is a necessary task in a wide variety of settings, when applying machine learning. For a given trained predictor, one typically needs to estimate its expected performance on future data in order to decide, whether it is of sufficient quality to be used in an application. When the subject of study is the properties of a given learning algorithm, one may need to evaluate its expected performance on a given domain, or compare it to other methods proposed for solving the same task. Finally, in model selection, one needs to estimate which choice of learner parameters leads to the best model. For kernel methods, typical such parameters are a regularization parameter, as well as kernel parameters, such as the width of the Gaussian kernel (2.9), or the degree of a polynomial kernel (2.10). For a detailed taxonomy of different performance estimation tasks we refer to Dietterich (1998).

5.1 Performance measures

To estimate performance one first needs to choose a performance measure. Analogously to loss functions, we can define a performance measure as a function

$$p : \bigcup_{m \in \mathbb{N}} \mathbb{R}^m \times \mathcal{Y} \mapsto \mathbb{R}$$

that measures how well the predicted labels and true labels for a set of data points match. Some loss functions, such as the squared error (2.11), are standardly used also for performance estimation. More often however, performance measures are not suitable as such for efficient optimization. Following Joachims (2005) we divide performance measures into two categories: *univariate*, and *multivariate*.

Univariate performance measures can be decomposed into a sum of evaluations over individual data points, where a prediction for a single data point

is compared to its true label. A typical example of a univariate performance measure is the classification error rate, defined as

$$p(f(X), \mathbf{y}) = \frac{1}{m} \sum_{i=1}^{m} [y_i f(x_i) \leq 0],$$

where we assume that $y_i \in \{-1, 1\}$, and $[P]$ follows the Iverson bracket notation, denoting the number 1 if the statement P is true, and number 0 otherwise. Another typical example is the squared error (2.11). What is notable for both is that they can be represented as a sum, consisting of comparisons of predicted and true labels for individual data points.

There are several types of multivariate performance measures. Some of them can be decomposed in an analogous fashion as the univariate performance measures. Bivariate performance measures, such as AUC (3.5), pairwise ranking error (3.4), and the magnitude preserving ranking error (4.2), decompose into a sum of pairwise comparisons of data points. Information retrieval specific measures such as MAP and NDCG, as well as the pairwise ranking error, allow a query-wise decomposition of the type

$$p(f(X), \mathbf{y}, Q) = \frac{1}{|Q|} \sum_{q \in Q} \overline{p}(f(X_q), \mathbf{y}_q),$$

where X_q and $\mathbf{y}_q$ contain the inputs and utility scores related to query q, and $\overline{p}$ is a performance measure that compares how well the rankings corresponding to the predicted and true scores match each other. Pairwise performance measures that are computed over queries can also be considered as query-wise performance measures in addition to being pairwise performance measures. Finally, we may have multivariate performance measures which do not decompose into subparts. Such are for example the precision, recall and F-score measure that have traditionally been used to evaluate information extraction systems (see e.g. Joachims (2005)).

5.2 Aims of performance estimation

Let us consider the different aims of performance evaluation. In the following, we adopt the shorthand notation $f_Z = \mathcal{A}(Z)$, used in Paper IV, to denote the prediction function returned by the learning algorithm $\mathcal{A}$ when trained on Z. We use $Z = (X, Y)$ to denote a training set, and $\overline{Z} = (\overline{X}, \overline{Y})$ to denote a test set sampled from our data distribution D. Following Dietterich (1998); Hastie et al. (2009); Schiavo and Hand (2000), we recognize two different questions in performance estimation.

First, let us assume that we are in advance given the training set Z, and we aim to estimate its generalization performance on new data. In this

setting, we aim to measure

$$A(f_Z) = \mathrm{E}_{\overline{Z}\sim D}[p(f_Z(\overline{X}), \overline{Y})], \tag{5.1}$$

which we call the *conditional performance*, since it is conditioned on a fixed training set.

On the other hand, if we want to estimate the average behavior of the learning algorithm, we take the expectation of (5.1) over all training sets resulting in

$$\mathrm{E}_{Z\sim D}[A(f_Z)] = \mathrm{E}_{Z,\overline{Z}\sim D}[p(f_Z(\overline{X}), \overline{Y})], \tag{5.2}$$

which we call the *unconditional performance*.

As discussed in Dietterich (1998); Hastie et al. (2009); Schiavo and Hand (2000), these two different measures correspond to two different questions of interest. Conditional performance measures how well a particular predictor that was constructed from a given training set can be expected to work on independent data. This question is relevant in machine learning applications, where a data set has been gathered in advance, and the subject of interest is the quality of the learned predictor. The variability resulting from the choice of different training sets is here of no interest, since one is already committed to using the given training set. In contrast, the unconditional performance measures the average behavior of the learning algorithm. The goal may be to evaluate the suitability of a certain algorithm to a given application domain, or to compare different algorithms. In this setting, the variability resulting from sampling of different training sets needs to be taken into account.

In practice we can never compute (5.1) or (5.2), but are rather limited to using some estimate $\hat{A}$ instead. The quality of a single conditional performance estimate can be measured using the deviation $\hat{A}(f_Z) - A(f_Z)$ (Braga-Neto and Dougherty, 2004). Analogously, for unconditional performance estimation, one can compute a deviation $\hat{A}(f_Z) - \mathrm{E}_{Z\sim D}[A(f_Z)]$. We may measure the quality of a performance estimation technique through the expected value (*bias*) and the *variance* of its deviation. Ideally, both of these should be as small as possible. The bias measures the tendency of a performance estimation technique to provide either too optimistic or pessimistic estimates of performance. It can be shown (see e.g. Paper IV), that the bias coincides regardless of whether we consider the conditional or unconditional setting. However, the deviation variance behaves differently in the two settings. In conditional performance estimation one aims to model the variability resulting from the training set choice, to achieve low variance the estimates need to be close to $A(f_Z)$. However, in unconditional performance estimation, the estimates should be close to the average case $\mathrm{E}_{Z\sim D}[A(f_Z)]$, in order to achieve low variance.

5.3 Cross-validation

It is commonly known that estimating the performance of a learned model on training data can lead to highly over-optimistic estimates of performance. This phenomenon is due to overfitting, as the particular model evaluated was in the first place specifically chosen for its fit to the training data. Rather, independent test data that was not observed during training is needed to test the learned models. In applications where data is scarce, large separate test sets often however cannot be afforded. Here, a technique known as *cross-validation* is typically applied.

In cross-validation, one repeatedly splits the data set into two parts, a training set and a holdout set. The model is trained on the training set, after which it is used to make predictions on the holdout set. This procedure is repeated a number of times, after which a final estimate of the performance is computed over all the holdout sets on which predictions were made on. One of the most common cross-validation techniques is *N-fold cross-validation*, whose most popular variant is 10-fold cross-validation. Here, the data is split into N mutually disjoint folds, each of which is in its turn used as the holdout set. An extreme variant of N-fold CV is leave-one-out cross-validation (LOOCV), where each example constitutes its own fold.

Formally, let $[m]$ denote the indices of the training examples. In cross-validation, we have a set $\mathcal{U} = \{U_1, \ldots, U_N\}$ of hold-out sets, where $N \in \mathbb{N}$ and $U_i \subseteq [m]$. By $f_{\overline{U}}$ we denote the prediction functions learned from all the training examples except those indexed by U. Then the cross-validation procedure consists of computing the predictions $f_{\overline{U}_i}(X_{U_i})$ for each $1 \leq i \leq N$, and aggregating the resulting performance estimates together.

There exists two general strategies for aggregating cross-validation results together. Bradley (1997) who considered the specific problem of AUC estimation referred to these alternatives as *pooling* and *averaging*. In pooling all the predictions are combined together and the performance measure is then computed over the combined predictions. In averaging, the performance is computed separately for each holdout set, and finally the average over these is computed. The pooling estimate can be written as

$$p([f_{\overline{U}_1}(X_{U_1}), \ldots, f_{\overline{U}_N}(X_{U_N})], [Y_{U_1}, \ldots, Y_{U_N}]),$$

whereas the averaged estimate is written as

$$\frac{1}{N} \sum_{i=1}^{N} p(f_{\overline{U}_i}(X_{U_i}), Y_{U_i}).$$

It is easy to see that for univariate performance measures the approaches yield equivalent results[1]. For this reasons, considering whether to use aver-

[1] Barring differences due to normalizations, if the holdout sets have differing sizes.

aging or pooling has not really been an issue in statistics or machine learning, when using traditional performance measures such as classification error rate or squared error. Similarly, as long as data points related to the same query are not split between different holdout sets, the approaches yield equivalent results. However for other multivariate performance measures such as AUC this equivalence does not hold. The study of this issue is the subject of Paper IV.

The application settings considered in this thesis constitute also a number of additional challenges for reliable performance estimation via cross-validation. Firstly, both the query-structured data encountered in many ranking settings (see Papers I, II, III), as well as the data encountered in the protein-protein interaction mining study presented in Paper V, strongly violate the independence assumptions that are typically made when applying cross-validation. Further, when using the Nyström approximation to create a feature representation for the data, it becomes apparent that the fact that the effect of holdout basis vectors needs to be removed from the mapping on each round of cross-validation (see Paper III). Next, we examine the aforementioned challenges in more detail.

5.4 AUC-estimation

Let us consider the problem of estimating the expected AUC performance of a classifier. We recall the two approaches to computing the cross-validation estimate, pooling and averaging. Let U_+ denote the positive, and U_- the negative examples in a set U. For AUC estimation, the pooling approach can be formalized as follows:

$$\frac{1}{N} \sum_{U,U' \in \mathcal{U}} \sum_{i \in U_+, j \in U'_-} H(f_{\overline{U}}(x_i) - f_{\overline{U'}}(x_j)),$$

where the normalizer $N = \sum_{U,U' \in \mathcal{U}} |U_+||U'_-|$ is the number of positive-negative pairs encountered in the summation, and H is the Heaviside step function (3.2). For pooling we assume that the folds are defined such that $U \cap U' = \emptyset$ for all $U, U' \in \mathcal{U}$.

The averaging estimate can be written as

$$\frac{1}{N} \sum_{U \in \mathcal{U}} \sum_{i \in U_+, j \in U_-} H(f_{\overline{U}}(x_i) - f_{\overline{U}}(x_j)),$$

where $N = \sum_{U \in \mathcal{U}} |U_+||U_-|$.

The major difference between the approaches is that in pooling, predictions made by different models are compared together, whereas this is not the case in averaging. The pooling approach allows making use of all

the example pairs in the training set, whereas this is not the case for typical averaging strategies. Previously, Parker et al. (2007) has shown that the pooling approach can lead to large pessimistic biases. In their experiments with no-signal data sets, AUC values of less than 0.3 were observed instead of the expected 0.5. On the other hand such bias was not observed in averaging. Similar results have been demonstrated in Forman and Scholz (2010).

Based on these considerations we propose using *leave-pair-out cross-validation* (LPOCV) for AUC-estimation. The method is an extreme form of averaging, where each positive-negative example pair constitutes a hold-out set. The method allows using all the positive-negative pairs to compute the performance estimate, while still comparing only predictions made by the same predictor. The LPOCV estimate can be formalized as :

$$\frac{1}{|[m]_+||[m]_-|} \sum_{i \in [m]_+} \sum_{j \in [m]_-} H(f_{\overline{\{i,j\}}}(x_i) - f_{\overline{\{i,j\}}}(x_j)),$$

where $f_{\overline{\{i,j\}}}$ denotes a classifier trained without the i-th and j-th training instance, and $[m]_+ \subset [m]$ and $[m]_- \subset [m]$ denote the indices of the positive and negative instances in the training set Z, respectively.

Paper IV studies through an extensive simulation study the problem of estimating the conditional AUC performance of a trained classifier via cross-validation. The main conclusions of the paper are the following. First, it was observed that the pooling approach indeed can lead to clear negative bias, whereas standard averaging strategies such as averaged 10-fold cross-validation can have unacceptably high variance on small data sets. LPOCV is shown to be close to unbiased, while having also as low variance as that of the most competitive compared approaches. On large enough datasets (several hundreds of examples or more), standard averaging approaches prove also to be as reliable as LPOCV.

The main shortcoming of the LPOCV is its time complexity, since it requires re-training the classifier a quadratic number of times, with respect to the training set size. To make the approach practical, we propose in Paper I computational shortcuts for computing the LPOCV estimate for the RankRLS method. The algorithm presented in Pahikkala et al. (2006a) can be used to achieve similar speedups for LPOCV with basic RLS regression. As an example application, we note that the simulation study that is the main contribution of Paper IV would not have been possible without the computational shortcuts, due to combinatorial explosion resulting from the combined number of repetitions of the experiments, number of cross-validation rounds etc. For example, implementing the sample size experiment presented in Figures 8 and 9 in Paper IV would require re-training a classifier of the order of 10^9 times.

A relevant question is, how other multivariate performance measures than AUC behave when considering the pooling and averaging approaches. It seems prudent to assume that the results should directly extend to the behavior of disagreement error, as well as to the magnitude preserving ranking error, whose estimation via LPOCV was proposed in Cortes et al. (2007a). For query-based loss functions averaging and pooling approaches lead to equivalent results as long as data points related to the same query are not divided across folds. However, for multivariate performance measures such as F-score it is easy to establish that pooling and averaging will lead to different results, but it is not straightforward to ascertain what this means in terms of reliability for the two approaches.

5.5 Data dependencies

Let us recall the ranking problem on query structured data, considered in Section 3.1. In a standard approach to learning to rank for information retrieval (Liu, 2009), the data consists of query-document pairs, where each data point has a feature representation that captures combined properties of a query and a document, and a label denoting how well the document matches the query. A similar ranking setting is studied in Tsivtsivadze et al. (2005), where the problem of re-ranking the syntactic parses produced by an automatic parser was modeled in the same framework, encoding the data as sentence-parse pairs.

Clearly, objects related to the same query are not independent of each other. A trained predictor can be in general be expected to perform better on such queries it has already observed during training, than it would on average perform on new unknown queries. Thus ignoring this structure can lead to strongly optimistic bias in cross-validation, if examples belonging to same query are split between training and test folds. Pahikkala et al. (2006a) experimentally demonstrated how on the task of parse ranking splitting related parses between training and test folds could lead to observed 0.7 Kendalls τ_b correlation instead of the 0.3 τ_b obtained using an estimation strategy, where the fold splits were defined on the level of queries.

Another type of data dependency was noted in Sætre et al. (2007) on the task of protein-protein interaction extraction from scientific literature. Here, each example represents a pair of protein names appearing in a sentence, and the correct label is whether the proteins are stated to have an interaction or not. There are two widely used approaches to perform cross-validation in this domain, one where the fold splits are defined randomly on the level of pairs, and one where pairs belonging to the same sentence, or article, are never divided into separate folds. The former approach can be considered problematic due to dependencies between examples originating

from a same sentence. To recall an example presented in Paper V, let us consider two interaction candidates extracted from the same sentence, e.g. from a statement of the form "P1 and P2 [...] P3", where "[...]" is any statement of interaction or non-interaction. Due to the near-identity of contexts a machine learning method should, for any reasonable feature representation, easily be able to predict the label of the pair (P1, P3), if it has seen (P2, P3) during training. This is because feature representations that are based on the textual context of the words, such as the one corresponding to the kernel presented in Paper V, will typically be almost identical for both pairs. However, one would not in real-world applications expect to often encounter the case where at test time predictions were needed for pairs of proteins appearing in sentences that were also part of the training set. Sætre et al. (2007) demonstrated, that the approach of randomly splitting the data into folds without taking into account the sentence structure could lead to up to 0.18 increase in F-score performance on a widely used benchmark data set, compared to the correct approach.

These findings support the notion that for data where strong dependencies occur between the examples, dependent data points should never be split between the training and test sets. These motivate cross-validation approaches such leave-query-out or leave-document out cross-validation, where one can make maximal use of the available data while still ensuring reliable performance estimation. A leave-document-out cross-validation procedure is implemented in the experimental setting of Paper V. The leave-query-out cross-validation procedure is considered in detail in Paper I, where an efficient algorithm for computing this estimate for RankRLS is presented.

5.6 Reduced set approximation

In Section 2.5 we considered training reduced set approximations of kernel methods, and discussed how this can be achieved using the Nyström approximation of the kernel matrix. Let us recall the formula for the Nyström approximation $\widetilde{\mathbf{K}} = (\mathbf{K}_{\mathcal{R}})^{\mathrm{T}}(\mathbf{K}_{\mathcal{R}\mathcal{R}})^{-1}\mathbf{K}_{\mathcal{R}}$, where $\mathcal{R}$ denotes the indices of the basis vectors. An empirical kernel map (2.4), corresponding to the reduced set approximation, can be defined as

$$(\widetilde{\Phi}_X(X))^{\mathrm{T}} = \mathbf{K}_{\mathcal{R}}^{\mathrm{T}}(\mathrm{C}^{\mathrm{T}})^{-1},$$

where $\mathrm{C}\mathrm{C}^{\mathrm{T}} = \mathbf{K}_{\mathcal{R}\mathcal{R}}$ denotes the Cholesky decomposition of $\mathbf{K}_{\mathcal{R}\mathcal{R}}$.

This mapping results in a $m \times |\mathcal{R}|$ data matrix, on which a linear algorithm can subsequently be trained on. The time complexity of computing the mapping is $O(m|\mathcal{R}|^2)$. However, as discussed in Paper III, doing cross-validation with the mapped data is problematic. This is, because the mapping is data dependent, meaning that it is affected by the choice of basis

vectors. Whenever the holdout set contains basis vectors, one needs to take into account the effect that removing elements from $\mathcal{R}$ has on $(\widetilde{\Phi}_X(X))^{\mathrm{T}}$. The most straightforward approach to N-fold cross-validation, where the mapping is re-computed on every round of cross-validation, results in the $O(Nm|\mathcal{R}|^2)$ time complexity for computing the mappings. However, the computations can be made more efficient using matrix update formulas. Let $\mathcal{H} \subseteq \mathcal{R}$ denote the subset of basis vectors belonging to the holdout set. Theorem 1 in Paper III states that the modified mapping where the effect of the basis vectors in $\mathcal{H}$ are removed can be computed in $O(m|\mathcal{R}||\mathcal{H}|)$ time. The presented update formula is somewhat complicated, though based on well known matrix identities and decompositions. Further, Corollary 2 in Paper III, states that as a result, the time complexity of N-fold cross-validation is $O(m|\mathcal{R}|^2)$.

Previously, Cawley and Talbot (2004) derived a fast LOOCV formula for the reduced set approximation of RLS. The approach had however the limitation that when the holdout example was a basis vector, its effect on the mapping was not removed in the proposed approach. Pahikkala et al. (2009) improved the method to ensure that the effect of holdout basis vectors is removed, and generalized it to N-fold cross-validation. The method can be considered a special case of the approach presented in Paper III, optimized to take advantage of the close formed solution of the RLS method. The method is used to implement the leave-document-out cross-validation procedure for the reduced set approximation of RLS in the experimental evaluation in Paper V, where the problem of protein-protein interaction extraction from scientific literature is studied.

Chapter 6

Conclusions

6.1 Contributions of the thesis

In this thesis, we both introduced novel efficient algorithms for learning with regularized kernel methods, and addressed the problem of reliable validation of results. Specifically, we studied the general problem of learning to rank, and its special case AUC optimization, as well as the use of cross-validation for performance evaluation. The studied problems are motivated by real life applications, of which the problem of protein-protein interaction extraction is presented as a case study in the second part of this thesis.

Papers I and II introduce novel computationally efficient algorithms for learning to rank. The considered methods are regularized kernel methods, allowing for principled balancing between under- and overfitting, learning from structured data, and recovery of globally optimal solutions via convex optimization. The RankRLS method (Paper I) is based on the pairwise least-squares loss, the method is a novel contribution resulting from the research work described in this thesis. In addition to efficient training methods, we introduce fast algorithms for cross-validation, parameter selection, and multi-output learning. The RankSVM is a well established ranking method previously known in the literature, the contributions of this thesis lie in improving the fastest known training methods for linear RankSVM training (Paper II). Paper III, which studies the use of the Nyström approximation of the kernel matrix for training reduced set approximations, presents a straightforward way for extending the results of Paper II for developing an efficient solver for kernel RankSVM training, an approach briefly considered already in Paper I.

Paper IV presents an experimental study comparing different cross-validation approaches for AUC estimation. The main findings of the study are that averaged cross-validation approaches should be preferred over pooled ones, and that the LPOCV approach can provide high quality AUC

estimation, if it can be computationally afforded. Paper III discusses the challenges of using cross-validation when training methods on a feature map recovered from the Nyström approximation of kernel matrix. For reliable performance estimation removing the effect of basis vectors included in the current test set proves to be necessary, computationally efficient algorithms for doing this are presented.

Both computational complexity analysis and practical experiments support the notion that the introduced algorithms make it possible to apply kernel based ranking methods to larger datasets than was previously possible. Experimental results demonstrate that the methods achieve high ranking performance on a number of datasets from different application domains, and that RankRLS appears to have similar predictive performance as the previously well established RankSVM method. The analysis of different cross-validation techniques identifies suitable methods for reliable performance estimation when dealing with pairwise ranking measures, dependent data points, and reduced set approximations. The introduced fast algorithms make the techniques practical from computational efficiency point of view. Software implementations of the methods developed during the thesis work are made publicly available under open source software licence, in order to support the adaptation of the methods in data intensive areas of research and engineering. A possible limitation of the proposed ranking algorithms is that in settings where it is much more important to get the top ranked entries in the ranking correct than the rest, other approaches such as those based on listwise modeling may be more appropriate.

The protein-protein interaction extraction study in Paper V combines together a number of approaches considered in this thesis, such as the use of kernel functions and RLS based learners, computational shortcuts for cross-validation, the combination of cross-validation and Nyström approximation, and learner evaluation via averaged AUC estimation. Paper V further considers a number of pitfalls that can affect the reliability of achieved results, including incorrect cross-validation strategies that ignore dependencies between the data points. The work results in a novel system for protein-protein interaction extraction, and a number of recommendations on how to improve the validity and comparability of different studies in the field. The recommendations have been adopted in several subsequent studies (see e.g. Miwa et al. (2009); Fayruzov et al. (2009); Choi and Myaeng (2010); Kuksa et al. (2010); Tikk et al. (2010)).

6.2 Open source software

The importance of sharing open source implementations of published methods has recently been advocated in the machine learning community. Son-

nenburg et al. (2007) discusses a number of advantages provided by this model, including better reproducibility of experimental results, detection of implementational errors, avoidance of unnecessary re-implementation work, increased possibilities for combining different methods together, and faster adaptation of methods in application fields. In a similar vein, Pedersen (2008) discusses in the aptly titled article "Empiricism is not a matter of faith", the importance of sharing software in order to ensure the reproducibility of published results.

In our opinion, the raised issues are more than valid. Therefore, we have during the thesis work begun the development of the RLScore machine learning open source software framework, which is made publicly available[1] under the MIT open source license. Further, both the All-paths graph kernel -software package, as well as the five datasets used to implement the experiments described in Paper V, have been made publicly available[2], under an open source licence. Finally, an open source implementation for the efficient linear RankSVM training algorithm described in Paper II is also made publicly available in the TreeRankSVM package.

At the time of writing this thesis, the current 0.4 version of RLScore contains implementations for a majority of methods introduced in this thesis, including the RLS and RankRLS learners, kernel functions, computational shortcuts for cross-validation, parameter selection, and multi-output learning, as well as reduced set approximations and fast performance measure implementations. Additionally, the package contains methods for fast greedy forward feature selection for both RLS and RankRLS methods (Pahikkala et al., 2010c,a), as well as a RLS based maximum-margin clustering algorithm (Gieseke et al., 2009). Experimental code not yet available in the publicly available main distribution includes bundle methods for SVM training, as well as methods for relational learning using pairwise Kronecker kernels (Pahikkala et al., 2010d). The RLScore package is implemented in the Python programming language. The choice was made due to ease of implementation allowed by a high-level language, coupled with the availability of high-quality matrix and optimization libraries such as NumPy, SciPy and CVXOPT. Further, Python can be easily extended to integrate optimized implementations written in low-level languages such as C or Fortran, an approach we chose for implementing the RankSVM solver described in Paper II.

[1] `http://tucs.fi/rlscore`

[2] `http://mars.cs.utu.fi/PPICorpora/GraphKernel.html`

6.3 Future work

A main theme in the research presented in this thesis was the development of computationally efficient algorithms, especially by means of matrix algebra. A natural future direction of study is to explore what other computational shortcuts are possible for the considered, or other learning methods having similar properties, and whether such new shortcuts are compatible with the existing ones. As an example, we have lately studied the problem of feature selection for linear RLS and RankRLS learners (Pahikkala et al., 2010c,a). The feature selection process can be modeled as a search, where the cross-validation shortcuts are used to evaluate the quality of the considered feature subsets at each search step, and matrix update formulas are used to add the effect of new chosen features to the current solution. Similarly, the problem of online learning where new data becomes available over time can be efficiently solved for the RLS method through the use of update formulas (Pahikkala et al., 2011). It is more than likely that one could for example derive an efficient online version of RankRLS, which updates the solution one query at a time.

All the methods considered in this thesis are supervised learning methods, meaning that it is assumed that all the training examples come with label information. It would be beneficial if the methods could also make use of unlabeled data, that is often available in much larger quantities than labeled data. The area of semi-supervised learning addresses this problem (Chapelle et al., 2006). For example, semi-supervised SVMs aim to improve classification performance by requiring the decision hyperplane to pass through a low density area in the feature space, as determined by both labeled and unlabeled data (Chapelle et al., 2008). There is some existing work on kernel methods for semi-supervised ranking (see e.g. Chu and Ghahramani (2005a)), and Tsivtsivadze et al. (2011) has recently introduced a semi-supervised variant of RankRLS based on the idea of co-regularization. Still, further research is required in order to achieve efficient semi-supervised methods that can consistently outperform their purely supervised counterparts. A particular unsolved problem in the area is that of model selection. Semi-supervised methods tend to have many adjustable parameters, but are applied in settings where there is very little labeled data available to guide the correct selection of these parameters.

The RLScore open source software is still a work in progress. Future releases of the software are expected to incorporate new learning algorithms developed in our research. Parallelization of the learning methods provides a promising direction to applying the methods on very large datasets. Previously, general frameworks for parallelizing machine learning methods have been proposed for example in Chu et al. (2007); Low et al. (2010). Currently, the methods included in the package are being used in a collaborative re-

search project for analyzing data from Genome Wide Association Studies (see e.g. Okser et al. (2010) for relevant prior work).

Bibliography

Agarwal, S., Graepel, T., Herbrich, R., Har-Peled, S., and Roth, D. (2005). Generalization bounds for the area under the ROC curve. *Journal of Machine Learning Research*, 6:393–425.

Ailon, N. and Mohri, M. (2008). An efficient reduction of ranking to classification. In Servedio, R. A. and Zhang, T., editors, *Proceedings of the 21st Annual Conference on Learning Theory (COLT 2008)*, pages 87–98. Omnipress.

Airola, A., Pahikkala, T., and Salakoski, T. (2010). Large scale training methods for linear RankRLS. In Hüllermeier, E. and Fürnkranz, J., editors, *Proceedings of the ECML/PKDD-Workshop on Preference Learning (PL-10)*.

An, S., Liu, W., and Venkatesh, S. (2007). Fast cross-validation algorithms for least squares support vector machine and kernel ridge regression. *Pattern Recognition*, 40(8):2154–2162.

Argall, B. D., Chernova, S., Veloso, M., and Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469 – 483.

Ball, N. M., Brunner, R. J., Myers, A. D., and Tcheng, D. (2006). Robust machine learning applied to astronomical data sets. I. Star-galaxy classification of the Sloan digital sky survey DR3 using decision trees. *The Astrophysical Journal*, 650(1):497.

Bamber, D. (1975). The area above the ordinal dominance graph and the area below the receiver operating characteristic graph. *Journal of Mathematical Psychology*, 12:387–415.

Bengio, Y. and LeCun, Y. (2007). Scaling learning algorithms towards AI. In Bottou, L., Chapelle, O., Decoste, D., and Weston, J., editors, *Large-Scale Kernel Machines*. MIT Press.

Berry, M. J. A. and Linoff, G. S. (2004). *Data Mining Techniques: For Marketing, Sales, and Customer Relationship Management*. John Wiley & Sons.

Bottou, L. and Lin, C.-J. (2007). Support vector machine solvers. In Bottou, L., Chapelle, O., DeCoste, D., and Weston, J., editors, *Large-Scale Kernel Machines*, Neural Information Processing, pages 1–28. MIT Press, Cambridge, MA, USA.

Bradley, A. P. (1997). The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159.

Braga-Neto, U. M. and Dougherty, E. R. (2004). Is cross-validation valid for small-sample microarray classification? *Bioinformatics*, 20(3):374–380.

Brefeld, U. and Scheffer, T. (2005). AUC maximizing support vector learning. In Lachiche, N., Ferri, C., Macskassy, S. A., and Rakotomamonjy, A., editors, *Proceedings of the 2nd Workshop on ROC Analysis in Machine Learning (ROCML 2005)*.

Brualdi, R. A. and Ryser, H. J. (1991). *Combinatorial Matrix Theory*. Cambridge University Press.

Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., and Hullender, G. (2005). Learning to rank using gradient descent. In Raedt, L. D. and Wrobel, S., editors, *Proceedings of the 22nd international conference on Machine learning (ICML 2005)*, pages 89–96, New York, NY, USA. ACM.

Burges, C. J. C., Ragno, R., and Le, Q. V. (2007). Learning to rank with nonsmooth cost functions. In Schölkopf, B., Platt, J. C., Hoffman, T., Schölkopf, B., Platt, J. C., and Hoffman, T., editors, *Advances in Neural Processing Systems 19*, pages 193–200. MIT Press.

Cao, Y., Xu, J., Liu, T.-Y., Li, H., Huang, Y., and Hon, H.-W. (2006). Adapting ranking SVM to document retrieval. In Efthimiadis, E. N., Dumais, S. T., Hawking, D., and Järvelin, K., editors, *Proceedings of the 29th annual international ACM SIGIR conference on research and development in information retrieval (SIGIR 2006)*, pages 186–193, New York, NY, USA. ACM.

Cao, Z., Qin, T., Liu, T.-Y., Tsai, M.-F., and Li, H. (2007). Learning to rank: from pairwise approach to listwise approach. In Ghahramani, Z., editor, *Proceedings of the 24th international conference on Machine learning (ICML 2007)*, pages 129–136, New York, NY, USA. ACM.

Cawley, G. C. and Talbot, N. L. C. (2004). Fast exact leave-one-out cross-validation of sparse least-squares support vector machines. *Neural Networks*, 17(10):1467–1475.

Cesa-Bianchi, N. (2007). Applications of regularized least squares to pattern classification. *Theoretical Computer Science*, 382:221–231.

Chang, Y.-W., Hsieh, C.-J., Chang, K.-W., Ringgaard, M., and Lin, C.-J. (2010). Training and testing low-degree polynomial data mappings via linear SVM. *Journal of Machine Learning Research*, 11:1471–1490.

Chapelle, O. and Chang, Y. (2011). Yahoo! learning to rank challenge overview. In Chapelle, O., Chang, Y., and Liu, T.-Y., editors, *JMLR Workshop and Conference Proceedings: Yahoo! Learning to Rank Challenge*, volume 14 of *JMLR Workshop and Conference Proceedings*, pages 1–24.

Chapelle, O. and Keerthi, S. S. (2010). Efficient algorithms for ranking with SVMs. *Information Retrieval*, 13(3):201–215.

Chapelle, O., Le, Q., and Smola, A. (2007). Large margin optimization of ranking measures. In Zhou, D., Chapelle, O., Joachims, T., and Hofmann, T., editors, *NIPS 2007 Workshop on Machine Learning for Web Search*.

Chapelle, O., Metlzer, D., Zhang, Y., and Grinspan, P. (2009). Expected reciprocal rank for graded relevance. In Cheung, D. W.-L., Song, I.-Y., Chu, W. W., Hu, X., and Lin, J. J., editors, *Proceeding of the 18th ACM conference on Information and knowledge management (CIKM 2009)*, pages 621–630, New York, NY, USA. ACM.

Chapelle, O., Schölkopf, B., and Zien, A., editors (2006). *Semi-Supervised Learning*. MIT Press, Cambridge, MA.

Chapelle, O., Sindhwani, V., and Keerthi, S. S. (2008). Optimization techniques for semi-supervised support vector machines. *Journal of Machine Learning Research*, 9:203–233.

Choi, S.-P. and Myaeng, S.-H. (2010). Simplicity is better: revisiting single kernel ppi extraction. In Huang, C.-R. and Jurafsky, D., editors, *Proceedings of the 23rd International Conference on Computational Linguistics (COLING 2010)*, pages 206–214, Stroudsburg, PA, USA. Association for Computational Linguistics.

Chu, C.-T., Kim, S. K., Lin, Y.-A., Yu, Y., Bradski, G., Ng, A. Y., and Olukotun, K. (2007). Map-reduce for machine learning on multicore. In

Schölkopf, B., Platt, J., and Hoffman, T., editors, *Advances in Neural Information Processing Systems 19*, pages 281–288. MIT Press, Cambridge, MA.

Chu, W. and Ghahramani, Z. (2005a). Extensions of Gaussian processes for ranking: semi-supervised and active learning. In Agarwal, S., Cortes, C., and Herbrich, R., editors, *Proceedings of NIPS 2005 Workshop on Learning to Rank*, pages 29–34.

Chu, W. and Ghahramani, Z. (2005b). Gaussian processes for ordinal regression. *Journal of Machine Learning Research*, 6:2005.

Chu, W. and Keerthi, S. S. (2005). New approaches to support vector ordinal regression. In Raedt, L. D. and Wrobel, S., editors, *Proceedings of the 22nd international conference on Machine learning (ICML 2005)*, pages 145–152, New York, NY, USA. ACM.

Chu, W. and Keerthi, S. S. (2007). Support vector ordinal regression. *Neural Computation*, 19(3):792–815.

Clémençon, S., Lugosi, G., and Vayatis, N. (2005). Ranking and scoring using empirical risk minimization. In Auer, P. and Meir, R., editors, *Proceedings of the 18th Annual Conference on Learning Theory (COLT 2005)*, volume 3559 of *Lecture Notes in Computer Science*, pages 1–15. Springer.

Cohen, W. W., Schapire, R. E., and Singer, Y. (1998). Learning to order things. *Journal of Artificial Intelligence Research*, 10:243–270.

Cooper, W. S., Gey, F. C., and Dabney, D. P. (1992). Probabilistic retrieval based on staged logistic regression. In Belkin, N. J., Ingwersen, P., and Pejtersen, A. M., editors, *Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR 1992)*, pages 198–210, New York, NY, USA. ACM.

Cortes, C. and Mohri, M. (2004). AUC optimization vs. error rate minimization. In Thrun, S., Saul, L., and Schölkopf, B., editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, Massachusetts, USA.

Cortes, C., Mohri, M., and Rastogi, A. (2007a). An alternative ranking problem for search engines. In Demetrescu, C., editor, *Proceedings of the 6th Workshop on Experimental Algorithms (WEA 2007)*, volume 4525 of *Lecture Notes in Computer Science*, pages 1–21. Springer, Berlin / Heidelberg, Germany.

Cortes, C., Mohri, M., and Rastogi, A. (2007b). Magnitude-preserving ranking algorithms. In Ghahramani, Z., editor, *Proceedings of the 24th Annual International Conference on Machine Learning (ICML 2007)*, volume 227 of *ACM International Conference Proceeding Series*, pages 169–176, New York, NY, USA. ACM Press.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297.

Cossock, D. and Zhang, T. (2006). Subset ranking using regression. In Lugosi, G. and Simon, H.-U., editors, *Proceedings of the 19th Annual Conference on Learning Theory (COLT 2006)*, pages 605–619.

Crammer, K. and Singer, Y. (2002). Pranking with ranking. In Dietterich, T. G., Becker, S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems 14*, pages 641–647. MIT Press.

Dekel, O., Manning, C. D., and Singer, Y. (2003). Log-linear models for label ranking. In Thrun, S., Saul, L. K., and Schölkopf, B., editors, *Advances in Neural Information Processing Systems 16*. MIT Press.

Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10:1895–1923.

Domshlak, C., Hüllermeier, E., Kaci, S., and Prade, H. (2011). Preferences in AI: An overview. *Artificial Intelligence*, 175:1037–1052.

Drucker, H., Burges, C. J., Kaufman, L., Smola, A., and Vapnik, V. (1997). Support vector regression machines. In Mozer, M., Jordan, M., and Petsche, T., editors, *Advances in Neural Information Processing Systems 9*, pages 155–161, Cambridge. MIT Press.

Evgeniou, T., Pontil, M., and Poggio, T. (2000). Regularization networks and support vector machines. *Advances in Computational Mathematics*, 13:1–50.

Fawcett, T. (2001). Using rule sets to maximize ROC performance. In Cercone, N., Lin, T. Y., and Wu, X., editors, *Proceedings of the 2001 IEEE International Conference on Data Mining (ICDM 2001)*, pages 131–138, Washington, DC, USA. IEEE Computer Society.

Fawcett, T. and Flach, P. A. (2005). A response to Webb and Ting's "On the application of ROC analysis to predict classification performance under varying class distributions". *Machine Learning*, 58(1):33–38.

Fayruzov, T., De Cock, M., Cornelis, C., and Hoste, V. (2009). Linguistic feature analysis for protein interaction extraction. *BMC Bioinformatics*, 10(1):374.

Fayyad, U., Piatetsky-shapiro, G., and Smyth, P. (1996). From data mining to knowledge discovery in databases. *AI Magazine*, 17:37–54.

Ferri, C., Flach, P. A., and Hernández-Orallo, J. (2002). Learning decision trees using the area under the ROC curve. In Sammut, C. and Hoffmann, A. G., editors, *Proceedings of the Nineteenth International Conference on Machine Learning (ICML 2002)*, pages 139–146, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Forman, G. and Scholz, M. (2010). Apples-to-apples in cross-validation studies: Pitfalls in classifier performance measurement. *SIGKDD Explorations*, 12(1):49–57.

Franc, V. and Sonnenburg, S. (2009). Optimized cutting plane algorithm for large-scale risk minimization. *Journal of Machine Learning Research*, 10:2157–2192.

Frank, E. and Hall, M. (2001). A simple approach to ordinal classification. In Raedt, L. D. and Flach, P. A., editors, *Proceedings of the 12th European Conference on Machine Learning (ECML 2001)*, pages 145–156, London, UK. Springer-Verlag.

Freund, Y., Iyer, R., Schapire, R. E., and Singer, Y. (2003). An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4:933–969.

Fuhr, N. (1989). Optimum polynomial retrieval functions based on the probability ranking principle. *ACM Transactions on Information Systems*, 7(3):183–204.

Fürnkranz, J. and Hüllermeier, E. (2010). Preference learning. In *Encyclopedia of Machine Learning*, pages 789–795.

Gärtner, T., Flach, P. A., and Wrobel, S. (2003). On graph kernels: Hardness results and efficient alternatives. In Schölkopf, B. and Warmuth, M. K., editors, *Proceedings of the Sixteenth Annual Conference on Learning Theory and Seventh Annual Workshop on Kernel Machines (COLT/Kernel 2003)*, volume 2777 of *Lecture Notes in Artificial Intelligence*, pages 129–143. Springer.

Gieseke, F., Pahikkala, T., and Kramer, O. (2009). Fast evolutionary maximum margin clustering. In Danyluk, A. P., Bottou, L., and Littman,

M. L., editors, *Proceedings of the 26th Annual International Conference on Machine Learning (ICML 2009)*, volume 382 of *ACM International Conference Proceeding Series*, pages 361–368, New York, NY, USA. ACM.

Ginter, F. (2007). *Information Extraction in the Biomedical Domain: Methods and Resources*. PhD thesis, Turku Centre for Computer Science (TUCS).

Golub, G. H. and Loan, C. V. (1989). *Matrix Computations*. The Johns Hopkins University Press, Baltimore and London, second edition.

Gong, Y. (1995). Speech recognition in noisy environments: A survey. *Speech Communication*, 16(3):261 – 291.

Hanley, J. A. and McNeil, B. J. (1982). The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 143(1):29–36.

Harmeling, S., Ziehe, A., Kawanabe, M., and Müller, K.-R. (2002). Kernel feature spaces and nonlinear blind source separation. In Dietterich, T. G., Becker, S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems 14*, pages 761–768. MIT Press.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference and Prediction, Second Edition*. Springer Series in Statistics. Springer.

Herbrich, R., Graepel, T., and Obermayer, K. (1999). Support vector learning for ordinal regression. In *Proceedings of the Ninth International Conference on Articial Neural Networks (ICANN 1999)*, pages 97–102, London. Institute of Electrical Engineers.

Herbrich, R., Graepel, T., and Obermayer, K. (2000). *Large Margin Rank Boundaries for Ordinal Regression*. MIT Press.

Herschtal, A. and Raskutti, B. (2004). Optimising area under the ROC curve using gradient descent. In Brodley, C. E., editor, *Proceedings of the twenty-first international conference on Machine learning (ICML 2004)*, pages 49–56, New York, NY, USA. ACM.

Hey, T., Tansley, S., and Tolle, K., editors (2009). *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research, Redmond, USA.

Hoerl, A. E. and Kennard, R. W. (1970). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12:55–67.

Huang, J. and Ling, C. X. (2005). Using AUC and accuracy in evaluating learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 17(3):299–310.

Järvelin, K. and Kekäläinen, J. (2000). Ir evaluation methods for retrieving highly relevant documents. In Belkin, N. J., Ingwersen, P., and Leong, M.-K., editors, *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR 2000)*, pages 41–48, New York, NY, USA. ACM.

Joachims, T. (1998). Making large-scale support vector machine learning practical. In Schölkopf, C. B., editor, *Advances in Kernel Methods: Support Vector Machines*. MIT Press, Cambridge, MA.

Joachims, T. (2002a). *Learning to Classify Text Using Support Vector Machines – Methods, Theory, and Algorithms*. Kluwer/Springer.

Joachims, T. (2002b). Optimizing search engines using clickthrough data. In Hand, D., Keim, D., and Ng, R., editors, *Proceedings of the 8th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2002)*, pages 133–142, New York, NY, USA. ACM Press.

Joachims, T. (2005). A support vector method for multivariate performance measures. In Raedt, L. D. and Wrobel, S., editors, *Proceedings of the 22nd International Conference on Machine learning (ICML 2005)*, volume 119 of *ACM International Conference Proceeding Series*, pages 377–384, New York, NY, USA. ACM Press.

Joachims, T. (2006). Training linear SVMs in linear time. In Eliassi-Rad, T., Ungar, L. H., Craven, M., and Gunopulos, D., editors, *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD 2006)*, pages 217–226, New York, NY, USA. ACM Press.

Joachims, T. and Radlinski, F. (2007). Search engines that learn from implicit feedback. *IEEE Computer*, 40(8):34–40.

Joachims, T. and Yu, C.-N. J. (2009). Sparse kernel SVMs via cutting-plane training. *Machine Learning*, 76(2-3):179–193.

King, R. D., Rowland, J., Oliver, S. G., Young, M., Aubrey, W., Byrne, E., Liakata, M., Markham, M., Pir, P., Soldatova, L. N., Sparkes, A., Whelan, K. E., and Clare, A. (2009). The automation of science. *Science*, 324:85–89.

Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *IEEE Computer*, 42(8):30–37.

Kuksa, P., Qi, Y., Bai, B., Collobert, R., Weston, J., Pavlovic, V., and Ning, X. (2010). Semi-supervised abstraction-augmented string kernel for multi-level bio-relation extraction. In Balcazar, J. L., Bonchi, F., Gionis, A., and Sebag, M., editors, *Proceedings of the 2010 European conference on Machine learning and knowledge discovery in databases: Part II (ECML PKDD 2010)*, pages 128–144, Berlin, Heidelberg. Springer-Verlag.

Kumar, S., Mohri, M., and Talwalkar, A. (2009). Sampling techniques for the Nyström method. In van Dyk, D. and Welling, M., editors, *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS 2009)*, pages 304–311.

Lan, Y., Liu, T.-Y., Ma, Z., and Li, H. (2009). Generalization analysis of listwise learning-to-rank algorithms. In Danyluk, A. P., Bottou, L., and Littman, M. L., editors, *Proceedings of the 26th Annual International Conference on Machine Learning (ICML 2009)*, pages 577–584, New York, NY, USA. ACM.

Langley, P. (2011). The changing science of machine learning. *Machine Learning*, 82:275–279.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Legg, S. and Hutter, M. (2007). Universal intelligence: A definition of machine intelligence. *Minds and Machines*, 17(4):391–444.

Li, P., Burges, C. J. C., and Wu, Q. (2008). Mcrank: Learning to rank using multiple classification and gradient boosting. In Platt, J. C., Koller, D., Singer, Y., and Roweis, S. T., editors, *Advances in Neural Information Processing Systems 20*. MIT Press.

Liu, T.-Y. (2009). Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331.

Low, Y., Gonzalez, J., Kyrola, A., Bickson, D., Guestrin, C., and Hellerstein, J. M. (2010). Graphlab: A new framework for parallel machine learning. In *The 26th Conference on Uncertainty in Artificial Intelligence (UAI 2010)*, pages 340–349, Corvallis, Oregon. AUAI Press.

Manning, C. D. and Schütze, H. (1999). *Foundations of statistical natural language processing*. MIT Press, Cambridge, MA, USA.

McCullagh, P. (1980). Regression models for ordinal data. *Journal of the Royal Statistical Society. Series B (Methodological)*, 42(2):109–142.

Meyer, C. D. (2000). *Matrix analysis and applied linear algebra*. Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, USA.

Minkov, E., Charrow, B., Ledlie, J., Teller, S., and Jaakkola, T. (2010). Collaborative future event recommendation. In Huang, J., Koudas, N., Jones, G. J. F., Wu, X., Collins-Thompson, K., and An, A., editors, *Proceedings of the 19th ACM international conference on Information and knowledge management (CIKM 2010)*, pages 819–828, New York, NY, USA. ACM.

Minsky, M. and Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA.

Mitchell, T. (2006). The discipline of machine learning. Technical Report CMU-ML-06-108, Carnegie Mellon University.

Miwa, M., Sætre, R., Miyao, Y., and Tsujii, J. (2009). Protein-protein interaction extraction by leveraging multiple kernels and parsers. *International Journal of Medical Informatics*, 78:e39–e46.

Nallapati, R. (2004). Discriminative models for information retrieval. In Sanderson, M., Järvelin, K., Allan, J., and Bruza, P., editors, *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR 2004)*, pages 64–71, New York, NY, USA. ACM.

Nrgaard, M., Ravn, O. E., Poulsen, N. K., and Hansen, L. K. (2000). *Neural Networks for Modelling and Control of Dynamic Systems: A Practitioner's Handbook*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Okser, S., Lehtimäki, T., Elo, L. L., Mononen, N., Peltonen, N., Kähönen, M., Juonala, M., Fan, Y.-M., Hernesniemi, J. A., Laitinen, T., Lyytikäinen, L.-P., Rontu, R., Eklund, C., Hutri-Kähönen, N., Taittonen, L., Hurme, M., Viikari, J. S. A., Raitakari, O. T., and Aittokallio, T. (2010). Genetic variants and their interactions in the prediction of increased pre-clinical carotid atherosclerosis: The cardiovascular risk in young Finns study. *PLoS Genetics*, 6(9):e1001146.

Pahikkala, T. (2008). *New Kernel Functions and Learning Methods for Text and Data Mining*. PhD thesis, Turku Centre for Computer Science (TUCS), Turku, Finland.

Pahikkala, T., Airola, A., Boberg, J., and Salakoski, T. (2008a). Exact and efficient leave-pair-out cross-validation for ranking RLS. In Honkela, T., Pöllä, M., Paukkeri, M.-S., and Simula, O., editors, *Proceedings of the*

2nd International and Interdisciplinary Conference on Adaptive Knowledge Representation and Reasoning (AKRR 2008), pages 1–8. Helsinki University of Technology.

Pahikkala, T., Airola, A., Naula, P., and Salakoski, T. (2010a). Greedy RankRLS: a linear time algorithm for learning sparse ranking models. In Gabrilovich, E., Smola, A. J., and Tishby, N., editors, *SIGIR 2010 Workshop on Feature Generation and Selection for Information Retrieval*, pages 11–18. ACM.

Pahikkala, T., Airola, A., and Salakoski, T. (2010b). Feature selection for regularized least-squares: New computational short-cuts and fast algorithmic implementations. In Kaski, S., Miller, D. J., Oja, E., and Honkela, A., editors, *Proceedings of the Twentieth IEEE International Workshop on Machine Learning for Signal Processing (MLSP 2010)*, pages 295–300. IEEE.

Pahikkala, T., Airola, A., and Salakoski, T. (2010c). Speeding up greedy forward selection for regularized least-squares. In Draghici, S., Khoshgoftaar, T. M., Palade, V., Pedrycz, W., Wani, M. A., and Zhu, X., editors, *Proceedings of The Ninth International Conference on Machine Learning and Applications (ICMLA 2010)*, pages 325–330. IEEE.

Pahikkala, T., Airola, A., Suominen, H., Boberg, J., and Salakoski, T. (2008b). Efficient AUC maximization with regularized least-squares. In Holst, A., Kreuger, P., and Funk, P., editors, *Proceedings of the 10th Scandinavian Conference on Artificial Intelligence (SCAI 2008)*, volume 173 of *Frontiers in Artificial Intelligence and Applications*, pages 12–19. IOS Press, Amsterdam, Netherlands.

Pahikkala, T., Airola, A., Xu, T. C., Liljeberg, P., Tenhunen, H., and Salakoski, T. (2011). A parallel online regularized least-squares machine learning algorithm for future multi-core processors. In *Proceedings of the 1st International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS 2011)*, pages 590–599. SciTePress.

Pahikkala, T., Boberg, J., and Salakoski, T. (2006a). Fast n-fold cross-validation for regularized least-squares. In Honkela, T., Raiko, T., Kortela, J., and Valpola, H., editors, *Proceedings of the Ninth Scandinavian Conference on Artificial Intelligence (SCAI 2006)*, pages 83–90, Espoo, Finland. Otamedia Oy.

Pahikkala, T., Suominen, H., Boberg, J., and Salakoski, T. (2009). Efficient hold-out for subset of regressors. In Kolehmainen, M., Toivanen, P., and Beliczynski, B., editors, *Proceedings of the 9th International Conference*

on Adaptive and Natural Computing Algorithms (ICANNGA 2009), volume 5495 of *Lecture Notes in Computer Science*, pages 350–359, Berlin, Germany. Springer.

Pahikkala, T., Tsivtsivadze, E., Airola, A., Boberg, J., and Salakoski, T. (2007). Learning to rank with pairwise regularized least-squares. In Joachims, T., Li, H., Liu, T.-Y., and Zhai, C., editors, *SIGIR 2007 Workshop on Learning to Rank for Information Retrieval*, pages 27–33.

Pahikkala, T., Tsivtsivadze, E., Boberg, J., and Salakoski, T. (2006b). Graph kernels versus graph representations: a case study in parse ranking. In Gärtner, T., Garriga, G. C., and Meinl, T., editors, *Proceedings of the ECML/PKDD 2006 workshop on Mining and Learning with Graphs (MLG 2006)*, Berlin, Germany.

Pahikkala, T., Waegeman, W., Airola, A., Salakoski, T., and De Baets, B. (2010d). Conditional ranking on relational data. In Balcázar, J. L., Bonchi, F., Gionis, A., and Sebag, M., editors, *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases, Part II (ECML PKDD 2010)*, volume 6322 of *Lecture Notes in Computer Science*, pages 499–514. Springer.

Pahikkala, T., Waegeman, W., Tsivtsivadze, E., Salakoski, T., and De Baets, B. (2010e). Learning intransitive reciprocal relations with kernel methods. *European Journal of Operational Research*, 206:676–685.

Parker, B. J., Gunter, S., and Bedo, J. (2007). Stratification bias in low signal microarray studies. *BMC Bioinformatics*, 8:326.

Pedersen, T. (2008). Empiricism is not a matter of faith. *Computational Linguistics*, 34:465–470.

Poggio, T. and Girosi, F. (1990). Networks for approximation and learning. *Proceedings of the IEEE*, 78(9).

Poggio, T. and Smale, S. (2003). The mathematics of learning: Dealing with data. *Notices of the American Mathematical Society (AMS)*, 50(5):537–544.

Provost, F. J., Fawcett, T., and Kohavi, R. (1998). The case against accuracy estimation for comparing induction algorithms. In Shavlik, J., editor, *Proceedings of the Fifteenth International Conference on Machine Learning (ICML 1998)*, pages 445–453, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Pyysalo, S. (2008). *A Dependency Parsing Approach to Biomedical Text Mining*. PhD thesis, Turku Centre for Computer Science (TUCS).

Qin, T., Liu, T.-Y., Xu, J., and Li, H. (2010). Letor: A benchmark collection for research on learning to rank for information retrieval. *Information Retrieval*, 13:346–374.

Qin, T., Zhang, X.-D., Tsai, M.-F., Wang, D.-S., Liu, T.-Y., and Li, H. (2008). Query-level loss functions for information retrieval. *Information Processing and Management*, 44(2):838–855.

Quiñonero-Candela, J. and Rasmussen, C. E. (2005). A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research*, 6:1939–1959.

Rahimi, A. and Recht, B. (2007). Random features for large-scale kernel machines. In Platt, J. C., Koller, D., Singer, Y., Roweis, S. T., Platt, J. C., Koller, D., Singer, Y., and Roweis, S. T., editors, *Advances in Neural Processing Systems 20*. MIT Press.

Rakotomamonjy, A. (2004). Optimizing area under ROC curve with SVMs. In Hernández-Orallo, J., Ferri, C., Lachiche, N., and Flach, P. A., editors, *Proceedings of the 1st International Workshop on ROC Analysis in Artificial Intelligence*, pages 71–80.

Rifkin, R. (2002). *Everything Old Is New Again: A Fresh Look at Historical Approaches in Machine Learning*. PhD thesis, Massachusetts Institute of Technology.

Rifkin, R. and Lippert, R. (2007). Notes on regularized least squares. Technical Report MIT-CSAIL-TR-2007-025, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA.

Rifkin, R., Yeo, G., and Poggio, T. (2003). Regularized least-squares classification. In Suykens, J., Horvath, G., Basu, S., Micchelli, C., and Vandewalle, J., editors, *Advances in Learning Theory: Methods, Model and Applications*, volume 190 of *NATO Science Series III: Computer and System Sciences*, chapter 7, pages 131–154. IOS Press, Amsterdam, Netherlands.

Sætre, R., Sagae, K., and Tsujii, J. (2007). Syntactic features for protein-protein interaction extraction. In *Second International Symposium on Languages in Biology and Medicine (LBM 2007)*.

Saunders, C., Gammerman, A., and Vovk, V. (1998). Ridge regression learning algorithm in dual variables. In Shavlik, J., editor, *Proceedings of the Fifteenth International Conference on Machine Learning (ICML 1998)*, pages 515–521, San Francisco, California, USA. Morgan Kaufmann Publishers Inc.

Schiavo, R. A. and Hand, D. J. (2000). Ten more years of error rate research. *International Statistical Review*, 68(3):295–310.

Schmidt, M. and Lipson, H. (2009). Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85.

Schölkopf, B., Herbrich, R., and Smola, A. J. (2001). A generalized representer theorem. In Helmbold, D. and Williamson, R., editors, *Proceedings of the 14th Annual Conference on Computational Learning Theory and and 5th European Conference on Computational Learning Theory (COLT 2001)*, pages 416–426, Berlin, Germany. Springer.

Schölkopf, B., Mika, S., Burges, C., Knirsch, P., Müller, K.-R., Rätsch, G., and Smola, A. (1999). Input space versus feature space in kernel-based methods. *IEEE Transactions On Neural Networks*, 10(5):1000–1017.

Schölkopf, B. and Smola, A. J. (2002). *Learning with kernels*. MIT Press, Cambridge, Massachusetts, USA.

Schölkopf, B., Weston, J., Eskin, E., Leslie, C., and Noble, W. S. (2002). A kernel approach for learning from almost orthogonal patterns. In Elomaa, T., Mannila, H., and Toivonen, H., editors, *Proceedings of the 13th European Conference on Machine Learning and 6th European Conference on Principles of Data Mining and Knowledge Discovery (ECML/PKDD 2002)*, volume 2430 of *Lecture Notes in Computer Science*, pages 494–511, London, UK, UK. Springer-Verlag.

Sculley, D. (2009). Large scale learning to rank. In *NIPS Workshop: Advances in Ranking*, pages 58–63.

Shashua, A. and Levin, A. (2003). Ranking with large margin principle: Two approaches. In Becker, S., Thrun, S., and Obermayer, K., editors, *Advances in Neural Information Processing Systems 15*. MIT Press, Cambridge, Massachusetts, USA.

Shawe-Taylor, J. and Cristianini, N. (2004). *Kernel Methods for Pattern Analysis*. Cambridge University Press, Cambridge.

Shewchuk, J. R. (1994). An introduction to the conjugate gradient method without the agonizing pain. Technical report.

Shwartz, S. S., Singer, Y., and Srebro, N. (2007). Pegasos: Primal Estimated sub-GrAdient SOlver for SVM. In Ghahramani, Z., editor, *Proceedings of the 24th international conference on Machine learning (ICML 2007)*, pages 807–814, New York, NY, USA. ACM.

Smale, S., Rosasco, L., Bouvrie, J., Caponnetto, A., and Poggio, T. (2010). Mathematics of the neural response. *Foundations of Computational Mathematics*, 10:67–91.

Smola, A. J. and Schölkopf, B. (2000). Sparse greedy matrix approximation for machine learning. In Langley, P., editor, *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)*, pages 911–918, San Francisco, CA. Morgan Kaufmann Publishers Inc.

Smola, A. J., Vishwanathan, S. V. N., and Le, Q. (2007). Bundle methods for machine learning. In McCallum, A., editor, *Advances in Neural Information Processing Systems 20*. MIT Press.

Sonnenburg, S., Braun, M. L., Ong, C. S., Bengio, S., Bottou, L., Holmes, G., Lecun, Y., Müller, K. R., Pereira, F., Rasmussen, C. E., Rätsch, G., Schölkopf, B., Smola, A., Vincent, P., Weston, J., and Williamson, R. (2007). The need for open source software in machine learning. *Journal of Machine Learning Research*, 8:2443–2466.

Suominen, H. (2009). *Machine Learning and Clinical Text: Supporting Health Information Flow*. PhD thesis, Turku Centre for Computer Science (TUCS).

Suominen, H., Pahikkala, T., Hiissa, M., Lehtikunnas, T., Back, B., Karsten, E. H., Salanterä, S., and Salakoski, T. (2006). Relevance ranking of intensive care nursing narratives. In Gabrys, B., Howlett, R., , and Jain, L., editors, *Proceedings of the 10th International Conference on Knowledge-Based & Intelligent Information & Engineering Systems, Part I (KES 2006)*, volume 4251, pages 720–727. Springer.

Suykens, J. A. K. and Vandewalle, J. (1999). Least squares support vector machine classifiers. *Neural Processing Letters*, 9(3):293–300.

Swets, J. A. (1988). Measuring the accuracy of diagnostic systems. *Science*, 240(4857):1285–1293.

Teo, C. H., Smola, A., Vishwanathan, S. V., and Le, Q. V. (2007). A scalable modular convex solver for regularized risk minimization. In Berkhin, P., Caruana, R., Wu, X., and Gaffney, S., editors, *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD 2007)*, pages 727–736, New York, NY, USA. ACM.

Teo, C. H., Vishwanathan, S. V., Smola, A., and Le, Q. V. (2010). Bundle methods for regularized risk minimization. *Journal of Machine Learning Research*, 11:311–365.

Tesauro, G. (1989). Connectionist learning of expert preferences by comparison training. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems 1*, pages 99–106. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Tikk, D., Thomas, P., Palaga, P., Hakenberg, J., and Leser, U. (2010). A comprehensive benchmark of kernel methods to extract protein-protein interactions from literature. *PLoS Computational Biology*, 6(7):e1000837+.

Tsivtsivadze, E. (2009). *Learning Preferences with Kernel-Based Methods*. PhD thesis, Turku Centre for Computer Science (TUCS).

Tsivtsivadze, E., Pahikkala, T., Airola, A., Boberg, J., and Salakoski, T. (2008). A sparse regularized least-squares preference learning algorithm. In Holst, A., Kreuger, P., and Funk, P., editors, *Tenth Scandinavian Conference on Artificial Intelligence (SCAI 2008)*, volume 173 of *Frontiers in Artificial Intelligence and Applications*, pages 76–83, Amsterdam, Netherlands. IOS Press.

Tsivtsivadze, E., Pahikkala, T., Boberg, J., Salakoski, T., and Heskes, T. (2011). Co-regularized least-squares for label ranking. In Fürnkranz, J. and Hüllermeier, E., editors, *Preference Learning*, pages 107–123. Springer.

Tsivtsivadze, E., Pahikkala, T., Pyysalo, S., Boberg, J., Mylläri, A., and Salakoski, T. (2005). Regularized least-squares for parse ranking. In Famili, A. F., Kok, J. N., Pena, J. M., Siebes, A., and Feelders, A. J., editors, *Proceedings of the 6th International Symposium on Intelligent Data Analysis (IDA 2005)*, volume 3646, pages 464–474. Springer.

Tsochantaridis, I., Joachims, T., Hofmann, T., and Altun, Y. (2005). Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484.

Tsuda, K. (1999). Support vector classifier with asymmetric kernel functions. In *European Symposium on Artificial Neural Networks (ESANN 1999)*, pages 183–188.

Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, LIX.

Vanderlooy, S. and Hüllermeier, E. (2008). A critical analysis of variants of the AUC. *Machine Learning*, 72(3):247–262.

Vapnik, V. (1979). *Estimation of Dependences Based on Empirical Data [in Russian]*. Nauka, Moscow. (English translation: Springer, New York, 1982).

Vapnik, V. N. (1995). *The nature of statistical learning theory*. Springer-Verlag New York, Inc., New York, NY, USA.

Vapnik, V. N. (1998). *Statistical Learning Theory*. Wiley-Interscience.

Waegeman, W., De Baets, B., and Boullart, L. (2008). ROC analysis in ordinal regression learning. *Pattern Recognition Letters*, 29(1):1–9.

Williams, C. K. and Rasmussen, C. E. (1996). Gaussian processes for regression. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., editors, *Advances in Neural Information Processing Systems 8*, pages 514–520. MIT Press.

Williams, C. K. I. and Seeger, M. (2001). Using the Nyström method to speed up kernel machines. In Leen, T. K., Dietterich, T. G., and Tresp, V., editors, *Advances in Neural Information Processing Systems 13*, pages 682–688. MIT Press.

Xia, F., Liu, T.-Y., Wang, J., Zhang, W., and Li, H. (2008). Listwise approach to learning to rank: theory and algorithm. In Cohen, W. W., McCallum, A., and Roweis, S. T., editors, *Proceedings of the 25th international conference on Machine learning (ICML 2008)*, pages 1192–1199, New York, NY, USA. ACM.

Yue, Y., Finley, T., Radlinski, F., and Joachims, T. (2007). A support vector method for optimizing average precision. In Kraaij, W., de Vries, A. P., Clarke, C. L. A., Fuhr, N., and Kando, N., editors, *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR 2007)*, pages 271–278, New York, NY, USA. ACM.

Zhang, K., Tsang, I. W., and Kwok, J. T. (2008). Improved Nyström low-rank approximation and error analysis. In Cohen, W. W., McCallum, A., and Roweis, S. T., editors, *Proceedings of the 25th international conference on Machine learning (ICML 2008)*, pages 1232–1239, New York, NY, USA. ACM.

Zhang, P. and Peng, J. (2004). SVM vs regularized least squares classification. In Kittler, J., Petrou, M., and Nixon, M., editors, *Proceedings of the 17th International Conference on Pattern Recognition (ICPR 2004)*, pages 176–179, Washington, DC, USA. IEEE Computer Society.

Zweigenbaum, P., Demner-Fushman, D., Yu, H., and Cohen, K. B. (2007). Frontiers of biomedical text mining: current progress. *Briefings in Bioinformatics*, 8(5):358–375.

Publication Reprints

Paper I

An efficient algorithm for learning to rank from preference graphs

Pahikkala, T., Tsivtsivadze, E., Airola, A., Järvinen, J., and Boberg, J. (2009). *Machine Learning*, 75(1):129–165.

Mach Learn (2009) 75: 129–165
DOI 10.1007/s10994-008-5097-z

An efficient algorithm for learning to rank from preference graphs

Tapio Pahikkala · Evgeni Tsivtsivadze · Antti Airola · Jouni Järvinen · Jorma Boberg

Received: 9 March 2007 / Revised: 5 December 2008 / Accepted: 10 December 2008 / Published online: 9 January 2009

Abstract In this paper, we introduce a framework for regularized least-squares (RLS) type of ranking cost functions and we propose three such cost functions. Further, we propose a kernel-based preference learning algorithm, which we call RankRLS, for minimizing these functions. It is shown that RankRLS has many computational advantages compared to the ranking algorithms that are based on minimizing other types of costs, such as the hinge cost. In particular, we present efficient algorithms for training, parameter selection, multiple output learning, cross-validation, and large-scale learning. Circumstances under which these computational benefits make RankRLS preferable to RankSVM are considered. We evaluate RankRLS on four different types of ranking tasks using RankSVM and the standard RLS regression as the baselines. RankRLS outperforms the standard RLS regression and its performance is very similar to that of RankSVM, while RankRLS has several computational benefits over RankSVM.

Keywords Ranking · Preference learning · Preference graph · Regularized least-squares · Kernel methods

Editors: Thomas Gärtner and Gemma C. Garriga.

T. Pahikkala (✉) · E. Tsivtsivadze · A. Airola · J. Järvinen · J. Boberg
Turku Centre for Computer Science (TUCS), Department of Information Technology, University of Turku, Joukahaisenkatu 3-5 B, 20520 Turku, Finland
e-mail: tapio.pahikkala@utu.fi

E. Tsivtsivadze
e-mail: evgeni.tsivtsivadze@utu.fi

A. Airola
e-mail: antti.airola@utu.fi

J. Järvinen
e-mail: jouni.jarvinen@utu.fi

J. Boberg
e-mail: jorma.boberg@utu.fi

1 Introduction

Preference learning has recently received a lot of attention in the machine learning literature—we refer to Fürnkranz and Hüllermeier (2005) for a compact and illuminating summary of the preference learning tasks. Preference learning can be considered as a task in which the aim is to learn a function capable of arranging data points according to a given preference relation. When comparing two data points, the function is able to evaluate whether the first point is preferred over the second one.

We assume that we are given a training data consisting of input data points and their pairwise preferences that are used to train a supervised learning algorithm for the prediction of the preference relations among unseen data points. We also consider the scoring setting in which we are given a training data consisting of scored data points, that is, each input data point is assigned a real valued score indicating its goodness. The pairwise preference between these data points are then determined by the differences of the scores. This type of preference learning tasks are often cast into classification tasks so that each pair of data points, in which one point is preferred over the other, is used as a training data point whose class indicates the direction of the preference (for recent in depth theoretical analysis of ranking algorithms see, e.g. Clémençon et al. 2005; Agarwal 2006; Cortes et al. 2007b). For example, Herbrich et al. (1999) used this approach together with support vector machines (SVM) for ordinal regression tasks. This method is often referred to as RankSVM. A similar SVM adaptation was made by Joachims (2002) to rerank the results obtained from a search engine.

Recently, it has been shown that the RLS classifiers (see e.g. Rifkin 2002), also known as the least-squares SVMs (Suykens and Vandewalle 1999), have a classification performance similar to the regular SVMs (see e.g. Rifkin 2002; Gestel et al. 2004; Zhang and Peng 2004). In Pahikkala et al. (2007b), we proposed RankRLS, an algorithm that learns preferences from scored data, in which for each input data point x we have a real value score s attached. If an input data point x is preferred over x', the difference to be regressed is $s - s'$, where s and s' are the scores of x and x', respectively. It was shown that the computational complexity of training RankRLS is equal to the complexity of training an RLS regressor for the same data set. Namely, the computational complexity of training RankRLS was shown to be $O(m^3)$ in the dual form, where m is the number of input data points in the training data, and $O(mn^2 + n^3)$ in the primal form, where n is the dimensionality of the feature space. A similar algorithm with equal computational times was at the same time independently proposed by Cortes et al. (2007a, 2007b) and called MPRank. They also provide a thorough theoretical analysis of the generalization error of the MPRank algorithm using stability bounds. The difference between RankRLS and MPRank is that the former includes in the training process only such input data point pairs that are relevant to the task in question, while the latter is defined to include every possible input pair. For example, in many document retrieval tasks, each input data point consists of a query and a document, and hence there should be no preferences between such inputs that are associated to different queries. Otherwise, the objective functions of RankRLS and MPRank are the same, but the closed form solution are derived in a slightly different way.

In this paper, we extend our consideration of RankRLS so that it can be used to learn not only from scored data, but also from a given sequence of pairwise preferences and their magnitudes when the scores are not given. Moreover, we introduce a general framework for RLS based ranking cost functions and propose three different specializations for it. Note that we only consider the case in which we learn so-called scoring function that maps each possible input to a real value. The function then induces a total order for the inputs. The direction of preference between two inputs is obtained by comparing their predicted scores.

Cost functions that are variations of the least-squares cost have certain computational advantages compared to the other types of costs, such as the hinge cost used with SVM. For example, the least-squares-based learning methods can be expressed using matrix calculus which makes them simple to implement and analyze. Moreover, RankRLS can be trained so that its computational complexity depends on the number of data points instead of the number of observed pairwise preferences. This is an important advantage, because the number of preferences is usually proportional to the square of the number of individual data points. Furthermore, there often exists efficient shortcut methods for calculating the cross-validation performance for the least-squares based learners and for parameter selection (see e.g. Pahikkala et al. 2006a, 2007a, 2008a; Rifkin and Lippert 2007a). RLS-based learning algorithms can also be extended for large-scale learning using the subset of regressors method (see e.g. Quiñonero-Candela et al. 2007; Tsivtsivadze et al. 2008). Further advantages include the possibility to learn several functions in parallel as considered by Rifkin and Klautau (2004). In this paper, we present efficient algorithms for training RankRLS both in small and large scale as well as for both linear and nonlinear learning tasks. We also present fast algorithms for cross-validation, parameter selection, and multiple output learning.

We also make a thorough consideration of the circumstances under which the fast cross-validation, parameter selection, and multiple output learning algorithms make RankRLS preferable to RankSVM from computational complexity point of view. Namely, the benefits and drawbacks of RankRLS and RankSVM in both small-scale and large-scale learning tasks are investigated and so are both the linear and nonlinear learning problems. For example, training a single instance of a RankSVM may be faster than training a single instance of RankRLS in the linear learning tasks, but the efficient cross-validation, parameter selection, and multiple output learning algorithms make RankRLS in many situations much faster method to use than RankSVM. This is especially the case if nonlinear kernel functions are used and if cross-validation is used for performance estimation.

In our experiments, we test our ranking algorithms with different tasks. The pairwise preferences in all of the tasks are induced by a scoring of the data points. Two of the considered tasks are the ranking of dependency parses and document retrieval. Comparing parses generated for different sentences or documents returned for different queries would, of course, make little sense. This kind of preference structure is typical for label ranking problems in which the aim is to rank for each object a set of labels, and this is the approach we use for these tasks. In the former task, the objects and labels are sentences and their parse candidates, and in the latter they are queries and documents retrieved by them. The other two tasks considered are document classification and collaborative filtering which we consider as object ranking problems in which the aim is to learn a given preference structure over all data points. Further, the document retrieval and binary classification tasks are bipartite ranking problems, that is, there are only two possible score values for the data points. In contrast, the parse ranking and collaborative filtering tasks have real-valued scores.

In the label ranking tasks, we test whether some of the input pairs that are not relevant to the learning problem to be solved would be beneficial if included in the training process. Our results suggest that this is not the case. Moreover, we compare the three proposed cost functions on the ranking tasks. In all experiments, we use as baselines the RankSVM method and the ordinary RLS regressor trained to regress the scores of the data points. The ranking for the data points is then obtained from the regressed scores. We observe that performances of RankRLS and RankSVM are very similar in all considered tasks, with no statistically significant differences observed, although RankRLS has many computational benefits over RankSVM as discussed in the paper. RankRLS and RankSVM perform better or as well as the RLS regressor.

This paper is organized as follows. In Sect. 2, we present a formal introduction on the preference learning tasks under consideration, and define the proposed method RankRLS. Section 3 considers computationally efficient algorithms for training and validation. We summarize the computational benefits of RankRLS and compare them to those of RankSVM in Sect. 4. RankRLS is experimentally evaluated in Sect. 5. We conclude the paper in Sect. 6.

2 The RankRLS algorithm

First, we give a formulation of preference learning problems in Sect. 2.1. Section 2.2 concerns the framework of regularized kernel methods. In Sect. 2.3, we introduce the RankRLS algorithm. Finally, we consider three different variations of the least-squares ranking cost function in Sect. 2.4.

2.1 Preference learning

Let $\mathcal{X}$, called the input space, denote the set of input data points which we call in the following shortly as inputs. We assume that we are given a sequence

$$X = (x_1, \ldots, x_m) \in (\mathcal{X}^m)^{\mathrm{T}}$$

of inputs. Moreover, let

$$E = (e_1, \ldots, e_l)^{\mathrm{T}} \in (\mathcal{X} \times \mathcal{X} \times \mathbb{R}^+)^l$$

be a sequence of observed preferences between the inputs, that is, $e_i = (x_h, x_j, y_i)$, where $1 \leq h, j \leq m$ and $h \neq j$, indicating that x_h is preferred over x_j with magnitude y_i. The magnitudes may be supplied in the training data, or in case such information is not available, magnitude 1 can be given for each pairwise preference. Altogether, we define the training data to be the tuple

$$G = (X, E).$$

G can be considered as a preference graph in which the inputs x_h, are the vertices and e_i are the edges. By the definition of E, there may be multiple observed preferences with possibly differing magnitudes between two inputs. Thus, G is a multigraph. Finally, we define $\mathcal{G}$ to be the set of all possible graphs of the above defined type.

We also consider a special case of preference learning setting in which E is not given but so-called scoring information of the inputs is available and the preferences are determined by this scoring. Thus, in the scoring setting we assume that we are given a sequence

$$S = (s_1, \ldots, s_m)^{\mathrm{T}} \in \mathbb{R}^m$$

of real valued scores corresponding to the input sequence $X = (x_1, \ldots, x_m) \in (\mathcal{X}^m)^{\mathrm{T}}$. In this case, we can obtain a sequence of observed preferences so that for each pair of inputs x_h and x_j, where $1 \leq h, j \leq m$ and $h \neq j$, there exists an edge $e_i = (x_h, x_j, y_i)$, where $y_i = s_h - s_j$, if and only if $s_h > s_j$. Unless stated otherwise, we do not assume that the observed preferences are determined by a scoring information.

Fürnkranz and Hüllermeier (2005) divided the preference learning tasks into two categories, namely, to the tasks of learning object preferences and learning label preferences. Here, we consider a similar type of division. The cases in which the aim is to learn a given

preference structure over all inputs are considered as object ranking problems. For example, any binary classification task can be considered as an object ranking task in which the aim is to rank all inputs belonging to the positive class above the ones belonging to the negative class. We define label ranking tasks to be such in which the inputs are comprised of an object and its label. In this case, the observed preferences make sense, that is, are relevant only between such inputs that are associated with the same object. In many document retrieval tasks, for example, each input consists of a query and a document. In this case, the documents can be considered as the labels of the queries. Clearly, there should be no preferences between such inputs that are associated to different queries. If the preferences of a label ranking task are induced by a scoring of the inputs, the preferences between inputs associated to different objects are considered to be irrelevant to the task in question and they are not included in the preference graph. Formally, the sequence of preferences is obtained from the scoring so that for each pair of inputs x_h and x_j, where $1 \leq h, j \leq m$ and $h \neq j$, there exists an edge $e_i = (x_h, x_j, y_i)$, where $y_i = s_h - s_j$, if and only if $s_h > s_j$ and the preference is relevant to the task under consideration. In our notation, we make no difference between the object and label ranking settings, and we assume that the sequence of observed preferences is formed according to the setting in question.

Let $\mathbb{R}^{\mathcal{X}}$ denote the set of all functions from $\mathcal{X}$ to $\mathbb{R}$, and let $\mathcal{H} \subseteq \mathbb{R}^{\mathcal{X}}$ be the hypothesis space. A natural way to measure how well a function $f \in \mathcal{H}$ agrees with preferences of E is to define a disagreement error:

$$D(f, G) = \frac{1}{2l} \sum_{i=1}^{l} (1 - \text{sign}(g(e_i))), \tag{1}$$

where $e_i = (x_h, x_j, y_i)$ for some $h \neq j$, $1 \leq h, j \leq m$,

$$g(e_i) = f(x_h) - f(x_j),$$

and sign is the signum function

$$\text{sign}(r) = \begin{cases} 1 & \text{if } r > 0, \\ -1 & \text{if } r \leq 0. \end{cases}$$

Note that in the disagreement error (1), we omit the magnitudes y_i. Nevertheless, we take advantage of the magnitude information when we design the learning algorithms.

Training can be considered as a process of selecting a function from the hypothesis space that best performs the learning task in question. Thus, learning can be viewed as an algorithm $\mathcal{A}$ that for a given preference graph G selects an appropriate function f from $\mathcal{H}$. Formally,

$$\mathcal{A} : \mathcal{G} \to \mathcal{H}. \tag{2}$$

2.2 Regularized kernel methods

Here, we consider the selection of the suitable function $f \in \mathcal{H}$. Let $\mathcal{X}$ denote the input space, which can be any set, and $\mathcal{F}$ denote an inner product space we call the feature space. For any mapping

$$\Phi : \mathcal{X} \to \mathcal{F},$$

the inner product

$$k(x, x') = \langle \Phi(x), \Phi(x') \rangle$$

of the mapped inputs is called a kernel function. We define the symmetric kernel matrix $K \in \mathbb{R}^{m \times m}$, where $\mathbb{R}^{m \times m}$ denotes the set of real $m \times m$-matrices, as

$$K = \begin{pmatrix} k(x_1, x_1) & \cdots & k(x_1, x_m) \\ \vdots & \ddots & \vdots \\ k(x_m, x_1) & \cdots & k(x_m, x_m) \end{pmatrix}$$

for the sequence $X = (x_1, \ldots, x_m) \in (\mathcal{X}^m)^{\mathrm{T}}$ of inputs. Unless stated otherwise, we assume that the kernel matrix is strictly positive definite, that is, $A^{\mathrm{T}} K A > 0$ for all $A \in \mathbb{R}^m$, $A \neq 0$. The strict positive definiteness of the kernel matrix K can be ensured, for example, by adding ϵI to K, where $I \in \mathbb{R}^{m \times m}$ is the identity matrix and ϵ is a small positive real number.

Following Schölkopf et al. (2001), we define the reproducing kernel Hilbert space (RKHS) determined by the input space $\mathcal{X}$ and the kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ as

$$\mathcal{H}_{k,\mathcal{X}} = \left\{ f \in \mathbb{R}^{\mathcal{X}} \;\middle|\; f(x) = \sum_{i=1}^{\infty} \beta_i k(x, x_i), \beta_i \in \mathbb{R}, x_i \in \mathcal{X}, \|f\|_k < \infty \right\},$$

where

$$\|f\|_k = \sqrt{\sum_{i,j=1}^{\infty} \beta_i \beta_j k(x_i, x_j)}$$

denotes the norm of the function f in the RKHS. Using $\mathcal{H}_{k,\mathcal{X}}$ as our hypothesis space, we next define the cost functions that we can use to measure how well the hypotheses fit our training data G. Overloading our notation, we denote

$$f(X) = (f(x_1), \ldots, f(x_m))^{\mathrm{T}}$$

for the sequence X of inputs and a hypothesis $f \in \mathcal{H}_{k,\mathcal{X}}$. We use cost functions of type

$$c : \mathbb{R}^m \times \mathcal{G} \to \mathbb{R}$$

to assign a value

$$c(f(X), G) \tag{3}$$

on the predictions $f(X)$, training data $G = (X, E)$, and a candidate hypothesis $f \in \mathcal{H}_{k,\mathcal{X}}$ that measures how well f fits G.

We now consider the following variational problem as a realization of algorithm (2) that we use to select an appropriate hypothesis f from $\mathcal{H}_{k,\mathcal{X}}$ for training data G. Namely, we consider an algorithm

$$\mathcal{A}(G) = \operatorname*{argmin}_{f \in \mathcal{H}_{k,\mathcal{X}}} J(f, G), \tag{4}$$

where

$$J(f, G) = c(f(X), G) + \lambda \|f\|_k^2 \tag{5}$$

and $\lambda > 0$ is a regularization parameter. The first term measures the performance of a candidate hypothesis on the training data and the second term, called the regularizer, measures the complexity of the hypothesis with the RKHS norm.

According to the representer theorem (Schölkopf et al. 2001), any minimizer $f \in \mathcal{H}_{k,\mathcal{X}}$ of (5) admits the representation of the following form:

$$f(x) = \sum_{i=1}^{m} a_i k(x, x_i), \tag{6}$$

where $a_i \in \mathbb{R}$ and k is the kernel function associated with the RKHS mentioned above. Let

$$A = (a_1, \ldots, a_m)^{\mathrm{T}} \in \mathbb{R}^m$$

be a vector consisting of the values that determine the solution (6). By overloading our notation, we write $k(x, X) = (k(x, x_1), \ldots, k(x, x_m)) \in (\mathbb{R}^m)^{\mathrm{T}}$, where $x \in \mathcal{X}$ and $X = (x_1, \ldots, x_m) \in (\mathcal{X}^m)^{\mathrm{T}}$. Using this type of matrix notation, we can write

$$f(x) = \sum_{i=1}^{m} a_i k(x, x_i) = k(x, X)A. \tag{7}$$

Similarly, the column vector $f(X) \in \mathbb{R}^m$, that contains the predictions for the inputs obtained with the function f, is

$$f(X) = \begin{pmatrix} f(x_1) \\ \vdots \\ f(x_m) \end{pmatrix} = \begin{pmatrix} k(x_1, X)A \\ \vdots \\ k(x_m, X)A \end{pmatrix} = KA. \tag{8}$$

Further, according to (6) and to the definition of the RKHS norm, the regularizer can be written as

$$\lambda \|f\|_k^2 = \lambda \sum_{i,j=1}^{m} a_i a_j k(x_i, x_j) = \lambda A^{\mathrm{T}} K A. \tag{9}$$

Next, we consider realizations for the cost function.

2.3 Regularized least-squares regression of preferences

In order to construct a regularized kernel method that would learn the preferences defined on the training data $G = (X, E)$, we have to define an appropriate cost function. A natural way to encode the preference information into a cost function is to use the disagreement error (1) for the preferences:

$$c(f(X), G) = \sum_{i=1}^{l} (1 - \operatorname{sign}(g(e_i))), \tag{10}$$

where $e_i = (x_h, x_j, y_i)$ and $g(e_i) = f(x_h) - f(x_j)$. Note that in (1), $\frac{1}{2l}$ can be considered as a constant, and hence it can be omitted from (10). It is well-known that the use of this type of cost functions leads to intractable optimization problems. Therefore, instead of using (10),

we use functions approximating it. We adopt an approach similar to the regularized least-squares classification (Rifkin 2002) which has been shown to have a performance similar to that of the support vector machine classifiers. That is, we use the following type of square cost as an approximation of (10):

$$c(f(X), G) = \sum_{i=1}^{l} w_i^2 (z_i - g(e_i))^2, \tag{11}$$

where $z_i, w_i \geq 0$ are real-valued parameters. When defining the actual cost functions, we fix the parameters z_i and w_i to be constants or dependent on the magnitude y_i. We observe that the cost is a sum of parabolae whose zeros and widths are determined by z_i and w_i, respectively. Intuitively, the parameters w_i can be considered as importance weights of the edges e_i, since the cost function (11) is more sensitive to the predictions $g(e_i)$ of the edges having a large value of w_i than to those having a small value. In Sect. 2.4, we present three different specializations of the cost function by setting the parameters.

Before presenting the actual learning algorithm, we introduce some notation. Let $M \in \mathbb{R}^{m \times l}$ be a matrix whose rows and columns are indexed by the vertices and edges of the preference graph, respectively, and its entries are given by

$$M_{h,i} = \begin{cases} w_i & \text{if } e_i = (x_h, x_j, y_i) \text{ for some } j \neq h, \\ -w_i & \text{if } e_i = (x_j, x_h, y_i) \text{ for some } j \neq h, \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

In the graph theory (see e.g. Brualdi and Ryser 1991), the matrix M is sometimes called the oriented incidence matrix of a graph and the product $L = MM^{\mathrm{T}}$ is called the Laplacian matrix of the graph. We also note that Laplacian matrix is always positive semidefinite, since it is a product of a real-valued matrix with its own transpose. We consider M as a linear transformation from $\mathbb{R}^m$ to $\mathbb{R}^l$, that is, it can be used to map the vector $f(X)$ consisting of the values $f(x_h)$, $1 \leq h \leq m$, to a vector $M^{\mathrm{T}} f(X)$ containing the values $w_i g(e_i)$, $1 \leq i \leq l$. Further, let us write

$$N = (w_1 z_1, \ldots, w_l z_l)^{\mathrm{T}}. \tag{13}$$

Using these notations, we can rewrite the cost function (11) in a matrix form as

$$c(f(X), G) = (N - M^{\mathrm{T}} f(X))^{\mathrm{T}} (N - M^{\mathrm{T}} f(X)). \tag{14}$$

The next theorem characterizes a method called RankRLS.

Theorem 1 *Let $G = (X, E)$ and let*

$$\mathcal{A}(G) = \operatorname*{argmin}_{f \in \mathcal{H}_{k,\mathcal{X}}} J(f, G), \tag{15}$$

where

$$J(f, G) = \sum_{i=1}^{l} w_i^2 (z_i - g(e_i))^2 + \lambda \| f \|_k^2, \tag{16}$$

is the algorithm under consideration. A coefficient vector $A \in \mathbb{R}^m$ that determines a minimizer of (16) is

$$A = (MM^T K + \lambda I)^{-1} MN. \tag{17}$$

Proof According to the representer theorem, the minimizer of (16) is of the form (6), that is, the problem of finding the optimal hypothesis can be solved by finding the coefficients a_h, $1 \leq h \leq m$.

According to (8), the vector consisting of the input predictions can be written as $f(X) = KA$. We use the matrix M to transform the input predictions to a vector of prediction differences $M^{\mathrm{T}}KA$. Then, the ith entry of the vector $M^{\mathrm{T}}KA$ contains the value $w_i g(e_i)$. We use the matrix forms (9) and (14) to rewrite the algorithm (15) as follows:

$$\mathcal{A}(G) = \operatorname*{argmin}_{A \in \mathbb{R}^m} J(A, G),$$

where

$$J(A, G) = (N - M^{\mathrm{T}}KA)^{\mathrm{T}}(N - M^{\mathrm{T}}KA) + \lambda A^{\mathrm{T}}KA.$$

We take the derivative of $J(A, G)$ with respect to A:

$$\begin{aligned}\frac{d}{dA}J(A, G) &= -2KM(N - M^{\mathrm{T}}KA) + 2\lambda KA \\ &= -2KMN + 2(KMM^{\mathrm{T}}K + \lambda K)A.\end{aligned}$$

We set the derivative to zero and solve with respect to A:

$$\begin{aligned}A &= (KMM^{\mathrm{T}}K + \lambda K)^{-1}KMN \\ &= (MM^{\mathrm{T}}K + \lambda I)^{-1}MN,\end{aligned}$$

where the last equality follows from our assumption of the kernel matrix being strictly positive definite. □

We refer to (17) as the dual solution of RankRLS in contrast to the primal solution considered in Sect. 3.1.

The multiplication of M with N can be performed in $O(l)$ time, since M contains only $2l$ nonzero elements. This is also the complexity of calculating the Laplacian matrix $L = MM^{\mathrm{T}}$ of the preference graph as can be shown in the following way. First, we note (see e.g. Brualdi and Ryser 1991) that, if $h \neq j$, $L_{h,j} = -\sum_i w_i^2$, where i goes through the indices of the edges that are between the hth and jth vertex. Further, $L_{h,h} = \sum_i w_i^2$, where i goes through the indices of the edges starting from or ending to the hth vertex. Therefore, for constructing the matrix L, four operations per edge are needed. For example, an edge starting from the h vertex and ending to the jth vertex affects the entries $L_{h,h}$, $L_{h,j}$, $L_{j,h}$, and $L_{j,j}$. Thus, we have:

$$\text{the complexity of calculating the products } MM^{\mathrm{T}} \text{ and } MN \text{ is } O(l). \tag{18}$$

The subsequent multiplication of L with K and the inversion of the matrix $MM^{\mathrm{T}}K + \lambda I$ can be done in $O(m^3)$ time. Note that, in the time complexities considered in this paper, we do not count the complexity of calculating the kernel matrix, because it depends on the kernel function used. Thus, provided that the kernel matrix is already calculated,

$$\text{the complexity of dual RankRLS training is } O(m^3 + l). \tag{19}$$

Note that the preference graph determined by the sequence of observed preferences E is a multigraph, and hence the number l of the pairwise preferences may not necessarily be

dominated by the term m^3 in (19). However, in the scoring setting, which we discuss more in Sect. 3, we have $l = O(m^2)$, because the number of edges is bounded above by the number of all possible input pairs.

2.4 Specializations of the cost function

We now consider three different specializations of the least-squares cost function (11) for approximating the disagreement error. The variations are depicted in Fig. 1. In the first version, we just set $z_i = 1$ and $w_i = 1$ for all $1 \leq i \leq l$:

$$c(f(X), G) = \sum_{i=1}^{l} (1 - g(e_i))^2. \tag{20}$$

This is a cost function that, similarly to the disagreement error (1), simply ignores the preference magnitudes treating every input pair in E as if their magnitudes would be equal to one.

The second approach uses the magnitude information to determine the zero points, that is, we set $z_i = y_i$ and $w_i = 1$ for all $1 \leq i \leq l$:

$$c(f(X), G) = \sum_{i=1}^{l} (y_i - g(e_i))^2. \tag{21}$$

This cost function is equal to the one proposed by us in Pahikkala et al. (2007b). It has many computational advantages in case preference magnitudes are induced by a scoring of

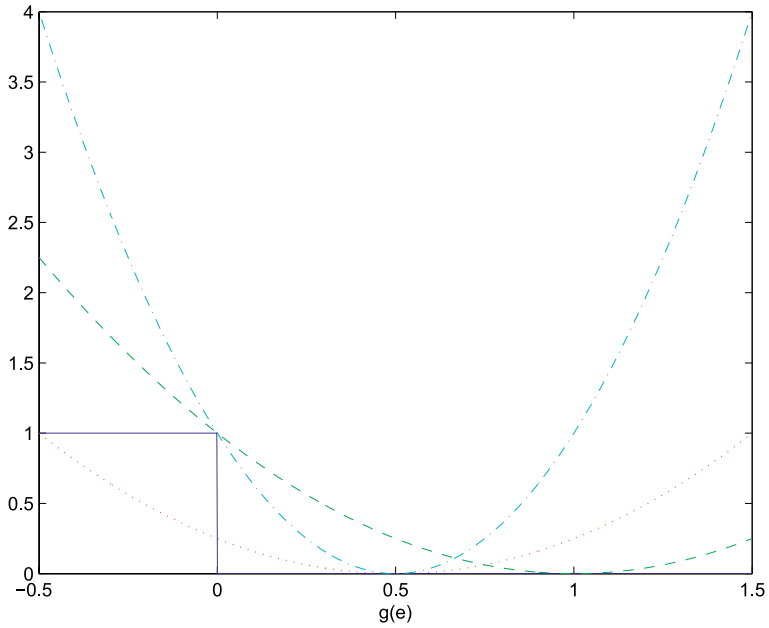

Fig. 1 The x-axis represents the value of g and the y-axis the value of the cost functions when the preference magnitude is 0.5. The disagreement cost is depicted with a *solid line* and the functions (20), (21), and (22) with '–', '. . .', and '-·-', respectively

the inputs as discussed in Sect. 3. A disadvantage of this cost is that it does not form an upper bound on the disagreement error, and therefore this cost function is harder to analyze theoretically.

The third cost function also uses the magnitude information to determine the zero points, and thus we set $z_i = y_i$. In addition, the width parameters are set to $w_i = 1/y_i$ which ensures that the disagreement error is bounded above by this cost function:

$$c(f(X), G) = \sum_{i=1}^{l} \frac{1}{y_i^2}(y_i - g(e_i))^2. \tag{22}$$

Moreover, this cost can be intuitively justified so that the RankRLS algorithm is allowed to make larger prediction errors for edges having a large magnitude than for edges having a small magnitude. This is, because even a small prediction error can reverse the direction of preference for an edge with a small magnitude but not the with a large one.

We observe that the functions (20), (21), and (22) are equal if we have only preferences but not magnitudes, that is, $y_i = 1$ for all $1 \leq i \leq l$. This is the case especially in bipartite ranking, that is, when the preferences are induced by the scoring in which there are only two different scores, say 1 and 0. If $y_i = 1$ for all $1 \leq i \leq l$, also the function (21) forms an upper bound on the disagreement error. Therefore, one may derive results similar to those of Agarwal and Niyogi (2005) and Cortes et al. (2007b) to analyze the generalization performance of the ranking algorithms.

The possibility to give importance weights to the preferences with the parameters w_i also enables the design of cost functions that are more suitable for label ranking tasks than those using only the magnitudes. Consider, for example, the task of parse ranking in which the aim is to rank for each sentence the set of parses according to some preference criterion. The parse candidate sets can be of different size for each sentence, while each sentence is equally important. In this case, it may be beneficial to use a normalized version of the cost function in which each edge associated to a sentence has a weight equal to the inverse of the number of edges associated to the same sentence.

3 Efficient implementations

In this section, we consider ways to speed up the training process of RankRLS. We pay special attention to the preference learning task in the scoring setting using the magnitude preserving cost function (21). In the scoring setting, the inputs $X = (x_1, \ldots, x_m) \in (\mathcal{X}^m)^{\mathrm{T}}$ and the corresponding scores $S = (s_1, \ldots, s_m)^{\mathrm{T}} \in \mathbb{R}^m$ are given. Recall the definitions (12) and (13) of M and N, respectively. We observe, that in case we use (21), the following equation holds:

$$N = M^{\mathrm{T}} S. \tag{23}$$

Therefore, the matrix form (14) can be rewritten as

$$c(f(X), G) = (S - f(X))^{\mathrm{T}} M M^{\mathrm{T}} (S - f(X)). \tag{24}$$

Note that we use the notation $c(f(X), G)$, while we also have the sequence of scores S. Further, while considering the efficient implementations with the cost function (21) in the scoring setting, we also allow preferences with magnitude zero. Namely, we consider cases in which the scoring S induces exactly one preference between every input x_h and x_j, where

$h \neq j$, even if x_h and x_j have equal scores. If the scores are equal, the direction of the edge does not matter, and hence only one edge is needed between the corresponding vertices in the preference graph also in this case. These preferences with zero magnitude are generated only for efficiency reasons and they are ignored when the performance of a ranking algorithm is measured with the disagreement error.

When there are preferences between every input, we can take advantage of the regularities of the matrix M in order to speed up the computations. We first consider the case in which every possible preference induced by the scoring is relevant to the task in question, as is often the case in object ranking tasks. We observe that we can write

$$MM^{\mathrm{T}} = D - PP^{\mathrm{T}}, \tag{25}$$

where $D \in \mathbb{R}^{m \times m}$ is a diagonal matrix whose every diagonal entry is equal to m, and $P \in \mathbb{R}^m$ is a vector whose every entry is equal to 1. It is much more efficient to perform matrix multiplications with the form $D - PP^{\mathrm{T}}$ than with MM^{T}, because P is an m-dimensional vector and D is a diagonal matrix, and thus having only m nonzero elements.

All preferences induced by the scoring are not always relevant to the task in question. In label ranking tasks, for example, we may want to exclude the preferences that are not relevant to the task, that is, the input pairs in which the inputs are associated to different objects. Next, we consider the removal of this type of irrelevant preferences. Assume there are q objects in the training data and each of the m inputs are associated to one of the objects. In the task of parse ranking, for example, an object is a sentence and an input is comprised of a sentence and a parse candidate generated for the sentence. Each parse is associated only with the sentence it is generated for and the aim is to learn to rank the parses for each sentence separately. The scoring induces preferences also between parses that are associated with different sentences but they are considered to be irrelevant to the task of parse ranking. We redefine $P \in \mathbb{R}^{m \times q}$ to be a matrix whose rows are indexed by the inputs and the columns are indexed by the q objects. The value of $P_{i,j}$ is defined to be 1 if the ith input is associated with the jth object and 0 otherwise. Further, we redefine D to be a diagonal matrix whose entries are defined as follows. If the ith input is associated to a certain object, then the ith diagonal element of D is the number of inputs that are associated to the same object. For example, assume that our training data consists of altogether 5 inputs and two objects. The first object is associated with the first two inputs and the second object with the last three inputs. Then, the matrices P and D are

$$P = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{pmatrix} \quad \text{and} \quad D = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 3 \end{pmatrix},$$

respectively. Now, we observe that MM^{T} can again be written as in (25). Similarly to the object ranking case, the matrix P contains only m nonzero elements, and hence the matrix multiplications involving P are as efficient to compute as with the object ranking case.

Further, provided that (23) and (25) hold, solving the dual form (17) involves calculating

$$MM^{\mathrm{T}}K = DK - PP^{\mathrm{T}}K$$

and

$$\begin{aligned} MN &= MM^{\mathrm{T}}S \\ &= DS - PP^{\mathrm{T}}S. \end{aligned}$$

These can be done in $O(m^2)$ and $O(m)$ time, respectively, because there are exactly m nonzero elements in both D and P. Therefore, the cubic complexity of the matrix inversion dominates the computation time of dual RankRLS training, and instead of (19),

$$\begin{aligned} &\text{the complexity of solving (17) using the cost function (21)} \\ &\text{in the scoring setting is } O(m^3). \end{aligned} \tag{26}$$

In some cases, we may also want to exclude the tied input pairs from the training process, that is, the ones whose both inputs belong to the same equivalence class as determined by the scoring. In this case, it is also possible to construct a form analogous to the one presented in (25) that can be efficiently used in computations. However, presenting it would lead to very technical and detailed considerations, and hence we leave it out from this paper.

The rest of this section is organized as follows. Section 3.1 concerns the primal form of RankRLS and its efficient implementation when using scored training data. In Sect. 3.2, we consider a way to train RankRLS simultaneously with several values of the regularization parameter. Computationally efficient cross-validation algorithms are proposed for label ranking in Sect. 3.3 and for object ranking in Sect. 3.4. Finally, Sect. 3.5 considers a large-scale training algorithm based on sparse approximation.

3.1 Primal RankRLS

In some cases, the number m of inputs $x_1, \ldots, x_m$ in the training data is much larger than the number of dimensions n in the feature space $\mathcal{F}$, that is, $n < m$ and $\mathcal{F} = \mathbb{R}^n$. Then, the sequence of mapped inputs is a matrix

$$H = (\Phi(x_1), \ldots, \Phi(x_m)) \in \mathbb{R}^{n \times m}$$

and the function (6) minimizing (5) can be equivalently expressed as

$$f(x) = \Phi(x)^{\mathrm{T}} HA = \Phi(x)^{\mathrm{T}} W, \tag{27}$$

where

$$W = HA$$

denotes the n-dimensional normal vector of the hyperplane corresponding to the RankRLS solution in the feature space, and A is the vector that determines the function (6). Output prediction for unseen inputs is more efficient with (27) than with (6) if $n < m$ and the mapping is fast to compute.

We next show that, if $n < m$, also the training process can be performed in a more efficient way than with dual RankRLS (17). We call this method the primal version of RankRLS. With the primal version, the computational complexity of the training process becomes more dependent on the dimensionality n of the feature space rather than on the number of inputs m.

Now we may write the algorithm (15) as

$$\mathcal{A}(G) = \operatorname*{argmin}_{W \in \mathbb{R}^n} J(W, G),$$

where

$$J(W, G) = (N - M^{\mathrm{T}} H^{\mathrm{T}} W)^{\mathrm{T}} (N - M^{\mathrm{T}} H^{\mathrm{T}} W) + \lambda W^{\mathrm{T}} W.$$

We take the derivative of $J(W, G)$ with respect to W:

$$\begin{aligned} \frac{d}{dW} J(W, G) &= -2HM(N - M^{\mathrm{T}} H^{\mathrm{T}} W) + 2\lambda W \\ &= -2HMN + 2(HMM^{\mathrm{T}} H^{\mathrm{T}} + \lambda I)W. \end{aligned}$$

Then, we set the derivative to zero and solve it with respect to W:

$$W = (HMM^{\mathrm{T}} H^{\mathrm{T}} + \lambda I)^{-1} HMN. \tag{28}$$

The computational complexity of the matrix inversion in (28) is in this case $O(n^3)$. Recall from (18) that constructing the matrices MM^{T} and MN needs $O(l)$ time. The other dominant operations involved in (28) are the multiplication of H with MM^{T} and HMM^{T} with H^{T} which require $O(m^2 n)$ and $O(n^2 m)$ time, respectively. Alternatively, one can first multiply H with M in $O(nl)$ time, because M contains only $2l$ nonzero elements, and then HM with its transpose in $O(n^2 l)$ time. In this case, the dominant terms are $O(n^3)$ and $O(n^2 l)$. Therefore we have:

$$\text{the complexity of calculating (28) is } O(n^3 + \min(n^2 m + m^2 n + l, n^2 l)). \tag{29}$$

However, there are some special cases in which the matrix multiplications can be performed more efficiently, as shown in the following.

Let us consider ranking in the scoring setting using the magnitude preserving cost function (21), that is, (23) and (25) hold. Then, the matrix $HMM^{\mathrm{T}} H^{\mathrm{T}}$ can be computed efficiently from

$$HMM^{\mathrm{T}} H^{\mathrm{T}} = HDH^{\mathrm{T}} - (HP)(P^{\mathrm{T}} H^{\mathrm{T}})$$

and HMN from

$$\begin{aligned} HMN &= HMM^{\mathrm{T}} S \\ &= H(DS - P(P^{\mathrm{T}} S)). \end{aligned}$$

The multiplication HDH^{T} needs $O(n^2 m)$ time, because D is a diagonal matrix. Further, the multiplication of H with P can be performed in $O(nm)$ time, because H has n rows and there are exactly m nonzero elements in P. The other complexities can be analyzed analogously. Therefore, we have:

$$\begin{aligned} &\text{the complexity of calculating (28) using the cost function (21)} \\ &\text{in the scoring setting is } O(n^3 + n^2 m), \end{aligned} \tag{30}$$

where the first term corresponds to the matrix inversion involved in (28) and the second to the matrix multiplications.

3.2 Efficient regularization and learning multiple outputs

For simplicity, we assume in this section that the equations (23) and (25) hold, that is, the cost function (21) is used in the scoring setting. Generalizations to cases in which the assumption does not hold can also be made but we omit their consideration from this paper, because their presentations would be too long and technical.

As noted in Sect. 2, the Laplacian matrix $D - PP^{\mathrm{T}}$ of a preference graph is positive semidefinite and the kernel matrix K is assumed to be strictly positive definite. Therefore, it can be shown that the matrix $(D - PP^{\mathrm{T}})K$ is diagonalizable and has real non-negative eigenvalues (see Horn and Johnson 1985, p. 465). By performing the eigen decomposition of $(D - PP^{\mathrm{T}})K$, it is possible to calculate the solutions (17) of dual RankRLS for several values of the regularization parameter λ with only small increase in the computational cost compared to calculating with just one value. Let us consider the eigen decomposition $V\Lambda V^{-1} = (D - PP^{\mathrm{T}})K$, where V is the matrix of eigenvectors and Λ is a diagonal matrix containing the corresponding eigenvalues. The decomposition and the inversion V^{-1} can be calculated in $O(m^3)$ time, and hence the complexity is not worse than that given in (26). Let $Q = V^{-1}(D - PP^{\mathrm{T}})S$, where the matrix products and subtractions can be computed in $O(m^2)$ time. Then, the solution for a regularization parameter value λ can be calculated from

$$A = V(\Lambda + \lambda I)^{-1}Q, \tag{31}$$

in $O(m^2)$ time, since $\Lambda + \lambda I$ is a diagonal matrix and Q is an m-dimensional vector.

An analogous approach can be used also in the primal form (28) by calculating the matrix $H(D - PP^{\mathrm{T}})H^{\mathrm{T}}$, its eigen decomposition $V\Lambda V^{\mathrm{T}}$, and the vector $Q = V^{\mathrm{T}}H(D - PP^{\mathrm{T}})S$ in altogether $O(n^3 + n^2m)$ time. In this case, the solutions for different values of regularization parameter can be subsequently obtained in $O(n^2)$ time, since Q is in an n-dimensional vector.

We now consider learning multiple outputs simultaneously, that is, we assume that we have multiple scores per each input. Analogously to the standard RLS, instead of having a single column matrix S for the outputs, we now have a $m \times v$-matrix S, where v is the number of outputs. We observe that only the time complexity of calculating Q and the subsequent use of (31) for different values of the regularization parameter depend on v. In the dual case, the complexity of calculating Q is $O(m^2v)$. Therefore, training RankRLS in the dual case needs altogether $O(m^3 + m^2v)$ time in the first phase. Subsequently, $O(m^2v)$ time is needed per each different value of the regularization parameter λ. In the primal case, the calculation of Q needs $O(n^2v + mnv)$ time, and hence the time complexity of the first phase in the primal case is $O(n^3 + n^2m + n^2v + mnv)$. Subsequently, the solution for a regularization parameter value can be calculated in $O(n^2v)$ time.

3.3 Cross-validation for label ranking

In Pahikkala et al. (2006a), we described an efficient method for calculating hold-out estimates for the standard RLS regression algorithm in which several inputs were held out simultaneously. We also described a similar hold-out algorithm for label ranking with RankRLS in the scoring setting (Pahikkala et al. 2007a), that is, by leaving out all inputs associated to the same object simultaneously. Here, we make a more thorough consideration of the label ranking hold-out algorithm for dual RankRLS without tying it to the scoring setting.

Recall that in label ranking tasks, we assume that each input consists of an object and a label and one object is associated to several inputs. However, each input is associated to only one object. Therefore, we assume that there are no preferences between inputs that are associated to different objects. Let $U \subset \{1, \ldots, m\}$ denote the index set that contains the indices of the inputs that are associated to a hold-out object. Leaving more than one object out can be defined analogously. In that case, U would refer to the union of index sets of every hold-out object.

With any matrix (or a column vector) Ψ that has its rows indexed by a superset of U, we use the subscript U so that the matrix Ψ_U contains only the rows that are indexed by U. Similarly, for any matrix Ψ that has its rows indexed by a superset of U and columns indexed by a superset of V, we use Ψ_{UV} to denote a matrix that contains only the rows indexed by U and the columns indexed by V. Moreover, we also denote $\overline{U} = \{1, \ldots, m\} \setminus U$. Further, let $f_{\overline{U}}$ be the hypothesis obtained by training RankRLS without the preferences between the inputs indexed by U. We will frequently make use of the following block matrix multiplication identity:

$$\Psi^{\mathrm{T}} \Upsilon = (\Psi_U)^{\mathrm{T}} \Upsilon_U + (\Psi_{\overline{U}})^{\mathrm{T}} \Upsilon_{\overline{U}},$$

where Ψ and Υ are matrices whose rows are indexed by a superset of U.

Note that in the case of label ranking, we obtain the incidence matrix corresponding to the training data from which the inputs associated to the hold-out object have been removed by just removing the rows indexed by U. After removing the rows, the columns corresponding to the edges incident to the hold-out inputs contain only zeros. This is because both the start and end vertices of those edges are indexed by U. Therefore, the columns have no effect on the values of the matrix multiplications.

Let $Q = M_{\overline{U}}(M_{\overline{U}})^{\mathrm{T}} K_{\overline{U}\overline{U}} + \lambda I_{\overline{U}\overline{U}}$. Then, according to (7) and (17), the predicted scores for the inputs of the hold-out object can be obtained from

$$\begin{aligned} f_{\overline{U}}(X_U) &= K_{U\overline{U}} Q^{-1} M_{\overline{U}} N \\ &= K_{U\overline{U}} Q^{-1} (MN)_{\overline{U}}. \end{aligned} \tag{32}$$

However, having already calculated the solution with the whole training data, the predictions for the hold-out instance can be performed more efficiently than using (32) which calculates Q^{-1}.

Let $R = (MM^{\mathrm{T}} K + \lambda I)^{-1}$. In the case of label ranking, the entries of the matrix $M_{\overline{U}}(M_U)^{\mathrm{T}}$ are zeros for all U, because there are no preferences between the inputs indexed by U and inputs indexed by $\overline{U}$. Therefore, we can write

$$\begin{aligned} Q &= M_{\overline{U}}(M_{\overline{U}})^{\mathrm{T}} K_{\overline{U}\overline{U}} + \lambda I_{\overline{U}\overline{U}} \\ &= M_{\overline{U}}(M_{\overline{U}})^{\mathrm{T}} K_{\overline{U}\overline{U}} + M_{\overline{U}}(M_U)^{\mathrm{T}}(K_{U\overline{U}}) + \lambda I_{\overline{U}\overline{U}} \\ &= M_{\overline{U}}((M_{\overline{U}})^{\mathrm{T}} K_{\overline{U}} + (M_U)^{\mathrm{T}} K_U)(I_{\overline{U}})^{\mathrm{T}} + \lambda I_{\overline{U}\overline{U}} \\ &= M_{\overline{U}} M^{\mathrm{T}} (K_{\overline{U}})^{\mathrm{T}} + \lambda I_{\overline{U}\overline{U}} \\ &= (R^{-1})_{\overline{U}\overline{U}}. \end{aligned}$$

Then, due to the matrix inversion lemma (see e.g. Horn and Johnson 1985),

$$Q^{-1} = R_{\overline{U}\overline{U}} - R_{\overline{U}U}(R_{UU})^{-1} R_{U\overline{U}}.$$

Therefore,

$$\begin{aligned} f_{\overline{U}}(X_U) &= K_{U\overline{U}}(R_{\overline{U}\overline{U}} - R_{\overline{U}U}(R_{UU})^{-1}R_{U\overline{U}})(MN)_{\overline{U}} \\ &= K_{U\overline{U}}(R_{\overline{U}\overline{U}}(MN)_{\overline{U}} - R_{\overline{U}U}(R_{UU})^{-1}R_{U\overline{U}}(MN)_{\overline{U}}) \\ &= K_{U\overline{U}}(R_{\overline{U}}(I_{\overline{U}})^{\mathrm{T}}M_{\overline{U}}N - R_{\overline{U}U}(R_{UU})^{-1}R_{U\overline{U}}(MN)_{\overline{U}}) \\ &= K_{U\overline{U}}(R_{\overline{U}}(M - (I_U)^{\mathrm{T}}M_U)N - R_{\overline{U}U}(R_{UU})^{-1}R_{U\overline{U}}(MN)_{\overline{U}}) \\ &= K_{U\overline{U}}(R_{\overline{U}}MN - R_{\overline{U}U}(MN)_U - R_{\overline{U}U}(R_{UU})^{-1}R_{U\overline{U}}(MN)_{\overline{U}}) \\ &= K_{U\overline{U}}((RMN)_{\overline{U}} - R_{\overline{U}U}(MN)_U - R_{\overline{U}U}(R_{UU})^{-1}R_{U\overline{U}}(MN)_{\overline{U}}). \end{aligned} \tag{33}$$

If (15) has been solved with the whole training data, we already have the matrices R, MN, and RMN stored in the memory, and hence the computational complexity of calculating the matrix inversions, products, and subtractions (in the optimal order) involved in (33) is $O(|\overline{U}||U|+|U|^3) = O(m|U|+|U|^3)$. This is more efficient than the method (32) which calculates the inverse of Q with complexity $O(m^3)$. Assuming that the size of the label sets are of the same size, there is $m/|U|$ objects in the training data, and hence the complexity of calculating a leave-object-out cross-validation is $O((m/|U|)(m|U|+|U|^3)) = O(m^2+m|U|^2)$. This is more efficient than the training of dual RankRLS with the whole training data. Therefore, the cross-validation method can also be combined with the method of selecting the regularization parameter described in Sect. 3.2. We omit the formulas describing this combination, because their presentation would be too long and technical.

3.4 Leave-pair-out cross-validation for object ranking

Here, we consider object ranking in the scoring setting, and hence there is an edge between every input in the preference graph. We consider only the magnitude preserving cost function (21). Therefore, (23) and (25) hold. We now consider leave-pair-out cross-validation (LPOCV) in which every pair of inputs is held out from the training data at a time. If a certain pair of inputs is held out, the edges that are incident to those inputs are not used in the training process in that cross-validation round. In each round, the predictions for the hold-out inputs are calculated. These predictions can then be used for ranking performance measurement. This method is very useful for performance estimation when dealing with so small amounts of data that using a subset of the inputs as a separate test data does not provide reliable enough performance estimate. For a more detailed description of this method, we refer to Pahikkala et al. (2008a).

Cortes et al. (2007a) have proposed an algorithm that approximates the result of LPOCV for the object ranking in $O(m^2)$ time, provided that an inversion of a certain $m \times m$-matrix is already computed and stored in the memory. The larger the number of inputs m is, the closer the approximation to the exact result of the cross-validation is. Here, we improve their result by presenting an algorithm that calculates an exact result of LPOCV in $O(m^2)$ time, again given that the inverse of a certain $m \times m$-matrix is already computed and stored in the memory.

Before presenting the LPOCV algorithm, we consider the following result (for a proof, see e.g. Rifkin and Lippert 2007b; Johnson and Zhang 2008; Pahikkala et al. 2008a).

Lemma 2 *Let $U \subset \{1,\ldots,m\}$ denote the index set that contains the indices of the inputs belonging to the hold-out set and let $\overline{U} = \{1,\ldots,m\} \setminus U$. Further, let $K \in \mathbb{R}^{m\times m}$ be the*

kernel matrix constructed from the inputs X, $f_{\overline{U}}$ be the hypothesis obtained by training RankRLS without using the inputs indexed by U, and $c_{\overline{U}}$ be a cost function that depends only on the predictions made for the inputs indexed by $\overline{U}$. Then, the vector of predictions $f_{\overline{U}}(X)$ can be computed from

$$f_{\overline{U}}(X) = \operatorname*{arginf}_{r \in \mathbb{R}^m} \left\{ c_{\overline{U}}(r, G) + \lambda r^T K^{-1} r \right\}. \tag{34}$$

If K is singular, the term $r^T K^{-1} r$ should be interpreted as

$$\lim_{\epsilon \to 0^+} r^T (K + \epsilon I)^{-1} r.$$

The main insight of the lemma is that we can obtain the hold-out predictions by using a modified cost $c_{\overline{U}}$ that only takes account of the predictions of the inputs not belonging to the hold-out set U. In contrast, the regularizer $\lambda r^{\mathrm{T}} K^{-1} r$ does not have to be modified and it can still depend on all of the m predictions. Note that this property holds for any cost function, and hence the lemma provides us a powerful framework for designing cross-validation algorithms.

Next, we apply Lemma 2 to the cost function (21). Let

$$U = \{h_1, h_2\}$$

be the index set containing the indices h_1 and h_2 of the two hold-out inputs and let $\overline{U} = \{1, \ldots, m\} \setminus U$. Further, let $S = (s_1, \ldots, s_m)^{\mathrm{T}} \in \mathbb{R}^m$ be a vector of real valued scores of the inputs. We now reformulate the matrix form (24) so that its value is independent of the predictions for the hold-out inputs. Recall that the preference magnitudes in the scoring setting can be expressed with the differences of the scores of the inputs and that we also include the preferences with zero magnitudes in training for efficiency reasons. Therefore, the cost function (21) which is calculated over the whole training data can be expressed as

$$c(r, G) = \frac{1}{2} \sum_{i,j=1}^{m} ((s_i - s_j) - (r_i - r_j))^2.$$

The sum is multiplied with the constant $\frac{1}{2}$, because the sum contains each index pair twice, since in this setting $((s_i - s_j) - (r_i - r_j))^2 = ((s_j - s_i) - (r_j - r_i))^2$ and $((s_i - s_i) - (r_i - r_i))^2 = 0$ for all $i, j \in \{1, \ldots, m\}$. In order to make the cost function independent of the predictions for the hold-out inputs, we remove the terms involving the hold-out inputs from the sum. Thus, the cost function from which the terms have been removed is

$$\begin{aligned} c_{\overline{U}}(r, G) &= \frac{1}{2} \sum_{i,j \in \overline{U}} ((s_i - s_j) - (r_i - r_j))^2 \\ &= (m-2) \sum_{i \in \overline{U}} (s_i - r_i)^2 - \sum_{i,j \in \overline{U}} (s_i - r_i)(s_j - r_j) \\ &= (S - r)^{\mathrm{T}} \widetilde{L} (S - r), \end{aligned} \tag{35}$$

where $\widetilde{L} \in \mathbb{R}^{m \times m}$ is a matrix whose entries are defined as

$$\widetilde{L}_{i,j} = \begin{cases} -1 & \text{if } i \neq j \text{ and } i, j \in \overline{U}, \\ m-3 & \text{if } i = j \text{ and } i \in \overline{U}, \\ 0 & \text{otherwise.} \end{cases}$$

The matrix form (35) is similar to (24) except that the Laplacian matrix $\widetilde{L}$ corresponds to a graph from which all edges incident to the hold-out inputs have been removed.

Next, we substitute (35) into (34). Then, derivating (34) with respect to r and setting it to zero provides us the predictions for all of the m inputs made by $f_{\overline{U}}$:

$$f_{\overline{U}}(X) = (\widetilde{L} + \lambda K^{-1})^{-1} \widetilde{L} S.$$

Now, multiplying with I_U from the left provides us the predictions for the hold-out inputs

$$f_{\overline{U}}(X_U) = I_U (\widetilde{L} + \lambda K^{-1})^{-1} \widetilde{L} S. \tag{36}$$

We continue by observing that we can also write

$$\widetilde{L} = \widetilde{D} - B B^{\mathrm{T}}, \tag{37}$$

where $B \in \mathbb{R}^{m \times 3}$ is a matrix whose values are determined by

$$B_{i,j} = \begin{cases} 1 & \text{if } i \in \overline{U} \text{ and } j = 1, \\ \sqrt{m-2} & \text{if } i = h_1 \text{ and } j = 2, \\ \sqrt{m-2} & \text{if } i = h_2 \text{ and } j = 3, \\ 0 & \text{otherwise} \end{cases}$$

and

$$\widetilde{D} = (m-2) I \in \mathbb{R}^{m \times m}.$$

Let

$$Q = (\widetilde{D} + \lambda K^{-1})^{-1}.$$

Using (37), we can rewrite (36) as

$$\begin{aligned} f_{\overline{U}}(X_U) &= I_U (Q^{-1} - B B^{\mathrm{T}})^{-1} \widetilde{L} S \\ &= I_U (Q - Q B (-I + B^{\mathrm{T}} Q B)^{-1} B^{\mathrm{T}} Q) \widetilde{L} S \\ &= (Q \widetilde{L} S)_U - (Q B)_U (-I + B^{\mathrm{T}} Q B)^{-1} B^{\mathrm{T}} Q \widetilde{L} S, \end{aligned} \tag{38}$$

where the second equality is due to the Sherman-Morrison-Woodbury formula. Let $C \in \mathbb{R}^{m \times 3}$ be a matrix whose values are determined by

$$C_{i,j} = \begin{cases} 1 & \text{if } j = 1, \\ 0 & \text{otherwise} \end{cases}$$

and let

$$R = \begin{pmatrix} -1 & \sqrt{m-2} & 0 \\ -1 & 0 & \sqrt{m-2} \end{pmatrix}.$$

We observe that we can write

$$(I_U)^{\mathrm{T}} R = B - C,$$

where I is an $m \times m$-identity matrix, and hence

$$R = B_U - C_U.$$

Let us assume that we have calculated the matrices

$$Q, \widetilde{D}S, Q\widetilde{D}S, QC, C^{\mathrm{T}}QC, C^{\mathrm{T}}S, QCC^{\mathrm{T}}S, \text{ and } C^{\mathrm{T}}Q\widetilde{D}S, \tag{39}$$

and stored them into the memory before starting the hold-out calculations. In order to calculate (38), we have to compute the following matrices:

$$\begin{aligned} B^{\mathrm{T}}QB &\in \mathbb{R}^{3\times 3}, \\ B^{\mathrm{T}}Q\widetilde{L}S &\in \mathbb{R}^{3\times 1}, \\ (QB)_U &\in \mathbb{R}^{2\times 3}, \\ (Q\widetilde{L}S)_U &\in \mathbb{R}^{2\times 1}. \end{aligned}$$

Given that the matrices (39) have already been calculated, the above matrices can be calculated in a constant time as follows:

$$\begin{aligned} B^{\mathrm{T}}QB &= (C + (I_U)^{\mathrm{T}}R)^{\mathrm{T}}Q(C + (I_U)^{\mathrm{T}}R) \\ &= C^{\mathrm{T}}QC + R^{\mathrm{T}}I_U QC + C^{\mathrm{T}}Q(I_U)^{\mathrm{T}}R + R^{\mathrm{T}}I_U Q(I_U)^{\mathrm{T}}R \\ &= C^{\mathrm{T}}QC + R^{\mathrm{T}}(QC)_U + (R^{\mathrm{T}}(QC)_U)^{\mathrm{T}} + R^{\mathrm{T}}Q_{UU}R, \\ B^{\mathrm{T}}Q\widetilde{L}S &= B^{\mathrm{T}}Q\widetilde{D}S - B^{\mathrm{T}}QBB^{\mathrm{T}}S \\ &= C^{\mathrm{T}}Q\widetilde{D}S + R^{\mathrm{T}}(Q\widetilde{D}S)_U - B^{\mathrm{T}}QB(C^{\mathrm{T}}S + R^{\mathrm{T}}S_U), \\ (QB)_U &= (QC)_U + (Q(I_U)^{\mathrm{T}}R)_U \\ &= (QC)_U + Q_{UU}R, \\ (Q\widetilde{L}S)_U &= (Q\widetilde{D}S)_U - (QBB^{\mathrm{T}}S)_U \\ &= (Q\widetilde{D}S)_U - (QB)_U B^{\mathrm{T}}S \\ &= (Q\widetilde{D}S)_U - ((QC)_U + Q_{UU}R)(C^{\mathrm{T}}S + R^{\mathrm{T}}S_U). \end{aligned}$$

By substituting these into (38), the hold-out predictions for a pair of inputs can be calculated in a constant time.

Concerning the matrices (39) calculated in advance, the calculation of the matrix Q is the computationally dominant one. Namely, its time complexity is $O(m^3)$ in the worst case of K being of full rank. This is the same as that of training the RankRLS algorithm in the worst case. However, if the rank of K is not full, the matrix Q can be calculated as follows.

Let $K = V\Lambda V^{\mathrm{T}}$ be the eigen decomposition of the kernel matrix, where V contains the eigenvectors of K and Λ is a diagonal matrix containing the eigenvalues of K. Then,

$$Q = V\widehat{\Lambda}V^{\mathrm{T}},$$

where $\widehat{\Lambda}$ is the diagonal matrix whose elements are determined by

$$\widehat{\Lambda}_{i,i} = \frac{\Lambda_{i,i}}{\lambda + (m-2)\Lambda_{i,i}}.$$

The calculation of the other matrices in (39) need only $O(m^2)$ time if Q is already calculated.

After the matrices (39) are calculated, the overall complexity of LPOCV is $O(m^2)$, since only a constant time is needed to compute (38) for each hold-out set U and there are $O(m^2)$ different hold-out sets. This is advantageous, for example, if we have many independent ranking tasks we aim to learn from the same input data. In this case, the outputs are stored, instead of a single column matrix, in a matrix $S \in \mathbb{R}^{m\times v}$, where v is the number of tasks. Then, the time complexity of the cross-validation is $O(m^3 + m^2 v)$, since the complexity of calculating Q is not affected by the number of tasks.

3.5 Sparse approximation

If the number of inputs m is large, the time complexities (19) or (26) of training dual RankRLS may become infeasible and approximative techniques are needed. In this section, we propose an approach that is based on a similar kind of idea as the subset of regressors method (see e.g. Poggio and Girosi 1990; Smola and Schölkopf 2000; Rifkin et al. 2003; Quiñonero-Candela et al. 2007) for the standard regularized least-squares regression. More detailed considerations and experimental results of this approach for RankRLS are presented in Tsivtsivadze et al. (2008).

Recall that the training data contains a sequence of inputs $X = (x_1, \ldots, x_m) \in (\mathcal{X}^m)^{\mathrm{T}}$ of length m and let $R \subseteq \{1, \ldots, m\}$, where $|R| \ll m$. The inputs indexed by the set R are called the basis vectors. Now we consider instead of (6) a solution that allows only the inputs indexed by R to have nonzero coefficient, that is,

$$f(x) = \sum_{i\in R} a_i k(x, x_i). \tag{40}$$

Note that it is not guaranteed that the optimal solution with only $|R|$ nonzero coefficients a_i will have a representation as in formula (40), because the sparse approximation cannot straightforwardly resort to the representer theorem anymore. Clearly, the selection of the index set R may have an influence on results obtained by our method. Different approaches for selecting R are discussed, for example, in Rifkin et al. (2003). There, it was found that simply selecting the elements of R randomly performs no worse than more sophisticated methods.

The problem of finding this type of hypothesis can be solved by finding the coefficients a_i, where $i \in R$. In this case, the predictions for the training inputs can be expressed as $f(X) = (K_R)^{\mathrm{T}}A$ and the regularizer as $\lambda A^{\mathrm{T}} K_{RR} A$, where $A \in \mathbb{R}^{|R|}$ is a vector consisting of the coefficients a_i. Using these definitions, we present a method we call sparse RankRLS:

$$\mathcal{A}(G) = \operatorname*{argmin}_{A\in\mathbb{R}^{|R|}} J(A, G),$$

where

$$J(A, G) = (N - M^{\mathrm{T}}(K_R)^{\mathrm{T}}A)^{\mathrm{T}}(N - M^{\mathrm{T}}(K_R)^{\mathrm{T}}A) + \lambda A^{\mathrm{T}}K_{RR}A.$$

We take the derivative of $J(A, G)$ with respect to A, set it to zero, and solve with respect to A:

$$A = (K_R M M^{\mathrm{T}}(K_R)^{\mathrm{T}} + \lambda K_{RR})^{-1} K_R M N. \tag{41}$$

The computational complexity of calculating (41) can be analyzed in a similar way as that of the primal RankRLS (28), because the former contains the matrix K_R in place of H and K_{RR} in place of I, that is, we substitute $|R|$ in place of n in (29). However, the solution can be found more efficiently if we assume the scoring setting and that the magnitude preserving cost function (21) is used making (23) and (25) to hold. Then, the training complexity of the sparse RankRLS algorithm can be analyzed by substituting $|R|$ in place of n in (30) resulting in $O(m|R|^2)$ complexity, since $|R| \ll m$. Thus, the size of R can be selected so that these computation times are feasible.

The efficient selection of regularization parameter discussed in Sect. 3.2 can also be performed with sparse RankRLS using the following method. Here, we again assume that we use the cost function (21) in the scoring setting, and hence (23) and (25) hold. Using the Cholesky decomposition $K_{RR} = ZZ^{\mathrm{T}}$, where $Z \in \mathbb{R}^{|R|\times|R|}$ is a lower triangular matrix called the Cholesky triangle of K_{RR}, we can rewrite the solution (41) as follows:

$$A = (K_R M M^{\mathrm{T}}(K_R)^{\mathrm{T}} + \lambda Z Z^{\mathrm{T}})^{-1} K_R M M^{\mathrm{T}} S.$$

Note that since we assume the kernel matrix K to be strictly positive definite, it follows that also its principal submatrix K_{RR} is strictly positive definite, and hence Z is invertible. Let

$$V \Lambda V^{\mathrm{T}} = Z^{-1} K_R M M^{\mathrm{T}}(K_R)^{\mathrm{T}}(Z^{-1})^{\mathrm{T}} \tag{42}$$

be the eigen decomposition of $Z^{-1}K_R M M^{\mathrm{T}}(K_R)^{\mathrm{T}}(Z^{-1})^{\mathrm{T}}$, where V and Λ are the eigenvector matrix and diagonal matrix containing the corresponding eigenvalues, respectively. Further, let $\hat{\Lambda}_\lambda = (\Lambda + \lambda I)^{-1}$. Then,

$$\begin{aligned}(K_R M M^{\mathrm{T}}(K_R)^{\mathrm{T}} + \lambda Z Z^{\mathrm{T}})^{-1} &= (Z^{-1})^{\mathrm{T}}(V \Lambda V^{\mathrm{T}} + \lambda I)^{-1} Z^{-1} \\ &= (Z^{-1})^{\mathrm{T}} V \hat{\Lambda}_\lambda V^{\mathrm{T}} Z^{-1}.\end{aligned}$$

Therefore, we rewrite the solution (41) as follows:

$$A = (Z^{-1})^{\mathrm{T}} V \hat{\Lambda}_\lambda V^{\mathrm{T}} Z^{-1} K_R M M^{\mathrm{T}} S.$$

The decompositions and the inversion of Z can be calculated in $O(|R|^3)$ time, and hence the overall training complexity is not increased. The computational cost of calculating $\hat{\Lambda}_\lambda$ is $O(|R|)$, since $\Lambda + \lambda I$ is a diagonal matrix. When the matrices $V^{\mathrm{T}}Z^{-1}K_R M M^{\mathrm{T}} S \in \mathbb{R}^{|R|\times 1}$ and $(Z^{-1})^{\mathrm{T}}V \in \mathbb{R}^{|R|\times|R|}$ are stored in the memory, the subsequent training with different values of regularization parameters can be performed in $O(|R|^2)$ time.

We also note that the sparsity of the learned solution speeds up the prediction when nonlinear kernels are used. Namely, the prediction complexity scales with respect to $|R|$ times the complexity of calculating the kernel function.

4 Summary of computational benefits and comparison to RankSVM

Here, we make a summary about the computational properties of RankRLS and compare them to those of RankSVM. RankSVM (Herbrich et al. 1999; Joachims 2002) is a state-of-the-art ranking method very closely related to RankRLS. Their objective functions are the same except that RankSVM uses the hinge cost function:

$$c(f(X), G) = \sum_{i=1}^{l} \max(1 - g(e_i)\text{sign}(y_i), 0),$$

where $e_i = (x_h, x_j, y_i)$ are the observed preferences, $g(e_i) = f(x_h) - f(x_j)$, and $\text{sign}(y_i)$ are the directions of the preferences e_i. As for RankRLS, there are also various different methods for finding a minimizer of the objective function. It can be argued that the hinge cost is a better approximation of the disagreement error than the squared costs as it does not penalize correct predictions with magnitudes larger than one. However, in our experiments, we observe that the ranking performance of RankRLS is essentially the same as that of RankSVM. A similar phenomenon has also been observed between support vector machine and regularized least-squares classifiers (see e.g. Rifkin 2002; Gestel et al. 2004; Zhang and Peng 2004). Thus, the computational issues become the main factor for deciding whether RankSVM or RankRLS is preferable.

Next, we investigate in which circumstances it is more beneficial to use RankSVM or RankRLS method from the computational complexity point of view. For simplicity, we make the investigation only in the case the preferences are induced by a scoring of the inputs. Further, we only consider the cost function (21), and hence (23) and (25) hold.

Recently, Joachims (2006) proposed an efficient linear support vector machine type of ranking algorithm for scored data. The training complexity of the algorithm is $O(\bar{n}m \log m)$, where $\bar{n}$ is the average number of nonzero features in the inputs. In our comparisons, we consider this algorithm in the linear case. Further, we investigate the possibilities to use this algorithm also in the nonlinear case.

We have divided our consideration into four cases. We start with small-scale learning using linear kernel in Sect. 4.1 and continue with nonlinear case in Sect. 4.2. With small-scale learning we refer to the case in which the number m of inputs in the training data is such that $O(m^3)$ time complexity can be afforded, and with large-scale learning we refer to the opposite case. Large-scale learning with linear and nonlinear kernels are considered in Sects. 4.3 and 4.4, respectively.

4.1 Linear small scale learning

Because of the efficient training algorithm in the linear case, RankSVM has a computational advantage over RankRLS when only one instance of the ranking method is needed and no parameter selection or performance evaluation with cross-validation are performed. However, the advantage becomes less clear when there is a need for selecting the value of the regularization parameter, learning multiple outputs, or performing cross-validation. For example, the ability to perform cross-validation efficiently is very important when the number of inputs with known scores is small, since in many cases large enough test sets for reliable performance estimation can not be afforded. Next, we present some examples in which these properties are especially beneficial.

Small-size data appears frequently, for example, when solving medical and biological tasks, and hence cross-validation is often the only reliable way to measure the ranking

performance. In this case, the efficient methods presented in Sects. 3.3 and 3.4 are very useful. For example, when aiming for a maximal AUC with biological data as considered by Parker et al. (2007), a common practice for performance evaluation is to use a ten-fold cross-validation. Then, the overall AUC is obtained by computing AUC for each fold and taking their average or by first pooling the predictions and computing AUC afterwards. Taking the average suffers from large variance, because the number of input pairs in each fold may be too small. Moreover, Parker et al. (2007) reported that the pooling technique suffers from a pessimistic bias. The efficient leave-pair-out cross-validation provides a third way for AUC calculation that avoids many of the pitfalls associated to the pooling and averaging techniques.

Another advantage of RankRLS in linear small-scale learning is its ability to learn multiple outputs at the cost of only one, provided that the number of outputs is linear in m or in n. For example, in our experiments with the Reuters data in Sect. 5.4, there are 25 outputs that can be learned in parallel. Of course, learning multiple outputs is also very efficient with the fast RankSVM training methods. However, the fast cross-validation algorithms of RankRLS can be combined with multiple output learning. This makes it possible, for example, to perform permutation tests similar to those used for classification (see e.g. Golland et al. 2005). In the permutation tests, the outputs or scores of the training data are shuffled randomly and the learner is then trained and cross-validated with the data having permuted outputs. The shuffling and training is repeated many times and the cross-validation results are used, for example, to estimate the reliability of the cross-validation results with the original training data. This method can be used very efficiently with the RLS-based learning algorithms, because the permuted output vectors can be considered as extra outputs that can be learned and cross-validated in parallel.

4.2 Nonlinear small scale learning

In general, when nonlinear kernel functions are used, support vector machine (SVM) type of learners have an advantage in prediction time, because the form of the SVM solution may be sparse. However, this depends on the level of regularization and the amount of noise in the training data.

RankRLS has the advantage that its training time scales linearly instead of quadratically with respect to the number m of inputs in the training data. To our knowledge, at least the most commonly used implementations of nonlinear RankSVM scale roughly quadratically with respect to the number of preferences, and hence their computational complexity can be considered to be at least of the order $O(m^4)$, because the number of preferences in the scoring setting is assumed to be of order m^2. This makes RankSVM impractical even on small datasets. The cubic complexity of RankRLS makes it thus the method of choice when using nonlinear kernels and datasets which consist of at most a few thousand inputs. However, the dual implementations of the RankSVM do not necessarily represent the state-of-the-art in kernel based SVM ranking.

To demonstrate the scalability properties of nonlinear RankRLS and RankSVM algorithms, we provide a comparison of running times on the acq dataset of the Reuters AUC-maximization task (see Sect. 5.4 for description of the task and data). The RankSVM implementation is the one included in the SVM-light package, for RankRLS we use our own Python implementation. The runs are performed on a modern desktop computer using the Gaussian kernel and the default parameter values of the software packages are used. The results are presented in Table 1.

The runtime comparison of training provides further empirical support to the conclusions derived from the computational complexity considerations. RankRLS is efficient to use in

Table 1 Runtime comparisons of training for nonlinear RankRLS and RankSVM on the acq dataset. The number of inputs in the training data ranges from 200 to 6000, the runtimes are measured in seconds

	Running times								
Inputs	200	500	750	1000	1500	2000	2500	4000	6000
RankRLS	1	3	5	10	24	48	83	280	841
RankSVM	2	150	579	1740	4685	13707	20055	–	–

the small-scale setting where the number of inputs in the training data is measured in thousands. RankSVM however does not scale well, for example, at a point of 2500 inputs where RankRLS training takes less than one and a half minutes, training RankSVM takes five and a half hours. Taking further into consideration the efficient regularization, cross-validation and multiple output learning algorithms presented for RankRLS, it is clearly the better choice in this setting.

Next, we consider an alternative approach using the empirical kernel map (Schölkopf et al. 1999) to transform the SVM dual problem into a primal one, and hence to achieve cubic complexity also for the RankSVM. Formally, if a full rank kernel matrix K is decomposed, for example, with the Cholesky decomposition

$$K = ZZ^{\mathrm{T}},$$

where the Cholesky triangle $Z \in \mathbb{R}^{m \times m}$ of K can be considered as an empirical feature space representation of the input sequence X. It can be shown that after a linear RankSVM is trained with these features, the dual variables needed in prediction for new inputs can be obtained by multiplying the normal vector of the learned separating hyperplane with the inverse of Z^{T}. The computational complexity of the Cholesky decomposition of K is $O(m^3)$. After the decomposition is performed, the training of RankSVM with this feature representation is of complexity $O(m^2 \log m)$, because the average number of nonzero features per input in this case is m. When testing in practice this approach for training a single RankSVM learner, we observed training times that were very close to that of training a single RankRLS learner. This is because the $O(m^3)$ complexities of the eigen decomposition used in training RankRLS and the Cholesky decomposition in training RankSVM dominate the running times.

On the one hand, the Cholesky decomposition has to be performed only once, since the same feature space representation can be used for multiple outputs, multiple values of the regularization parameter, and in each cross-validation round. On the other hand, the $O(m^2 \log m)$ time is spent for every combination of the regularization parameter, every separate output, and every round in a cross-validation. Compared to that, RankRLS spends $O(m^2)$ time for every combination of the regularization parameter and output. However, the fast cross-validation properties of RankRLS make it more suitable than RankSVM for small-scale nonlinear ranking tasks. For example, the constant time hold-out computation introduced in Sect. 3.4 means that the eigen decomposition still dominates the RankRLS running time in leave-pair-out cross-validation but the complexity of RankSVM would rise to $O(m^4 \log m)$, since there are m^2 cross-validation rounds.

4.3 Linear large scale learning

Recall from (30) that the computational complexity of training the primal RankRLS is $O(n^3 + n^2 m)$, where n is the dimensionality of the feature space. Further, after RankRLS is

trained once, the level of regularization can be adjusted and multiple tasks learned efficiently as shown in Sect. 3.2. If n is a small constant and m is large enough, these properties make RankRLS faster to train than RankSVM that has the $O(\bar{n}m \log m)$, where $\bar{n}$ is the average number of nonzero features per input, training complexity.

If both the number of inputs in the training data m and the number of features n are large, the cubic time complexities of training RankRLS with the matrix calculus based techniques become infeasible. However, it may still be possible to take advantage of the sparsity of the feature representation of the inputs, that is, $\bar{n}$ being small. We note that, similarly to the standard RLS regression (see e.g. Rifkin et al. 2003; Shewchuk 1994), RankRLS can also be trained in such circumstances using conjugate gradient type of algorithms where the complexity of each iteration is $O(\bar{n}m)$. How close the coefficient vector obtained with this method is to the minimizer of (16) depends of the number of iterations. Since we assume the use of the cost function (21) in the scoring setting, we can write $MM^{\mathrm{T}} = D - PP^{\mathrm{T}}$, where D and P are defined as in the beginning of Sect. 3. Moreover, recall that in both the object and label ranking cases, the matrices P and D have only m nonzero entries. Further, let $H \in \mathbb{R}^{n \times m}$ be the sparse matrix containing the feature vectors of the training inputs having an average of $\bar{n}$ nonzero features per input and let $v \in \mathbb{R}^m$ be a vector. Then, we can compute the product

$$(KMM^{\mathrm{T}}K + \lambda K)v = H^{\mathrm{T}}HDH^{\mathrm{T}}Hv - H^{\mathrm{T}}HPP^{\mathrm{T}}H^{\mathrm{T}}Hv + \lambda H^{\mathrm{T}}Hv$$

in $O(\bar{n}m)$ time, since H contains approximately $\bar{n}m$ nonzero elements, and both D and P contain only m nonzero elements. Computing this product is the most expensive operation in each conjugate gradient iteration.

We run a test of the conjugate gradient algorithm using the Reuters classification task and linear kernel (see Sect. 5.4) using more than 12000 inputs and features, which generate over 23 million pairwise preferences. The algorithm needs only a couple of hundred iterations to converge, and hence the training takes only a few seconds.

4.4 Nonlinear large scale learning

The cubic complexity of nonlinear RankRLS is impractical in large-scale learning. However, it is possible to use sparse approximations as discussed in Sect. 3.5 having $O(m|R|^2)$ training complexity, where R is the set consisting of the indices of the basis vectors and $|R| \ll m$. This type of approximations are also possible for SVM type of learners as outlined in the following. Similarly to the empirical kernel map approach described in Sect. 4.2, the training tasks can again be transformed in $O(m|R|^2)$ time into a more efficient linear learning task, where the dimension of the feature space is $|R|$. Formally, the use of the sparse approximation corresponds to the use of the following type of modified kernel function (see e.g. Quiñonero-Candela and Rasmussen 2005):

$$\tilde{k}(x, x') = k(x, X_R)(K_{RR})^{-1}k(x', X_R)^{\mathrm{T}}, \tag{43}$$

where X_R is a sequence of basis vectors and $k(x, X_R) \in (\mathbb{R}^{|R|})^{\mathrm{T}}$ is a row vector consisting of the kernel evaluations between the input x and the training inputs indexed by R. Therefore, the kernel matrix corresponding to the modified kernel function $\tilde{k}$ can be written as

$$\widetilde{K} = (K_R)^{\mathrm{T}}(K_{RR})^{-1}K_R.$$

Now, if $(K_{RR})^{-1} = ZZ^{\mathrm{T}}$ is the Cholesky decomposition of $(K_{RR})^{-1}$, we can use $(K_R)^{\mathrm{T}}Z \in \mathbb{R}^{m \times |R|}$ as an empirical kernel map with which linear RankSVM can be trained. It can be

shown that after a linear RankSVM is trained using the feature representation obtained from this empirical kernel map, the vector of $|R|$ dual variables needed in making predictions for new inputs with the original kernel function k can be calculated by multiplying the normal vector of the learned separating hyperplane with Z.

After the feature representation based on the empirical kernel map has been constructed in $O(m|R|^2)$ time, the complexity of training a RankSVM is $O(|R|m \log m)$ for a single output and for a single value of the regularization parameter, since the number of dimensions in the feature representation determined by the empirical kernel map is $|R|$ and the representation is usually dense. Compared to that, after the eigen decomposition (42) and the other matrix operations needed in training a sparse RankRLS for a single output and a single value of the regularization parameter have been performed in $O(m|R|^2)$ time, subsequent training with different values of the regularization parameter for the same output is even more efficient, namely $O(|R|^2)$ per each parameter value.

5 Experiments

We test our ranking algorithms with various different tasks. The tasks considered are: ranking of dependency parses, document retrieval, binary document classification, and collaborative filtering. The two first tasks are instances of label ranking while the other two can be considered as object ranking problems. The pairwise preferences in all of the four tasks are induced by a scoring of the inputs. For example, in the document retrieval task the score of an input consisting of a query and a document is 1 if the document is relevant to the query and 0 otherwise. The document retrieval and binary classification tasks can be considered as bipartite ranking problems, that is, there are only two possible score values for the inputs. On the other hand, the true scores of the inputs in the parse ranking and collaborative filtering tasks are real numbers between a certain interval.

In all tasks, we test whether the irrelevant input pairs would be beneficial if included in the training process. For example, in document retrieval we do not measure the disagreement error between the inputs that are associated to different queries, but test if they are still useful in training.

We also compare the RankRLS algorithm with RankSVM and standard RLS regressor in all of the four tasks. The RankSVM baseline is always trained with only the relevant pairs, since the irrelevant pairs were found to be non-beneficial in our experiments with RankRLS. Moreover, in the document retrieval experiments, RankBoost is used as additional baseline. We also compare the cost functions (20), (21), and (22) with each other on the non-bipartite ranking tasks, since they are equal in bipartite tasks.

Both RankRLS and RLS regressor have a regularization parameter λ that controls the trade-off between the minimization of the training error and the complexity of the function. RankSVM has a similar parameter. The parameters are selected using cross-validation from the scale $[2^{-15}, 2^{-14}, \ldots, 2^{-14}, 2^{15}]$. Further, the used kernel functions have parameters that are set by cross-validation on the training data. Whenever a statistical significance is reported, the Wilcoxon signed-ranks test (Wilcoxon 1945) has been used with 0.05 as a significance threshold.

In Sect. 5.1, a simple example is presented in which we consider the effect of having irrelevant input pairs in the training process. Section 5.2 presents our experiments with parse ranking and Sect. 5.3 with document retrieval. We consider maximizing the area under ROC curve in Sect. 5.4 and collaborative filtering in Sect. 5.5.

5.1 The case of irrelevant input pairs

To investigate the possible effects of the irrelevant input pairs in the training process, we now present an artificial label ranking example that is illustrated in Fig. 2. In both figures, the feature vectors $\Phi(x_i)$ of the four inputs x_i are depicted as circles and they reside on a one-dimensional feature space. We assume that there are two objects, and inputs denoted x_1 and x_2 are associated to the first object and x_3 and x_4 to the second one. Therefore, only two pairs of inputs are relevant to the label ranking task, namely the pairs (x_1, x_2) and (x_3, x_4).

The four inputs are given scores that are $s_1 = 2$, $s_2 = 1$, $s_3 = 4$, and $s_4 = 3$. The scores induce a direction of preference for the two relevant input pairs. These preferences are depicted with arrows between the inputs in Fig. 2 (left). The scores also induce preferences for the four input pairs that are not relevant to the label ranking task in question. Both the relevant and irrelevant preferences are depicted in Fig. 2 (right). We observe that the preference direction of the relevant edges goes from left to right in the feature space but the direction of the irrelevant edges is opposite.

Since the feature vectors in the example are one-dimensional and we are learning only linear scoring functions, there are only two possible ways in which the inputs can be ordered. Namely, from left to right or in the opposite way. In Fig. 2, the direction is determined by the normal vector of the hyperplane corresponding to the RankRLS solution. Therefore, if the irrelevant input pairs are excluded from the training process, RankRLS learns the first type of ordering and it can correctly predict the preferences for both of the relevant inputs pairs. However, if the four irrelevant pairs are included in training, they overwhelm the effect of the two relevant pairs, and hence RankRLS learns the wrong type of ordering. This type of phenomenon may occur frequently in the label ranking task, since the inputs associated to the same object are often clustered in a similar way as in this example. Therefore, we speculate that the irrelevant pairs are usually more harmful than useful if included in the training of RankRLS for label ranking tasks.

Another type of input pair that may turn out to have an effect to the ranking performance is a tied one, that is, a pair whose both inputs have the same score. The tied pairs of inputs are not considered in our definition of the disagreement error. However, in our experiments we test whether it has a beneficial or harmful effect on the ranking performance if the tied pairs are used in the training. For simplicity, we perform the test only with (21), because it is the only cost function in which the ties can be treated in a trivial way, that is, by setting the zero point of the parabola to be zero.

We may also consider leaving out some of the relevant input pairs in case there is some redundancy created by the transitivity of the preferences, for example, in the scoring setting. However, this may not be an optimal strategy if the data is too noisy.

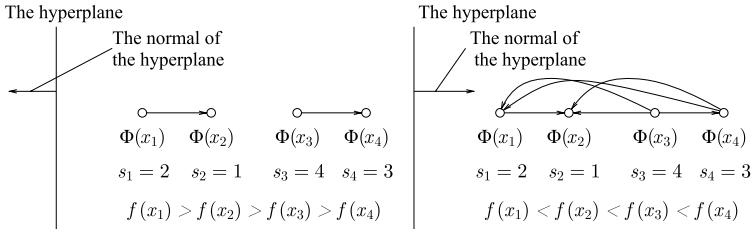

Fig. 2 Artificial label ranking example. Only the relevant input pairs are included in the training process (*left*). Both the relevant and irrelevant input pairs are included in the training process (*right*)

5.2 Ranking of dependency parses

First, we give a short description of the characteristics of the data. For a more detailed description, we refer to Tsivtsivadze et al. (2005). Next, we describe the task of parse ranking. Finally, we present the experimental evaluation.

Throughout our experiments, we use the BioInfer corpus (Pyysalo et al. 2007) which consists of 1100 manually annotated sentences.[1] For each sentence, we generate a set of candidate parses with a link grammar (LG) parser (Sleator and Temperley 1991). The LG parser is a full dependency parser based on a broad-coverage hand-written grammar. It generates all parses allowed by its grammar and applies a set of built-in heuristics to rank the parses. However, the ranking performance of the heuristics has been found to be poor when applied to biomedical text (Pyysalo et al. 2006), and hence subsequent ranking or selection methods are needed. In our previous studies, we used regularized least-squares regression for the reranking task that notably outperformed the LG heuristics (Tsivtsivadze et al. 2005). In these experiments, we use the graph kernel described in Pahikkala et al. (2006b).

In the task of parse ranking, each input consists of a sentence and a parse generated for it. We obtain a scoring for an input by comparing its parse to the hand annotated correct parse of its sentence. Tsivtsivadze et al. (2005) describes in detail how the scores are calculated. The relevant input pairs are those of which both inputs are associated to the same sentence and have different scores. All the other pairs are considered to be irrelevant to the task of parse ranking. We evaluate whether these irrelevant input pairs are beneficial if included in the training process. Furthermore, we compare the performance of RankRLS with the cost functions (20), (21), and (22).

In order to select the parameter values, we divide the set 1100 annotated sentences into two data sets containing 500 and 600 sentences. The first dataset is used for the parameter estimation and the second one is reserved for the final validation. The appropriate values of the regularization and the kernel parameters are determined by grid search with 10-fold cross-validation on the parameter estimation data.

Finally, the algorithm is trained on the whole parameter estimation data set with the selected parameter values and tested with the 600 sentences reserved for the final validation. The results of the validation are presented in Table 2. We observe that the regression approach is clearly worse than RankRLS. The performance differences between RLS regressor and RankRLS in the relevant pairs case are all statistically significant. Moreover, the performance differences of the results obtained by RankRLS methods when trained using

Table 2 Disagreement errors for the validation set using different methods. RankRLS is tested with the cost functions (20), (21), and (22), and both with only relevant pairs and all pairs

Method	Disagreement error
RankRLS (20)	0.225
RankRLS (20) All pairs	0.234
RankRLS (21)	0.222
RankRLS (21) All pairs	0.247
RankRLS (22)	0.216
RankRLS (22) All pairs	0.228
RankSVM	0.214
RLS Regressor	0.252

[1] Available at www.it.utu.fi/BioInfer.

relevant and all pairs are statistically significant indicating that the irrelevant pairs are harmful. Interestingly, the three cost functions seem to differ more in the all pairs experiments, while the results were clearly worse than with the relevant pairs only. When considering the relevant pairs, the results of RankSVM are very close to those of RankRLS.

5.3 Learning to rank for information retrieval

In this section, we present an evaluation of the RankRLS algorithm on the task of ranking documents according to queries using the publicly available Letor information retrieval dataset (Liu et al. 2007).[2] The problem is an example of a typical label ranking task. Given a set of query-document pairs, our aim is to learn to rank all the documents related to the same query according to how well they match the query. We also test how the inclusion of irrelevant preferences in the training data affects the performance of RankRLS. By irrelevant preferences, we mean in this setting pairs of inputs related to different queries or input pairs that are related to the same query, but have the same score associated with them. In addition to RankSVM and the standard RLS regressor comparisons, we also compare our results to those of RankBoost (Freund et al. 2003). Further details of these experiments can be found in Pahikkala et al. (2007b).

Recently, the Letor dataset for learning to rank in information retrieval containing three datasets of query-document pairs known as Ohsumed (16140 pairs), Trec2003 (49171 pairs) and Trec2004 (74170 pairs), as well as baseline results on RankBoost and RankSVM algorithms, has been made available. The Trec datasets contain only two possible scores for the inputs 0 and 1, while Ohsumed has three possible scores, 0, 1 and 2. In these experiments, we consider Ohsumed to be a bipartite ranking task by combining the scores 0 and 1 together.

We perform experiments on all of the three datasets. Because of the small dimensionality of the feature space (25 features in Ohsumed, 44 in Trecs) coupled with the large dataset sizes, we use the primal version of RankRLS which scales well in such settings as discussed in Sect. 3.1. Because of this choice, we use the linear kernel. The data is preprocessed by normalizing all of the feature values between 0 and 1 on per query basis. 5-fold cross-validation is used so that in each phase the learners are trained with three folds, parameters chosen on a fourth one and testing is done on the remaining fold. The fold split used is the one provided in the dataset. All results are averaged over the folds. We evaluate the results using disagreement error averaged over the different test queries. Such queries that are related only to documents that have score 1, or only to documents that have score 0, and thus contain no preferences, are not considered in the performance evaluation. The experimental results are presented in Table 3.

Table 3 Disagreement errors on the Letor datasets. RankRLS is tested in two settings: only relevant pairs are included and all pairs are included. Standard RLS regression, RankSVM and RankBoost are used as baselines

Method	Ohsumed	Trec2003	Trec2004
RankRLS	0.340	0.145	0.034
RankRLS All pairs	0.346	0.141	0.048
RLS Regressor	0.346	0.153	0.044
RankSVM	0.336	0.150	0.041
RankBoost	0.351	0.138	0.034

[2] Available at http://research.microsoft.com/users/tyliu/LETOR/.

In the Trec2004, RankRLS achieves best results when only those pairs that come from the same query and have documents with different relevance levels are used. On the other two datasets, the differences between the performance results of the two approaches are not statistically significant. There seems to be little to be gained from adding the irrelevant pairs to the training data, suggesting that the approach of training only with relevant pairs should be the default approach to take if given no prior information indicating otherwise. Compared to the baseline ranking algorithms, RankRLS achieves very similar performance. The standard RLS regressor, though slightly losing to the ranking algorithms, proves also to be quite a competitive choice.

5.4 Maximizing area under curve

It has been argued that for many types of binary classification tasks the area under the receiver operating characteristics curve (AUC) provides a more fitting performance measure than simple accuracy (Bradley 1997; Provost et al. 1998; Huang and Ling 2005). The task of AUC maximization can be considered as a bipartite ranking problem where each positive input is preferred over each negative one. Thus, it is equivalent to the task of disagreement error minimization (see e.g. Clémençon et al. 2005 for a more detailed consideration). Recent work in the field of support vector machines has shown AUC maximization to be a challenging task (see e.g. Rakotomamonjy 2004; Brefeld and Scheffer 2005; Joachims 2005). The need to consider all positive-negative input pairs easily leads to too cumbersome computations, or the use of approximative heuristics results in gains that are not statistically significant. However, the computational complexity of RankRLS is proportional to the number of individual inputs in the training data instead of the number of input pairs. This makes RankRLS a natural candidate for efficient AUC maximizing learner. For more discussion about this topic, see Pahikkala et al. (2008b).

In our experiments, we evaluate the capability of RankRLS to maximize AUC on the task of assigning topic labels to Reuters newswire documents. We approach the problem by transforming the original multi-label classification task into a series of binary classification tasks, where each sub-task consists of learning to classify documents on the basis of whether they have a certain topic or not.

Similarly to Brefeld and Scheffer (2005), we conduct the experiments on a subset of the Reuters-21578 dataset.[3] We limit the number of inputs in the training data to 500 to test the performance of the ranking methods on small imbalanced datasets. The rest 12397 documents are used as a test data. We take into account only the 25 most numerous classes, each of which corresponds to one possible topic a document can have. We consider the assignment of each of these labels as a separate binary classification problem, where the task is to decide whether a document should have the given label or not. Some of the documents belong to more than one class, and some to none of them.

We use the linear kernel. The regularization parameter λ is set using tenfold cross-validation on the training data, the chosen parameter is the one that provides maximal AUC on the pooled together cross-validation predictions (for a description of the pooling method, see e.g. Bradley 1997). We also calculate the 0.95 confidence intervals for the classifiers' AUC scores for each class. These statistical analyses are performed with SPSS 11.0. The comparison of RankRLS and standard RLS regression results is presented in Table 4 and similar comparison with RankSVM results is presented in Table 5.

[3] Available at http://www.daviddlewis.com/resources/testcollections/reuters21578/.

Table 4 Comparison of the AUC performance of the RankRLS and RLS algorithms on the Reuters-21578 dataset. In the first column is the name of the predicted class and in the next two are the AUC-values and corresponding confidence intervals for the tested algorithms. The last two columns present the numbers of positive inputs in the training set of 500 documents and test data of 12397 documents

Class	RankRLS	RLS Regressor	+train	+test
acq	0.980 (0.978–0.983)	0.979 (0.977–0.982)	94	2275
bop	0.966 (0.947–0.985)	0.880 (0.843–0.917)	4	101
cocoa	0.931 (0.891–0.970)	0.837 (0.776–0.899)	2	71
coffee	0.969 (0.948–0.990)	0.962 (0.950–0.975)	5	134
corn	0.970 (0.959–0.982)	0.950 (0.936–0.964)	11	226
cpi	0.947 (0.925–0.969)	0.601 (0.555–0.648)	3	94
crude	0.976 (0.969–0.982)	0.975 (0.969–0.982)	23	555
dlr	0.971 (0.961–0.981)	0.946 (0.926–0.965)	10	165
earn	0.994 (0.993–0.995)	0.993 (0.991–0.994)	158	3806
gnp	0.987 (0.981–0.993)	0.923 (0.891–0.956)	5	131
gold	0.970 (0.953–0.986)	0.922 (0.897–0.948)	4	120
grain	0.979 (0.973–0.985)	0.974 (0.968–0.980)	23	559
interest	0.965 (0.956–0.974)	0.952 (0.941–0.962)	19	459
livestock	0.701 (0.642–0.761)	0.637 (0.578–0.696)	3	96
money-fx	0.954 (0.946–0.962)	0.947 (0.938–0.957)	28	689
money-supply	0.949 (0.930–0.968)	0.907 (0.877–0.937)	7	165
nat-gas	0.957 (0.933–0.981)	0.941 (0.920–0.962)	5	100
oilseed	0.898 (0.877–0.919)	0.816 (0.783–0.849)	6	165
reserves	0.943 (0.908–0.977)	0.511 (0.458–0.564)	2	71
ship	0.949 (0.934–0.963)	0.925 (0.907–0.942)	13	273
soybean	0.876 (0.839–0.913)	0.805 (0.757–0.853)	4	107
sugar	0.985 (0.979–0.991)	0.964 (0.952–0.976)	6	156
trade	0.978 (0.970–0.986)	0.969 (0.960–0.977)	20	466
veg-oil	0.890 (0.865–0.914)	0.697 (0.656–0.739)	4	120
wheat	0.984 (0.978–0.990)	0.976 (0.969–0.983)	12	271

The results show that the RankRLS clearly outperforms the standard RLS regressor in the task of AUC maximization on the Reuters-21578 dataset. We observe that the smaller the amount of positive inputs is, the larger the performance gains seem to be. Between RankRLS and RankSVM no statistically significant differences are found.

We further examined whether including the ties in the training process has a beneficial or a harmful effect on the ranking performance. The effect was found to be negligible.

5.5 Collaborative filtering

We next present the results of a series of experiments run on a publicly available collaborative filtering dataset, the Jester Joke (Goldberg et al. 2001).[4] The task it to learn to predict the joke preferences of a user based on the preferences of other users, an approach com-

[4] Available at http://www.ieor.berkeley.edu/ goldberg/jester-data/.

Table 5 Comparison of the AUC performance of the RankRLS and RankSVM algorithms on the Reuters-21578 dataset

Class	RankRLS	RankSVM	+train	+test
acq	0.980 (0.978–0.983)	0.979 (0.977–0.982)	94	2275
bop	0.966 (0.947–0.985)	0.966 (0.949–0.983)	4	101
cocoa	0.931 (0.891–0.970)	0.923 (0.881–0.966)	2	71
coffee	0.969 (0.948–0.990)	0.962 (0.939–0.984)	5	134
corn	0.970 (0.959–0.982)	0.966 (0.956–0.975)	11	226
cpi	0.947 (0.925–0.969)	0.947 (0.925–0.969)	3	94
crude	0.976 (0.969–0.982)	0.976 (0.970–0.983)	23	555
dlr	0.971 (0.961–0.981)	0.972 (0.962–0.982)	10	165
earn	0.994 (0.993–0.995)	0.994 (0.992–0.995)	158	3806
gnp	0.987 (0.981–0.993)	0.987 (0.980–0.993)	5	131
gold	0.970 (0.953–0.986)	0.960 (0.940–0.980)	4	120
grain	0.979 (0.973–0.985)	0.979 (0.974–0.984)	23	559
interest	0.965 (0.956–0.974)	0.968 (0.960–0.976)	19	459
livestock	0.701 (0.642–0.761)	0.741 (0.698–0.784)	3	96
money-fx	0.954 (0.946–0.962)	0.959 (0.952–0.966)	28	689
money-supply	0.949 (0.930–0.968)	0.963 (0.950–0.976)	7	165
nat-gas	0.957 (0.933–0.981)	0.957 (0.933–0.981)	5	100
oilseed	0.898 (0.877–0.919)	0.895 (0.873–0.918)	6	165
reserves	0.943 (0.908–0.977)	0.920 (0.902–0.938)	2	71
ship	0.949 (0.934–0.963)	0.951 (0.939–0.964)	13	273
soybean	0.876 (0.839–0.913)	0.882 (0.848–0.916)	4	107
sugar	0.985 (0.979–0.991)	0.977 (0.970–0.985)	6	156
trade	0.978 (0.970–0.986)	0.982 (0.976–0.988)	20	466
veg-oil	0.890 (0.865–0.914)	0.853 (0.818–0.888)	4	120
wheat	0.984 (0.978–0.990)	0.983 (0.978–0.989)	12	271

mon to many recommender systems. In these experiments, we compare the performance of RankRLS with cost functions (20), (21) and (22) as measured by the disagreement error.

Jester Joke is a dataset of joke ratings, where a group of 73496 users has assigned real-valued ratings in the scale -10.0 to 10.0 to a set of 100 jokes, each rating describing how much they liked/disliked the joke in question. The task is to learn to predict the preferences of individual users from the preferences of the other users.

Our experimental setting follows that of Cortes et al. (2007b). We choose a set of 300 active users, for whom the task is to learn to predict their joke preferences. For each user, half of the jokes are chosen for training and half for testing. The preferences of the users are derived from the differences of the rating scores, a joke with a higher score is preferred to a joke with a lower score. To generate the features for the instances, a set of 300 reference users is chosen, and their given ratings for the corresponding joke are used as the feature values. In cases where these users have not rated the joke, the median of their ratings is used as the feature value.

In accordance to the original experimental setup, we perform three rounds of experiments, where we first choose the reference reviewers from people with 20–40, then with 40–60, and finally with 60–80 ratings. Finally, we remove these restrictions on feature den-

Table 6 Disagreement errors for the different versions of RankRLS, RankSVM, and the basic RLS regressor on the Jester dataset. RankRLS is tested with the cost functions (20), (21), and (22)

Method	20–40	40–60	60–80	All sizes
RankRLS (20)	0.413	0.400	0.378	0.371
RankRLS (21)	0.413	0.400	0.379	0.371
RankRLS (22)	0.445	0.426	0.388	0.379
RankSVM	0.413	0.400	0.378	0.371
RLS Regressor	0.414	0.401	0.378	0.371

sities and perform a fourth round of experiments using simply randomly chosen set of reference users. The kernel used is the Gaussian kernel and its width parameter was chosen from the interval $[2^{-15}, 2^{-14}, \ldots, 2^{14}, 2^{15}]$. The parameters for each experiment are chosen by taking the average over the performances on a hold-out set. The hold-out sets are created for each experiment similarly as the corresponding training/test data.

The results of the collaborative filtering experiments are included in Table 6. In these experiments, we found no difference between the performance of the cost functions (20) and (21). Further, we noticed that the cost function weighted by the inverse of the magnitude of the difference (22) performed worse than the other cost functions. This difference was statistically significant in each of the test settings. The performance of RankSVM was identical with that of the discretized (20) and magnitude preserving cost functions (21). Further, standard RLS also achieved as good performance as the ranking algorithms. We also tested the effects of including all pairs instead of only relevant ones in the training data. No performance differences were observed in the results.

6 Conclusions

There are many problems in which the aim is not to classify or to regress but to learn a ranking function. Inspired by the recent success of the regularized least-squares (RLS) based algorithms, we introduce a framework for RLS based ranking cost functions. Further, we propose three cost functions. We investigate their benefits and drawbacks from the perspectives of applicability and computational complexity. Finally, we propose a kernel-based preference learning algorithm, which we call RankRLS, for minimizing such cost functions.

RankRLS can be trained with a sequence of pairwise preferences between input data points and it outputs a ranking function for the individual inputs. The training time of RankRLS grows linearly with respect to the number of preferences and is cubic either with respect to the number of inputs or to the number of dimensions in the input space.

An important special case is the one in which the preference relation is induced by a scoring of input data points. For this case, it is possible to develop efficient shortcut methods using techniques based on matrix calculus. Namely, we introduce training algorithms whose complexities do not depend on the number of preferences, cross-validation algorithms for both object and label ranking, method for selection of the regularization parameter, and a method for learning multiple scorings simultaneously. These methods can be combined together. In addition, we show that some of these efficient methods can also be used in large-scale learning when the sparse approximation is used.

We also make a thorough comparison of the computational benefits and drawbacks of RankRLS in both small-scale and large-scale learning tasks with those of RankSVM that can be considered as a state-of-the-art ranking method. Moreover, both the linear and non-linear learning problems are considered in the comparison. While a single instance of a

RankSVM may be faster to train than a single instance of RankRLS in the linear learning tasks, the computational shortcuts of RankRLS in cross-validation, parameter selection, and multiple output learning make RankRLS in many situations much faster method to use than RankSVM. This is especially the case if nonlinear kernel functions are used and if cross-validation is used for performance estimation.

We evaluate RankRLS on four tasks with different characteristics. We use as the baseline method RankSVM. The results show that the performance of RankSVM and RankRLS is very similar. Further, the three proposed cost functions are compared with each other and it is found that the performance differences are task dependent. We also show that in all of the experiments RankRLS always performs better or as well as the RLS regressor trained to regress the scores of the input data points. Often some of the pairs of input data points are not relevant with respect to the learning task in question. We show that they may be even harmful to the ranking performance, because the RankRLS algorithm has to minimize their RLS error at the expense of the relevant pairs.

There has been recently discussion within the community about the importance of sharing open source implementations of introduced methods (see Sonnenburg et al. 2007). Inspired by this, we make freely available a software package called RLScore containing an implementation of RankRLS and the efficient cross-validation methods.[5]

Acknowledgements This work has been supported by Academy of Finland and Tekes, the Finnish Funding Agency for Technology and Innovation. We would like to thank CSC, the Finnish IT center for science, for providing us extensive computing resources. We are grateful to Hanna Suominen for her contributions to the document classification experiments. We also thank the anonymous reviewers for their insightful comments.

References

Agarwal, S. (2006). Ranking on graph data. In W. W. Cohen & A. Moore (Eds.), *ACM international conference proceeding series: Vol. 148. Proceedings of the 23rd international conference on machine learning* (pp. 25–32). New York: ACM.

Agarwal, S., & Niyogi, P. (2005). Stability and generalization of bipartite ranking algorithms. In P. Auer & R. Meir (Eds.), *Lecture notes in computer science: Vol. 3559. Proceedings of the 18th annual conference on learning theory* (pp. 32–47). Berlin: Springer.

Bradley, A. P. (1997). The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, *30*(7), 1145–1159.

Brefeld, U., & Scheffer, T. (2005). AUC maximizing support vector learning. In N. Lachiche, C. Ferri, S. A. Macskassy, & A. Rakotomamonjy (Eds.), *Proceedings of the 2nd workshop on ROC analysis in machine learning (ROCML'05)*.

Brualdi, R. A., & Ryser, H. J. (1991). *Combinatorial matrix theory*. Cambridge: Cambridge University Press.

Clémençon, S., Lugosi, G., & Vayatis, N. (2005). Ranking and scoring using empirical risk minimization. In P. Auer & R. Meir (Eds.), *Lecture notes in computer science: Vol. 3559. Proceedings of the 18th annual conference on learning theory* (pp. 1–15). Berlin: Springer.

Cortes, C., Mohri, M., & Rastogi, A. (2007a). An alternative ranking problem for search engines. In C. Demetrescu (Ed.), *Lecture notes in computer science: Vol. 4525. Proceedings of the 6th workshop on experimental algorithms* (pp. 1–21). Berlin: Springer.

Cortes, C., Mohri, M., & Rastogi, A. (2007b). Magnitude-preserving ranking algorithms. In Z. Ghahramani (Ed.), *ACM international conference proceeding series: Vol. 227. Proceedings of the 24th annual international conference on machine learning* (pp. 169–176). New York: ACM.

Freund, Y., Iyer, R., Schapire, R. E., & Singer, Y. (2003). An efficient boosting algorithm for combining preferences. *Journal Machine Learning Research*, *4*, 933–969.

Fürnkranz, J., & Hüllermeier, E. (2005). Preference learning. *Künstliche Intelligenz*, *19*(1), 60–61.

[5] Available at http://www.tucs.fi/RLScore.

Gestel, T. V., Suykens, J. A. K., Baesens, B., Viaene, S., Vanthienen, J., Dedene, G., Moor, B. D., & Vandewalle, J. (2004). Benchmarking least squares support vector machine classifiers. *Machine Learning*, *54*(1), 5–32.

Goldberg, K., Roeder, T., Gupta, D., & Perkins, C. (2001). Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, *4*(2), 133–151.

Golland, P., Liang, F., Mukherjee, S., & Panchenko, D. (2005). Permutation tests for classification. In P. Auer & R. Meir (Eds.), *Lecture notes in computer science: Vol. 3559. Proceedings of the 18th annual conference on learning theory* (pp. 501–515). Berlin: Springer.

Herbrich, R., Graepel, T., & Obermayer, K. (1999). Support vector learning for ordinal regression. In *Proceedings of the ninth international conference on artificial neural networks* (pp. 97–102). London, Institute of Electrical Engineers.

Horn, R., & Johnson, C. R. (1985). *Matrix analysis*. Cambridge: Cambridge University Press.

Huang, J., & Ling, C. X. (2005). Using AUC and accuracy in evaluating learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, *17*(3), 299–310.

Joachims, T. (2002). Optimizing search engines using clickthrough data. In D. Hand, D. Keim, & R. Ng (Eds.), *Proceedings of the 8th ACM SIGKDD conference on knowledge discovery and data mining* KDD'02 (pp. 133–142). New York: ACM.

Joachims, T. (2005). A support vector method for multivariate performance measures. In L. D. Raedt & S. Wrobel (Eds.), *ACM international conference proceeding series: Vol. 119. Proceedings of the 22nd international conference on machine learning* (pp. 377–384). New York: ACM.

Joachims, T. (2006). Training linear SVMs in linear time. In T. Eliassi-Rad, L. H. Ungar, M. Craven, & D. Gunopulos (Eds.), *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining* KDD'06 (pp. 217–226). New York: ACM.

Johnson, R., & Zhang, T. (2008). Graph-based semi-supervised learning and spectral kernel design. *IEEE Transactions on Information Theory*, *54*(1), 275–288.

Liu, T.-Y., Xu, J., Qin, T., Xiong, W., & Li, H. (2007). LETOR: Benchmark dataset for research on learning to rank for information retrieval. In T. Joachims, H. Li, T.-Y. Liu, & C. Zhai (Eds.), *SIGIR 2007 workshop on learning to rank for information retrieval* (pp. 3–10).

Pahikkala, T., Airola, A., Boberg, J., & Salakoski, T. (2008a). Exact and efficient leave-pair-out cross-validation for ranking RLS. In T. Honkela, M. Pöllä, M.-S. Paukkeri, & O. Simula (Eds.), *Proceedings of the 2nd international and interdisciplinary conference on adaptive knowledge representation and reasoning (AKRR'08)* (pp. 1–8). Helsinki University of Technology.

Pahikkala, T., Airola, A., Suominen, H., Boberg, J., & Salakoski, T. (2008b). Efficient AUC maximization with regularized least-squares. In A. Holst, P. Kreuger, & P. Funk (Eds.), *Frontiers in artificial intelligence and applications: Vol. 173. Proceedings of the 10th Scandinavian conference on artificial intelligence* SCAI, 2008 (pp. 12–19). Amsterdam: IOS Press.

Pahikkala, T., Boberg, J., & Salakoski, T. (2006a). Fast n-fold cross-validation for regularized least-squares. In T. Honkela, T. Raiko, J. Kortela, & H. Valpola (Eds.), *Proceedings of the ninth Scandinavian conference on artificial intelligence*, Espoo, Finland (pp. 83–90). Otamedia Oy.

Pahikkala, T., Suominen, H., Boberg, J., & Salakoski, T. (2007a). Transductive ranking via pairwise regularized least-squares. In P. Frasconi, K. Kersting, & K. Tsuda (Eds.), *Workshop on mining and learning with graphs* (pp. 175–178).

Pahikkala, T., Tsivtsivadze, E., Airola, A., Boberg, J., & Salakoski, T. (2007b). Learning to rank with pairwise regularized least-squares. In T. Joachims, H. Li, T.-Y. Liu, C. Zhai (Eds.), *SIGIR 2007 workshop on learning to rank for information retrieval* (pp. 27–33).

Pahikkala, T., Tsivtsivadze, E., Boberg, J., & Salakoski, T. (2006b). Graph kernels versus graph representations: a case study in parse ranking. In T. Gärtner, G. C. Garriga, & T. Meinl (Eds.), *Proceedings of the ECML/PKDD'06 workshop on mining and learning with graphs*, Berlin, Germany (pp. 181–188).

Parker, B. J., Gunter, S., & Bedo, J. (2007). Stratification bias in low signal microarray studies. *BMC Bioinformatics*, *8*, 326.

Poggio, T., & Girosi, F. (1990). Networks for approximation and learning. *Proceedings of the IEEE*, *78*(9), 1481–1497.

Provost, F. J., Fawcett, T., & Kohavi, R. (1998). The case against accuracy estimation for comparing induction algorithms. In J. Shavlik (Ed.), *Proceedings of the fifteenth international conference on machine learning* (pp. 445–453). San Mateo: Morgan Kaufmann.

Pyysalo, S., Ginter, F., Heimonen, J., Björne, J., Boberg, J., Järvinen, J., & Salakoski, T. (2007). BioInfer: A corpus for information extraction in the biomedical domain. *BMC Bioinformatics*, *8*, 50.

Pyysalo, S., Ginter, F., Pahikkala, T., Boberg, J., Järvinen, J., & Salakoski, T. (2006). Evaluation of two dependency parsers on biomedical corpus targeted at protein-protein interactions. *Recent Advances in Natural Language Processing for Biomedical Applications, special issue of the International Journal of Medical Informatics*, *75*(6), 430–442.

Quiñonero-Candela, J., & Rasmussen, C. E. (2005). A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, *6*, 1939–1959.

Quiñonero-Candela, J., Rasmussen, C. E., & Williams, C. K. I. (2007). Approximation methods for Gaussian process regression. In L. Bottou, O. Chapelle, D. DeCoste, & J. Weston (Eds.), *Large-scale kernel machines* (pp. 203–224). Cambridge: MIT Press.

Rakotomamonjy, A. (2004). Optimizing area under ROC curve with SVMs. In J. Hernández-Orallo, C. Ferri, N. Lachiche, & P. A. Flach (Eds.), *Proceedings of the 1st international workshop on ROC analysis in artificial intelligence* (pp. 71–80).

Rifkin, R. (2002). *Everything old is new again: a fresh look at historical approaches in machine learning*. Ph.D. thesis, Massachusetts Institute of Technology.

Rifkin, R., & Klautau, A. (2004). In defense of one-vs-all classification. *Journal of Machine Learning Research*, *5*, 101–141.

Rifkin, R., & Lippert, R. (2007a). *Notes on regularized least squares* (Technical Report MIT-CSAIL-TR-2007-025). Massachusetts Institute of Technology.

Rifkin, R., & Lippert, R. (2007b). Value regularization and Fenchel duality. *Journal of Machine Learning Research*, *8*, 441–479.

Rifkin, R., Yeo, G., & Poggio, T. (2003). Regularized least-squares classification. In J. Suykens, G. Horvath, S. Basu, C. Micchelli, & J. Vandewalle (Eds.), *NATO science series III: computer and system sciences: Vol. 190. Advances in learning theory: methods, model and applications* (pp. 131–154). Amsterdam: IOS Press. Chap. 7.

Schölkopf, B., Herbrich, R., & Smola, A. J. (2001). A generalized representer theorem. In D. Helmbold & R. Williamson (Eds.), *Proceedings of the 14th annual conference on computational learning theory and 5th European conference on computational learning theory* (pp. 416–426). Berlin: Springer.

Schölkopf, B., Mika, S., Burges, C., Knirsch, P., Müller, K.-R., Rätsch, G., & Smola, A. (1999). Input space versus feature space in kernel-based methods. *IEEE Transactions On Neural Networks*, *10*(5), 1000–1017.

Shewchuk, J. R. (1994). *An introduction to the conjugate gradient method without the agonizing pain* (Technical Report CMU-CS-94-125). Carnegie Mellon University, Pittsburgh, PA, USA.

Sleator, D. D., & Temperley, D. (1991). *Parsing English with a link grammar* (Technical Report CMU-CS-91-196). Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA.

Smola, A. J., & Schölkopf, B. (2000). Sparse greedy matrix approximation for machine learning. In P. Langley (Ed.), *Proceedings of the seventeenth international conference on machine learning* (pp. 911–918). San Mateo: Morgan Kaufmann.

Sonnenburg, S., Braun, M. L., Ong, C. S., Bengio, S., Bottou, L., Holmes, G., Lecun, Y., Müller, K. R., Pereira, F., Rasmussen, C. E., Rätsch, G., Schölkopf, B., Smola, A., Vincent, P., Weston, J., & Williamson, R. (2007). The need for open source software in machine learning. *Journal of Machine Learning Research*, *8*, 2443–2466.

Suykens, J. A. K., & Vandewalle, J. (1999). Least squares support vector machine classifiers. *Neural Processing Letters*, *9*(3), 293–300.

Tsivtsivadze, E., Pahikkala, T., Airola, A., Boberg, J., & Salakoski, T. (2008). A sparse regularized least-squares preference learning algorithm. In A. Holst, P. Kreuger, & P. Funk (Eds.), *Frontiers in artificial intelligence and applications: Vol. 173. Proceedings of the 10th Scandinavian conference on artificial intelligence* SCAI, 2008 (pp. 76–83). Amsterdam: IOS Press.

Tsivtsivadze, E., Pahikkala, T., Pyysalo, S., Boberg, J., Mylläri, A., & Salakoski, T. (2005). Regularized least-squares for parse ranking. In A. F. Famili, J. N. Kok, J. M. Peña, A. Siebes, & A. J. Feelders (Eds.), *Lecture notes in computer science: Vol. 3646. Proceedings of the 6th international symposium on intelligent data analysis* (pp. 464–474). Berlin: Springer.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics*, *1*, 80–83.

Zhang, P., & Peng, J. (2004). SVM vs regularized least squares classification. In J. Kittler, M. Petrou, & M. Nixon (Eds.), *Proceedings of the 17th international conference on pattern recognition* ICPR'04 (Vol. 1, pp. 176–179). Los Alamitos: IEEE Computer Society.

Paper II

Training linear ranking SVMs in linearithmic time using red-black trees

Airola, A., Pahikkala, T., and Salakoski, T. (2011). *Pattern Recognition Letters*, 32(9):1328–1336.

Paper III

On learning and cross-validation with decomposed Nyström approximation of kernel matrix

Airola, A., Pahikkala, T., and Salakoski, T. (2011). *Neural Processing Letters*, 33(1):17–30.

Neural Process Lett (2011) 33:17–30
DOI 10.1007/s11063-010-9159-4

On Learning and Cross-Validation with Decomposed Nyström Approximation of Kernel Matrix

Antti Airola · Tapio Pahikkala · Tapio Salakoski

Published online: 1 December 2010

Abstract The high computational costs of training kernel methods to solve nonlinear tasks limits their applicability. However, recently several fast training methods have been introduced for solving linear learning tasks. These can be used to solve nonlinear tasks by mapping the input data nonlinearly to a low-dimensional feature space. In this work, we consider the mapping induced by decomposing the Nyström approximation of the kernel matrix. We collect together prior results and derive new ones to show how to efficiently train, make predictions with and do cross-validation for reduced set approximations of learning algorithms, given an efficient linear solver. Specifically, we present an efficient method for removing basis vectors from the mapping, which we show to be important when performing cross-validation.

Keywords Cross-validation · Empirical kernel map · Kernel methods · Nyström approximation · Reduced set method

1 Introduction

Kernel methods are among the most popular approaches to pattern analysis. They enable solving nonlinear tasks by utilizing a nonlinear mapping from the input space $\mathcal{X}$ to some feature space $\mathcal{F}$, where the task is solved by a linear algorithm. Rather than using the mapping explicitly, kernel methods use a kernel function, which directly calculates, given two inputs, their inner product in the feature space. In addition to nonlinear learning, using kernel

A. Airola (✉) · T. Pahikkala · T. Salakoski
Department of Information Technology, Turku Centre for Computer Science (TUCS),
University of Turku, Joukahaisenkatu 3-5 B, Turku, Finland
e-mail: Antti.Airola@utu.fi

T. Pahikkala
e-mail: Tapio.Pahikkala@utu.fi

T. Salakoski
e-mail: Tapio.Salakoski@utu.fi

functions allows learning from non-vectorial data, such as images, text and graphs, as long as a suitable kernel function can be defined. For a detailed introduction to the topic of kernel methods we refer to [23,24].

Kernel-based learning algorithms, such as the support vector machine (SVM) [2,5], have been designed for a wide variety of tasks, including classification, regression and ranking. Typically for nonlinear learning tasks these methods are trained using dual algorithms which operate directly on the kernel matrix of the training examples (see e.g. [3]). However such methods scale poorly, with respect to the size of the training set.

Recently highly efficient training algorithms have been introduced for the primal case, where the input data resides in an Euclidean space, and the learned decision function is linear [8,25,27]. These methods can also be applied for training nonlinear kernel machines, by mapping the input data nonlinearly to a low dimensional Euclidean space prior to training. Known approaches for doing this include decomposing the kernel matrix [6,22,28,30] or using randomized features for shift invariant kernels [18].

In this work, we show how to recover a reduced set approximation of any kernel based learning algorithm, for which the representer theorem [21] holds, by training a linear learning algorithm on the decomposition of the Nyström approximation of kernel matrix, and applying a linear transformation on the learned weight vector. This is a straightforward consequence of previously known results but is not, we believe, well known in general. The considered approach yields prediction functions which use only a subset of the training examples, when making predictions on new examples. Previously equivalent reduced set methods have been proposed for a large variety of loss functions [10,11,16,17,19,26,28,29].

Further, we introduce a novel method for efficient recalculation of the Nyström factorization for the case where basis vectors are removed. This allows efficient cross-validation for the reduced set approximation, as long as there is an efficient primal solver available. Cross-validation with reduced set methods was previously considered by Cawley and Talbot [4], who introduced a fast leave-one-out algorithm for reduced set approximation of the regularized least squares (RLSs) algorithm, also known as least squares SVM [16,19,28]. However, their method did not remove basis vectors belonging to the holdout set, an approach which we will show to be problematic. A version of this cross-validation algorithm which takes care of this issue for the reduced set approximation of RLS has been introduced by Pahikkala et al. [14]. In this work, we generalize the result to arbitrary loss functions with quadratic regularization.

2 Low-dimensional Feature Map for Regularized Risk Minimization

In this section, we show how to efficiently train a reduced set approximation of a kernel method using a primal solver. This is based on three key observations. First, a kernel machine can be trained by training a linear learner on the decomposition of a kernel matrix [22,30]. Second, using the Nyström approximation corresponds to using a certain type of data dependent kernel function [17]. Third, one can transform the prediction function learned using this data dependent kernel to a one which uses only a subset of training examples for making predictions.

We use $\mathcal{X}$ to denote the input space, which can be any set. The sequence $X = (x_1, \ldots, x_m) \in (\mathcal{X}^m)^{\mathrm{T}}$ is used to denote the training inputs. By $Y = (y_1, \ldots, y_m) \in (\mathcal{Y}^m)^{\mathrm{T}}$ we denote the sequence of correct outputs corresponding to the training inputs.

Following type of notation is used for describing sliced matrices. Let us consider a matrix D, that has its rows indexed by a superset of an index set F and its columns indexed by a

superset of an index set G. Then by D_{FG} we denote a matrix containing only those elements of D whose row index is in F and column index in G.

By $\mathrm{D}^{1/2}$ we denote a symmetric decomposition of D, that is $\mathrm{D}^{1/2}(\mathrm{D}^{1/2})^{\mathrm{T}} = \mathrm{D}$. The decomposition matrix has at least as many columns as is the rank of D. D^{+} denotes the Moore–Penrose pseudoinverse of D. We recall the following properties of the Moore–Penrose pseudoinverse D^{+} of a given matrix D. First, $\mathrm{D}^{+} = \mathrm{D}^{-1}$, if D is nonsingular. Second, $\mathrm{D}^{\mathrm{T}} = \mathrm{D}^{+}\mathrm{D}\mathrm{D}^{\mathrm{T}}$. Third, if D has full column rank and E full row rank, $(\mathrm{DE})^{+} = \mathrm{E}^{+}\mathrm{D}^{+}$. Finally, if D has full row rank, then $\mathrm{D}\mathrm{D}^{+} = \mathrm{I}$

2.1 Empirical Kernel Map for the Nyström Approximation

Let $\mathcal{F}$ denote an inner product space known as the feature space. For any mapping

$$\Phi : \mathcal{X} \to \mathcal{F},$$

the inner product

$$k(x, x') = \langle \Phi(x), \Phi(x') \rangle$$

of the mapped inputs is called a kernel function. In the following we restrict ourselves to consider only real valued inner products.

For a given input sequence X, a kernel function gives rise to a symmetric matrix

$$\mathrm{K} = \begin{pmatrix} k(x_1, x_1) & \cdots & k(x_1, x_m) \\ \vdots & \ddots & \vdots \\ k(x_m, x_1) & \cdots & k(x_m, x_m) \end{pmatrix}$$

which is known as the kernel matrix of X. The kernel matrix is positive semidefinite, that is, $\mathbf{a}^{\mathrm{T}}\mathrm{K}\mathbf{a} \geq 0$ for all $\mathbf{a} \in \mathbb{R}^m$.

Though the feature space may have infinite dimension, the subspace where a finite sequence of inputs lies has finite dimension. Thus as a notational convenience we represent mapped inputs as finite dimensional vectors. We overload our notation by defining $\Phi(X) = (\Phi(x_1), \ldots, \Phi(x_m))$ for any given sequence X of inputs. The corresponding kernel matrix can be expressed as $\mathrm{K} = \Phi(X)^{\mathrm{T}}\Phi(X)$.

Even though the feature map Φ exists for any given positive definite kernel function, in practice this map is rarely explicitly used when the kernel is nonlinear. The map is often onto a very high dimensional, or even infinite dimensional space, or one may not know what the map is. Instead we consider the empirical kernel map.

By overloading our notation, we write $k(x, X) = (k(x, x_1), \ldots, k(x, x_m)) \in (\mathbb{R}^m)^{\mathrm{T}}$, where $x \in \mathcal{X}$. Following Schölkopf et al. [22], we define the empirical kernel map with respect to X as

$$\Phi_X : \mathcal{X} \to \mathbb{R}^r,$$

where

$$\Phi_X(x) = (\mathrm{K}^{1/2})^{+}(k(x, X))^{\mathrm{T}}. \tag{1}$$

In practice r would usually correspond to the rank of the kernel matrix, though one can define such empirical kernel maps for which it is larger. Following Xiong et al. [32] we call the space $\mathbb{R}^r$ the empirical feature space. This space is an Euclidean space, where the inner product is the standard dot product.

Now, let us consider the empirical kernel map of the inputs used in defining the mapping. According to (1)

$$\begin{aligned}\Phi_X(X) &= \left(\mathrm{K}^{1/2}\right)^+ \mathrm{K} \\ &= \left(\mathrm{K}^{1/2}\right)^+ \mathrm{K}^{1/2} \left(\mathrm{K}^{1/2}\right)^{\mathrm{T}} \\ &= \left(\mathrm{K}^{1/2}\right)^{\mathrm{T}}.\end{aligned}$$

It is easy to see that $\Phi_X(X)^{\mathrm{T}}\Phi_X(X) = \mathrm{K}$, and thus for all examples in X the inner products are the same in the empirical feature space, as in the feature space. The inner product fully determines distances between these examples, and their norms. Due to the linearity of the real valued inner product this result extends to any linear combinations of these examples.

What this result in practice means is that any learning algorithm that operates within the linear span of the training examples can be trained in a finite dimensional Euclidean space using the empirical kernel map of the data. This is particularly useful for algorithms based on regularized risk minimization with a quadratic regularizer, as they provably find their optimal solution within this span. We will discuss this issue further in the next section.

The approach of training kernel machines using the decomposition of the kernel matrix and a primal solver was used by Tsuda [30] for SVM optimization. However, in practice the approach has not been very popular due to computational burdens inherent in the method. Assuming a training set of m examples the computational cost of decomposing the kernel matrix is $O(m^3)$ and the memory complexity of storing the matrix is $O(m^2)$. Further, the resulting mapping is high dimensional having m features, meaning that any subsequent method trained on the data has at least $O(m^2)$ complexity if it passes at least once through each mapped example. Such complexities provide no improvement over the existing dual solvers.

In the machine learning field one of the most popular and successful techniques for finding an approximative low-rank decomposition for the kernel matrix is the Nyström approximation [31]. Let us consider the index set $M = \{1, \ldots m\}$ of the training set X. We choose a subset of training examples indexed by $R \subseteq M$, such that $|R| << |M|$. We call this subset the basis vectors. Then the Nyström approximation of K is

$$\widetilde{\mathrm{K}} = \mathrm{K}_{MR}(\mathrm{K}_{RR})^{-1}\mathrm{K}_{RM}. \tag{2}$$

Let $\mathrm{CC}^{\mathrm{T}} = \mathrm{K}_{RR}$ denote the cholesky decomposition of K_{RR}. Instead of explicitly constructing $\widetilde{\mathrm{K}}$, we can calculate its decomposition from $(\widetilde{\Phi}_X(X))^{\mathrm{T}} = \mathrm{K}_{MR}(\mathrm{C}^{\mathrm{T}})^{-1}$. The cost of calculating C^{-1} is $O(|R|^3)$ and the cost of the matrix multiplication $O(m|R|^2)$. The memory complexity is $O(m|R|)$, and the resulting mapping has only $|R|$-dimensions. This means that for small $|R|$ the method scales substantially better than the exact decomposition. As we will discuss next, it turns out that applying this mapping on the training data and using a regularized risk minimization based learner for subsequent training is equivalent to using reduced set type of sparse approximation algorithms, sometimes known as subset of regressors, proposed in the literature [16,17,19,26,29]. This approach is also quite similar, although not equivalent, to the sparse approximative empirical kernel map proposed by Abe [1] for fast RLS training.

An interesting result from [17] is that using the Nyström approximation is equivalent to using the following type of data dependent kernel function

$$\tilde{k}(x, x') = k(x, X_R)(\mathrm{K}_{RR})^{-1}k(x', X_R)^{\mathrm{T}}, \tag{3}$$

where X_R denotes the set of training examples indexed by R. Thus, the decomposition $\widetilde{\Phi}_X(X)$ of $\widetilde{\mathrm{K}}$ is an empirical kernel map corresponding to the feature space implicitly defined by the modified kernel function $\tilde{k}$.

The selection of basis vectors affects how well $\widetilde{\mathrm{K}}$ approximates K. Proposed approaches for selection include both adaptive methods such as a k-means clustering based method [33], as well as non-adaptive ones. While the adaptive data dependent approaches can perform better, Rifkin et al. [19] and Kumar et al. [9] criticize many of these for being computationally heavy. Both recommend uniform sampling without replacement, as the method is fast and according to Kumar et al. [9] can also be expected to yield superior results compared to non-uniform sampling distributions.

2.2 Regularized Risk Minimization and Reduced Set Methods

A large class of machine learning algorithms are based on minimizing the regularized risk functional

$$J(X, Y, f) = c(X, Y, f(X)) + \lambda \|f\|_k^2, \tag{4}$$

where $\lambda > 0$ is a parameter and f belongs to a reproducing kernel Hilbert space of functions (RKHS). The first term in (4) is the empirical risk measuring how well the considered hypothesis f fits the training data, as measured by some cost function c. The second term called the regularizer measures the complexity of the hypothesis with the RKHS norm.

According to the generalized representer theorem [21] any minimizer of (4) admits a representation of the form

$$f(x) = \sum_{i=1}^{m} a_i k(x, x_i) = k(x, X)\mathbf{a}, \tag{5}$$

where $a_i \in \mathbb{R}$, $\mathbf{a} = (a_1, \ldots, a_m)^{\mathrm{T}} \in \mathbb{R}^m$ and k is the kernel function associated with the RKHS mentioned above. This representation allows us to express the solution using only kernel evaluations between the argument and the training examples. We call this the dual formulation and $\mathbf{a}$ the dual solution.

The minimizer (5) can be re-written as

$$\begin{aligned} f(x) &= \sum_{i=1}^{m} a_i \langle \Phi(x), \Phi(x_i) \rangle \\ &= \left\langle \Phi(x), \sum_{i=1}^{m} a_i \Phi(x_i) \right\rangle \\ &= \langle \Phi(x), \mathbf{w} \rangle. \end{aligned}$$

We call $\mathbf{w} = \Phi(X)\mathbf{a}$ the primal solution. Conversely, given the primal solution $\mathbf{w}$, the dual solution can be recovered from

$$\mathbf{a} = \Phi(X)^{+}\mathbf{w}. \tag{6}$$

The consistency of the system of linear equations involved in solving (6) is guaranteed by the representer theorem. Thus, $\mathbf{a}$ can be recovered using the Moore–Penrose pseudoinverse of $\Phi(X)$ [12].

Depending on whether we solve the primal problem directly in the (empirical) feature space, or its dual counterpart, we can re-formulate (4) either as

$$J(X, Y, \mathbf{w}) = c(X, Y, \Phi(X)^{\mathrm{T}}\mathbf{w}) + \lambda \mathbf{w}^{\mathrm{T}}\mathbf{w},$$

or as

$$J(X, Y, \mathbf{a}) = c(X, Y, \mathrm{K}\mathbf{a}) + \lambda \mathbf{a}^{\mathrm{T}}\mathrm{K}\mathbf{a}. \tag{7}$$

Now let us consider the dual objective function, when using the modified kernel function $\tilde{k}$ corresponding to the Nyström approximation. Our analysis is similar to that applied by Rifkin et al. [19] to show the equivalence of the sparse approximation and the Nyström approach for the RLS algorithm. Let $\widetilde{\mathbf{a}}$ be the minimizer of (7), when using the modified kernel matrix $\widetilde{\mathrm{K}}$ constructed from the training set.

Let us define

$$\mathbf{b} = (\mathrm{K}_{RR})^{-1}\mathrm{K}_{RM}\widetilde{\mathbf{a}}, \mathbf{b} \in \mathbb{R}^{|R|\times 1}.$$

We note that

$$\begin{aligned}\widetilde{\mathrm{K}}\widetilde{\mathbf{a}} &= \mathrm{K}_{MR}(\mathrm{K}_{RR})^{-1}\mathrm{K}_{RM}\widetilde{\mathbf{a}} \\ &= \mathrm{K}_{MR}\mathbf{b}.\end{aligned}$$

Similarly,

$$\begin{aligned}\widetilde{\mathbf{a}}^{\mathrm{T}}\widetilde{\mathrm{K}}\widetilde{\mathbf{a}} &= \widetilde{\mathbf{a}}^{\mathrm{T}}\mathrm{K}_{MR}(\mathrm{K}_{RR})^{-1}\mathrm{K}_{RM}\widetilde{\mathbf{a}} \\ &= \widetilde{\mathbf{a}}^{\mathrm{T}}\mathrm{K}_{MR}(\mathrm{K}_{RR})^{-1}\mathrm{K}_{RR}(\mathrm{K}_{RR})^{-1}\mathrm{K}_{RM}\widetilde{\mathbf{a}} \\ &= \mathbf{b}^{\mathrm{T}}\mathrm{K}_{RR}\mathbf{b}.\end{aligned}$$

Substituting these into (7) we can define an equivalent objective function

$$J(X, Y, \mathbf{b}) = c(X, Y, \mathrm{K}_{MR}\mathbf{b}) + \lambda \mathbf{b}^{\mathrm{T}}\mathrm{K}_{RR}\mathbf{b}.$$

which is the objective function of the reduced set sparse approximation considered in [16, 17, 19, 26, 29].

This means that for the Nyström approximation, instead of using all the m training examples to represent the learned hypothesis, as in (5), we can limit ourselves to considering only the basis vectors. The optimal solution is of the form,

$$f(x) = \sum_{i\in R} a_i k(x, x_i) = k(x, X_R)\mathbf{b}. \tag{8}$$

Based on previous considerations the dual reduced set solution $\mathbf{b}$ can be recovered from the solution $\widetilde{\mathbf{w}}$ in the empirical feature space using a linear transformation as follows

$$\begin{aligned}\mathbf{b} &= (\mathrm{K}_{RR})^{-1}\mathrm{K}_{RM}\widetilde{\mathbf{a}} \\ &= (\mathrm{K}_{RR})^{-1}\mathrm{K}_{RM}\widetilde{\Phi}_X(X)^{+}\widetilde{\mathbf{w}} \\ &= (\mathrm{K}_{RR})^{-1}\mathrm{K}_{RM}(\mathrm{K}_{RM})^{+}\mathrm{C}\widetilde{\mathbf{w}} \\ &= (\mathrm{C}^{\mathrm{T}})^{-1}\widetilde{\mathbf{w}}.\end{aligned}$$

Thus, the dual coefficients can be recovered in $O(|R|^2)$ time, as we already know $(\mathrm{C}^{\mathrm{T}})^{-1}$. New predictions are calculated as defined in (8) with the weighted sum of $|R|$ kernel evaluations between the basis vectors and the new example.

3 Efficient Removal of Basis Vectors for Cross-validation

Cross-validation is one of the most commonly used tools for model selection and performance estimation. However, with the data dependent kernel defined for the sparse approximation, cross-validation becomes complicated. From (3) it can be seen that the basis vectors have an influence on the kernel function itself. The removal of any of these from the training set will result in a change in the kernel function. Next we explore this problem in more detail, and provide a computationally efficient solution for doing cross-validation.

Let us consider the index set M of the training set X. When performing cross validation, on each round one chooses a holdout set $U \subset M$ and removes the examples indexed by it from the training set. The the learner is trained on the examples indexed by $\overline{U} = M \setminus U$, and tested on the holdout examples.

If $R \cap U = \emptyset$, that is, none of the basis vectors belong to the holdout set, this is very straightforward to do. One can simply use

$$\left(\left(\widetilde{\mathrm{K}}^{1/2}\right)^{\mathrm{T}}\right)_{\overline{U}M} \tag{9}$$

for training the learner, as this is a valid decomposition of $\widetilde{\mathrm{K}}_{\overline{U}\overline{U}}$, which is the kernel matrix for the complement of the holdout set. Given the primal solution $\mathbf{w}$ predictions for the holdout examples can be directly calculated from $\left(\left(\widetilde{\mathrm{K}}^{1/2}\right)^{\mathrm{T}}\right)_{UM}$, as we have the empirical kernel map of the holdout data in it.

However, if $R \cap U \neq \emptyset$, the holdout procedure becomes more complicated. We could deal with this situation by just ignoring it and use (9) as before. This would conceptually mean selecting some of the basis vectors from the test set. However, this can provide unreliable performance estimates as we will show in Sect. 4.1. Another approach for circumventing this problem is, of course, to perform the cross-validation so that the training examples which are selected to be basis vectors are not held out in any cross-validation round. With large data sets, this approach usually provides reliable enough performance estimates. However, there are certain tasks in which this approach is not reliable due to dependencies between the examples in the training set. For a typical example of such a situation, see, for example, the experiments in [13,20]. Such dependencies are also present in the experimental setting of Sect. 4, and hence neither of the above presented simple approaches can be used there.

Next, we present an approach which enables the efficient removal of the effect of the hold-out basis vectors from the empirical kernel map. Let us consider the index set R of the basis vectors. Let $H = R \cap U$ and $L = R \setminus H$. Thus, H contains the basis vectors which are included in the holdout set, and L the rest of the basis vectors. Let us recall that

$$\widetilde{\mathrm{K}} = \mathrm{K}_{MR}(\mathrm{K}_{RR})^{-1}\mathrm{K}_{RM},$$

and $\mathrm{C} = (\mathrm{K}_{RR})^{1/2}$, that is, $\mathrm{K}_{RR} = \mathrm{C}\mathrm{C}^{\mathrm{T}}$.

The most straightforward approach for removing the effect of holdout basis vectors would be to calculate $(\mathrm{K}_{LL})^{-1/2}$ in $O(|R|^3)$ time, and $\mathrm{K}_{ML}(\mathrm{K}_{LL})^{-1/2}$ in $O(m|R|^2)$ time. This is however almost as expensive as calculating the original decomposition was. Given that the most efficient available linear learners have training complexities of the order $O(m|R|)$ for many types of tasks [8,25,27], this would mean that the cost of recalculating the decompositions would form a limiting computational bottleneck for performing cross-validation. However, given that we have previously calculated C^{-1} and $\mathrm{K}_{MR}(\mathrm{C}^{-1})^{\mathrm{T}}$, we can do better. In the following proof we will make use of the block inverse lemma, and the matrix inversion lemma, also known as Sherman–Morrison–Woodbury formula. Readers unfamiliar with these results, or wanting to refresh their memory, may check them from the appendix.

Theorem 1 *The complexity of calculating* $\mathrm{K}_{ML}(\mathrm{K}_{LL})^{-1/2}$ *is* $O(|H||R|m)$*, given* C^{-1} *and* $\mathrm{K}_{MR}(\mathrm{C}^{-1})^T$

Proof First, let us define $\mathrm{Z} = \mathrm{K}_{ML}((\mathrm{C}_{LL})^{\mathrm{T}})^{-1}$ and as a short hand notation write $\mathrm{G} = (\mathrm{C}^{\mathrm{T}})^{-1}$. We can calculate Z in $O(|H||R|m)$ time as follows.

$$\begin{aligned}
\mathrm{Z} &= \mathrm{K}_{ML}\left((\mathrm{C}_{LL})^{\mathrm{T}}\right)^{-1} \\
&= \mathrm{K}_{ML}\left(\mathrm{G}_{LL} - \mathrm{G}_{LH}\left(\mathrm{G}_{HH}\right)^{-1}\mathrm{G}_{HL}\right) \\
&= \mathrm{K}_{ML}\mathrm{G}_{LL} - \mathrm{K}_{ML}\mathrm{G}_{LH}\left(\mathrm{G}_{HH}\right)^{-1}\mathrm{G}_{HL} \\
&= \left(\mathrm{K}_{MR}\mathrm{G}\right)_{ML} - \mathrm{K}_{MH}\mathrm{G}_{HL} - \mathrm{K}_{ML}\mathrm{G}_{LH}\left(\mathrm{G}_{HH}\right)^{-1}\mathrm{G}_{HL}.
\end{aligned}$$

The second equality is due to the block inverse lemma (Lemma 4, Appendix). The first term on the last line can be efficiently calculated simply by slicing the columns indexed by L from $\mathrm{K}_{MR}(\mathrm{C}^{\mathrm{T}})^{-1}$, which we already have pre-computed in memory. The matrix products in the other terms can be performed in $O(|H||L|m) = O(|H||R|m)$ time.

Let $\mathrm{A} = (\mathrm{C}_{LL})^{-1}\mathrm{C}_{LH}$.

$$\begin{aligned}
(\mathrm{K}_{LL})^{-1} &= \left(\mathrm{C}_{LR}(\mathrm{C}_{LR})^{\mathrm{T}}\right)^{-1} \\
&= \left(\mathrm{C}_{LL}(\mathrm{C}_{LL})^{\mathrm{T}} + \mathrm{C}_{LH}\left(\mathrm{C}_{LH}\right)^{\mathrm{T}}\right)^{-1} \\
&= \left((\mathrm{C}_{LL})^{\mathrm{T}}\right)^{-1}(\mathrm{C}_{LL})^{-1} - \left((\mathrm{C}_{LL})^{\mathrm{T}}\right)^{-1}\mathrm{A}\left(\mathrm{I} + \mathrm{A}^{\mathrm{T}}\mathrm{A}\right)^{-1}\mathrm{A}^{\mathrm{T}}\left(\mathrm{C}_{LL}\right)^{-1} \\
&= \left((\mathrm{C}_{LL})^{\mathrm{T}}\right)^{-1}\left[\mathrm{I} - \mathrm{A}(\mathrm{I} + \mathrm{A}^{\mathrm{T}}\mathrm{A})^{-1}\mathrm{A}^{\mathrm{T}}\right](\mathrm{C}_{LL})^{-1},
\end{aligned}$$

where the third equality is due to the matrix inversion lemma (Lemma 5, Appendix). Let $\mathrm{B} = \mathrm{A}(\mathrm{I} + \mathrm{A}^{\mathrm{T}}\mathrm{A})^{-1/2} \in \mathbb{R}^{|L|\times|H|}$. It follows from the block inverse lemma that we can rewrite A as $\mathrm{A} = -(\mathrm{C}^{-1})_{LH}((\mathrm{C}^{-1})_{HH})^{-1}$. Therefore, we can calculate B in $O(|L||H|^2)$ time, since we already know C^{-1}.

Moreover, let $\mathrm{BB}^{\mathrm{T}} = \mathrm{V}\Lambda\mathrm{V}^{\mathrm{T}}$ be the eigenvalue decomposition of BB^{T}, where $\Lambda \in \mathbb{R}^{|L|\times|L|}$ is a diagonal matrix containing the eigenvalues of BB^{T}, and $\mathrm{V} \in \mathbb{R}^{|L|\times|L|}$ contains the corresponding eigenvectors. V is an orthogonal matrix, that is $\mathrm{VV}^{\mathrm{T}} = \mathrm{V}^{\mathrm{T}}\mathrm{V} = \mathrm{I}$. Because the rank of BB^{T} is at most $|H|$, Λ contains at most $|H|$ non-zero eigenvalues. The eigenvalue decomposition can be calculated for example from the singular value decomposition of B, which takes $O(|H||L|^2)$ time to compute.

We can write the Nyström approximation which uses only the examples indexed by L as basis vectors as

$$\begin{aligned}
\widetilde{\mathrm{K}} &= \mathrm{K}_{ML}\left(\mathrm{K}_{LL}\right)^{-1}\mathrm{K}_{LM} \\
&= \mathrm{Z}\left(\mathrm{I} - \mathrm{BB}^{\mathrm{T}}\right)\mathrm{Z}^{\mathrm{T}} \\
&= \mathrm{Z}\left(\mathrm{I} - \mathrm{V}\Lambda\mathrm{V}^{\mathrm{T}}\right)\mathrm{Z}^{\mathrm{T}} \\
&= \mathrm{ZV}\left(\mathrm{I} - \Lambda\right)\mathrm{V}^{\mathrm{T}}\mathrm{Z}^{\mathrm{T}} \\
&= \mathrm{ZV}\left(\sqrt{\mathrm{I} - \Lambda}\sqrt{\mathrm{I} - \Lambda}\right)\mathrm{V}^{\mathrm{T}}\mathrm{Z}^{\mathrm{T}} \\
&= \mathrm{ZV}\left(\sqrt{\mathrm{I} - \Lambda} - \mathrm{I} + \mathrm{I}\right)\mathrm{V}^{\mathrm{T}}\mathrm{V}\left(\sqrt{\mathrm{I} - \Lambda} - \mathrm{I} + \mathrm{I}\right)\mathrm{V}^{\mathrm{T}}\mathrm{Z}^{\mathrm{T}} \\
&= \mathrm{Z}\left[\mathrm{V}\left(\sqrt{\mathrm{I} - \Lambda} - \mathrm{I}\right)\mathrm{V}^{\mathrm{T}} + \mathrm{VV}^{\mathrm{T}}\right]\left[\mathrm{V}\left(\sqrt{\mathrm{I} - \Lambda} - \mathrm{I}\right)\mathrm{V}^{\mathrm{T}} + \mathrm{VV}^{\mathrm{T}}\right]^{\mathrm{T}}\mathrm{Z}^{\mathrm{T}} \\
&= \mathrm{Z}\left[\mathrm{V}\left(\sqrt{\mathrm{I} - \Lambda} - \mathrm{I}\right)\mathrm{V}^{\mathrm{T}} + \mathrm{I}\right]\left[\mathrm{V}\left(\sqrt{\mathrm{I} - \Lambda} - \mathrm{I}\right)\mathrm{V}^{\mathrm{T}} + \mathrm{I}\right]^{\mathrm{T}}\mathrm{Z}^{\mathrm{T}}
\end{aligned}$$

Thus, the empirical kernel map corresponding to the hold-out computation is $\mathrm{Z}[\mathrm{V}\left(\sqrt{\mathrm{I}-\Lambda}-\mathrm{I}\right)\mathrm{V}^{\mathrm{T}}+\mathrm{I}]$.

Let $\hat{\Lambda} \in \mathbb{R}^{|H|\times|H|}$ be a diagonal matrix containing the non-zero values in Λ and $\hat{\mathrm{V}} \in \mathbb{R}^{|L|\times|H|}$ the matrix containing the corresponding eigenvalues. Then $\mathrm{Z}[\mathrm{V}\left(\sqrt{\mathrm{I}-\Lambda}-\mathrm{I}\right)\mathrm{V}^{\mathrm{T}}+\mathrm{I}] = \mathrm{Z}[\hat{\mathrm{V}}\left(\sqrt{\mathrm{I}-\hat{\Lambda}}-\mathrm{I}\right)\hat{\mathrm{V}}^{\mathrm{T}}+\mathrm{I}]$. These matrix products can be calculated in $O(|H||L|m) = O(|H||R|m)$ time, which was also the complexity of calculating Z. Thus, the complexity of calculating $\mathrm{K}_{ML}(\mathrm{K}_{LL})^{-1/2}$ is $O(|H||R|m)$. □

Corollary 2 *The complexity of calculating the empirical kernel maps for n-fold cross-validation is $O(|R|^2 m)$, when using the kernel map corresponding to the Nyström approximation*

Proof First, the complexity of calculating the empirical kernel map of the Nyström approximation for the whole training set is $O(|R|^2 m)$. Second, the average number of hold-out examples belonging to the set of basis vectors in each cross-validation round is $|H| = |R|/n$. Thus, the complexity of removing the basis vectors on the n rounds of cross-validation is $O(n|H||R|m) = O(n(|R|/n)|R|m) = O(|R|^2 m)$. Once the effect of holdout basis vectors has been removed, an empirical kernel map for the holdout set and its complement can be recovered by slicing the corresponding rows from the mapping, as discussed before. □

4 Computer Simulations

In the simulations we consider the removal of holdout basis vectors during cross-validation. We explore two questions. First, to what extent is the cross-validation estimate affected by the choice of whether the holdout basis vectors are removed or not. Second, whether the algorithm we introduced for removing holdout basis vectors is in practice efficient. We refer to the relevant literature for results about the quality of the reduced set approximation when applied to different loss functions [10,11,16,17,19,26,29].

Our methods were implemented using the Python programming language, and the NumPy and SciPy programming libraries. All simulations were run on a modern desktop computer.

We consider a setting similar to that of Tsivtsivaze et al. [29], where the ranking performance of the reduced set based sparse approximation of the RankRLS algorithm [15] was evaluated. Training the primal version of the RankRLS algorithm using the feature map produced by decomposing $\bar{\mathrm{K}}$ would lead to an equivalent method.

The experiments were motivated by a task from the field of natural language processing. Given a sentence and a set of candidate parses generated by an automatic parser, the task is to order the parses according to how correctly they describe the syntactic structure of the sentence. The training dataset consists of approximately 600 sentences each having 20 or less candidate parses, yielding in total a set of 9856 examples. Each candidate parse is associated with a goodness score describing how close it is to the correct parse of the sentence. Following the setup of Tsivtsivadze et al. [29] we choose basis vectors from each sentence, and use a graph kernel for the feature representation of the parses.

The kernel function has the property that two parses originating from a same sentence have almost always larger mutual similarity than two parses originating from different sentences. This causes dependencies between the examples, which is something we have to take into account in designing the cross-validation experiments. Namely, we have to ensure that all parses originating from the same sentence are held out simultaneously, since it would be unrealistic to assume that both the training and test set would have parses originating from the same sentence. In our experiments, we perform the cross-validation on the sentence level

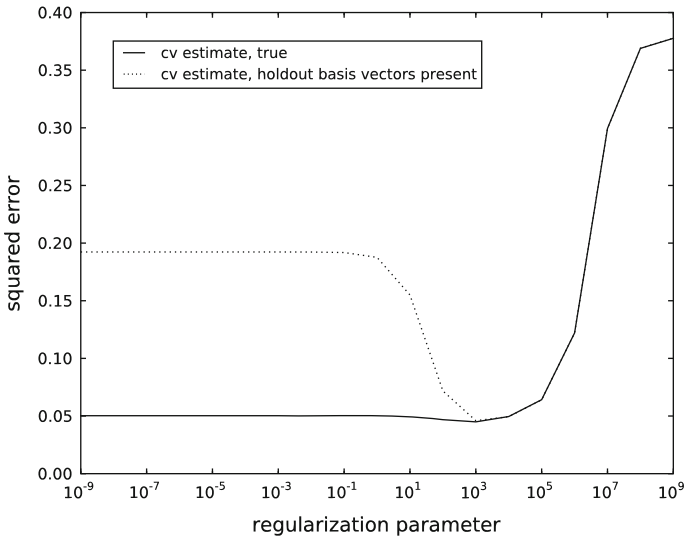

Fig. 1 Comparison between cross-validation estimates with and without removal of holdout basis vectors

so that all the parses generated from a sentence would always be either in the training set or in the test set. The question still left open is whether we also have to remove the effect of a basis vector from the kernel map if the corresponding example belongs to the hold-out set. Next, we show the necessity of this.

4.1 The Effect of Keeping Holdout Basis Vectors

First, we explore to which extent not removing the basis vectors belonging to the holdout set biases the cross-validation estimate. In this experiment, we fix the number of basis vectors to two per sentence, and perform 10-fold cross-validation, where the sentences are randomly divided to 10 mutually disjoint folds. On each round we train an RLS regressor to regress the parse goodness scores. We compare the method which removes the basis vectors present in the holdout set to an approach where they are not removed. The used performance measure is mean squared error. A regularization parameter grid $10^{-9} \ldots 10^{9}$ is explored.

The results are in Fig. 1. The approach where the basis vectors belonging to the holdout set are not removed shows a high degree of pessimism for low regularization parameter values, predicting almost four times higher error than the correct approach. This can lead to incorrect performance estimation, or to selecting suboptimal parameters during model selection, suggesting that it is important to remove the holdout basis vectors during cross-validation. However, when the level of regularization is increased enough the differences between the two estimates disappear.

4.2 Computational Efficiency

In the second set of experiments, we test the computational efficiency of the algorithm we introduced. We compare this method to the more straightforward approach of recalculating the whole feature map every time basis vectors are included in the holdout set. The

Table 1 Runtime comparisons in CPU seconds for leave-sentence-out cross-validation

	Size of $\|R\|$			
	598	1181	1742	2297
Initial decomposition	6	28	63	123
Efficient CV	532	1193	1864	2679
Baseline CV	3639	16648	38373	73375

Table 2 Runtime comparisons in CPU seconds for 10-fold cross-validation

	Size of $\|R\|$			
	598	1181	1742	2297
Initial decomposition	6	28	63	123
Efficient CV	33	112	234	408
Baseline CV	49	226	512	937

computational complexity of n-fold cross-validation is for the former approach $O(|R|^2 m)$ suggesting that it should significantly outperform the latter approach whose complexity is $O(n|R|^2 m)$. It is however not self-evident that this is the case for typical choices of m, $|R|$ and n as asymptotic upper bounds can hide large constant terms which may in practice have a large impact on performance.

We vary the value of chosen basis vectors per sentence from 1 to 4 to explore the effect of the size of R on the computational efficiency.

The ranking performance is typically calculated as an average over per sentence ranking performances. This suggests a natural cross-validation scheme for performance evaluation, the leave-sentence-out cross-validation. On each cross validation round the holdout set consists of the candidate parses of one of the sentences. Similar settings are typically encountered in information retrieval where machine learning methods are trained on sets of user queries paired with documents to be ranked.

In Table 1 is the runtime comparison of the efficient cross-validation method and the baseline approach. The cost of performing the initial decomposition for calculating the feature map for the whole dataset is presented separately. The results demonstrate that the efficient method based on computational shortcuts is much faster, suggesting that implementing it is worthwhile when doing cross-validation with a large number of folds.

In practice, due to computational constraints 10-fold cross-validation is more often used than the leave-sentence-out considered previously. The 10-fold estimate is more pessimistically biased due to a larger part of the training set being unused in each round of cross-validation. Still, for large datasets it can be expected to yield good estimate of performance.

In Table 2 is the runtime comparison for 10-fold cross-validation. The folds have been defined on the level of sentences, so that all the parses related to the same sentence belong to the same fold. The efficient method is about two times faster than the baseline. This suggests that the method is also useful for speeding up cross-validation when the number of folds is relatively small, but the more straightforward baseline approach is in this setting also competitive.

5 Conclusion

Motivated by the recent developments in efficient linear learning methods we have considered how nonlinear learning problems can be transformed to linear ones by using empirical kernel map of the Nyström approximation. A straightforward matrix decomposition approach allows one to derive a reduced set approximation of any regularized risk minimization algorithm that uses quadratic regularization. However, this mapping must be recalculated whenever examples that are basis vectors are removed from the training set. We derive a computational short cut for the removal of basis vectors. Our experiments show that removal of holdout basis vectors is necessary for achieving reliable cross-validation estimates, and that our method can do this efficiently. In our future work, we will explore other settings where the modification of the set of basis vectors is necessary.

Acknowledgements This work has been supported by the Academy of Finland.

Appendix

Here, we consider some well-known properties of block matrices. Let D be a square matrix, whose rows and columns are indexed by an index set J. Let us divide the indices into two disjoint subsets, $F \subset J$, and $G = J \setminus F$. Without losing generality, we can write D as a following block matrix

$$\mathrm{D} = \begin{bmatrix} \mathrm{D}_{FF} & \mathrm{D}_{FG} \\ \mathrm{D}_{GF} & \mathrm{D}_{GG} \end{bmatrix}. \tag{10}$$

First, we present a lemma that is a direct consequence of Schur's determinantal formula. (see e.g. [7, p. 21]).

Lemma 3 *If* D_{GG} *is invertible, then* D *is invertible if and only if*

$$\overline{\mathrm{S}} = \mathrm{D}_{FF} - \mathrm{D}_{FG}(\mathrm{D}_{GG})^{-1}\mathrm{D}_{GF}.$$

is invertible. Here, $\overline{\mathrm{S}}$ *is the Schur complement of* D_{GG}.

Next, we present a lemma that is known as the block inverse.

Lemma 4 *Assume* D *to be invertible. If* D_{GG} *is invertible, then the inverse matrix of* D *is*

$$\mathrm{D}^{-1} = \begin{bmatrix} (\mathrm{D}^{-1})_{FF} & (\mathrm{D}^{-1})_{FG} \\ (\mathrm{D}^{-1})_{GF} & (\mathrm{D}^{-1})_{GG} \end{bmatrix},$$

where

$$\begin{aligned}
(\mathrm{D}^{-1})_{FF} &= \overline{\mathrm{S}}^{-1}, \\
(\mathrm{D}^{-1})_{FG} &= -\overline{\mathrm{S}}^{-1}\mathrm{D}_{FG}(\mathrm{D}_{GG})^{-1}, \\
(\mathrm{D}^{-1})_{GF} &= -(\mathrm{D}_{GG})^{-1}\mathrm{D}_{GF}\overline{\mathrm{S}}^{-1}, \textit{and} \\
(\mathrm{D}^{-1})_{GG} &= (\mathrm{D}_{GG})^{-1} + (\mathrm{D}_{GG})^{-1}\mathrm{D}_{GF}\overline{\mathrm{S}}^{-1}\mathrm{D}_{FG}(\mathrm{D}_{GG})^{-1}.
\end{aligned}$$

Finally, we present the following result known as the matrix inversion lemma or Sherman–Morrison–Woodbury formula.

Lemma 5 *If* D_{FF}, D_{GG}, *and* $\overline{\mathrm{S}}$ *are invertible, then*

$$(\mathrm{D}_{GG} - \mathrm{D}_{GF}(\mathrm{D}_{FF})^{-1}\mathrm{D}_{FG})^{-1} = (\mathrm{D}_{GG})^{-1} - (\mathrm{D}_{GG})^{-1}\mathrm{D}_{GF}\overline{\mathrm{S}}^{-1}\mathrm{D}_{FG}(\mathrm{D}_{GG})^{-1}.$$

References

1. Abe S (2007) Sparse least squares support vector training in the reduced empirical feature space. Pattern Analy Appl 10(3):203–214
2. Boser BE, Guyon IM, Vapnik VN (1992) A training algorithm for optimal margin classifiers. In: Proceedings of the 5th annual ACM workshop on computational learning theory. ACM Press, pp 144–152.
3. Bottou L, Lin CJ (2007) Support vector machine solvers. In: DD Léon Bottou Olivier Chapelle, Weston J (eds) Large-scale kernel machines, neural information processing, MIT Press, Cambridge, pp 1–28
4. Cawley GC, Talbot NLC (2004) Fast exact leave-one-out cross-validation of sparse least-squares support vector machines. Neural Netw 17(10):1467–1475
5. Cortes C, Vapnik V (1995) Support-vector networks. Mach Learn 20(3):273–297
6. Harmeling S, Ziehe A, Kawanabe M, Müller KR (2002) Kernel feature spaces and nonlinear blind source separation. In: Dietterich TG, Becker S, Ghahramani Z (eds) Advances in neural information processing systems 14, MIT Press, Cambridge, pp 761–768.
7. Horn R, Johnson CR (1985) Matrix analysis. Cambridge University Press, Cambridge
8. Joachims T (2006) Training linear SVMs in linear time. In: Eliassi-Rad T, Ungar LH, Craven M, Gunopulos D (eds) Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD 2006). ACM Press, New York, pp 217–226
9. Kumar S, Mohri M, Talwalkar A (2009) Sampling techniques for the Nyström method. In: van Dyk D, Welling M (eds) Proceedings of the twelfth international conference on artificial intelligence and statistics (AISTATS 2009). JMLR workshop and conference proceedings, vol 5, JMLR, pp 304–311
10. Lee YJ, Mangasarian OJ (2001) RSVM: reduced support vector machines. In: Proceedings of the first SIAM international conference on data mining, Chicago
11. Lin KM, Lin CJ (2003) A study on reduced support vector machines. IEEE Trans Neural Netw 14:1449–1459
12. Meyer CD (2000) Matrix analysis and applied linear algebra. Society for Industrial and Applied Mathematics, Philadelphia
13. Pahikkala T, Boberg J, Salakoski T (2006) Fast n-fold cross-validation for regularized least-squares. In: Honkela T, Raiko T, Kortela J, Valpola H (eds) Proceedings of the ninth Scandinavian conference on artificial intelligence (SCAI 2006), Otamedia Oy, Espoo, Finland, pp 83–90.
14. Pahikkala T, Suominen H, Boberg J, Salakoski T (2009) Efficient hold-out for subset of regressors. In: Kolehmainen M, Toivanen P, Beliczynski B (eds) Proceedings of the international conference on natural and adaptive computing algorithms (ICANNGA 2009). Lecture notes in computer science, vol 5495. Springer, pp 350–359
15. Pahikkala T, Tsivtsivadze E, Airola A, Boberg J, Järvinen J (2009) An efficient algorithm for learning to rank from preference graphs. Mach Learn 75(1):129–165
16. Poggio T, Girosi F (1990) Networks for approximation and learning. Proceedings of the IEEE 78(9)
17. Quiñonero-Candela J, Rasmussen CE (2005) A unifying view of sparse approximate gaussian process regression. J Mach Learn Res 6:1939–1959
18. Rahimi A, Recht B (2007) Random features for large-scale kernel machines. In: Platt JC, Koller D, Singer Y, Roweis ST, Platt JC, Koller D, Singer Y, Roweis ST (eds) Advances in neural information processing systems 20. MIT Press, Cambridge
19. Rifkin R, Yeo G, Poggio T (2003) Regularized least-squares classification. In: Suykens J, Horvath G, Basu S, Micchelli C, Vandewalle J (eds) Advances in learning theory: methods, model and applications, nato science series III: computer and system sciences, vol 190, chap. 7. IOS Press, Amsterdam, pp 131–154
20. Sætre R, Sagae K, Tsujii J (2008) Syntactic features for protein–protein interaction extraction. In: Baker CJ, Jian S (eds) Proceedings of the 2nd international symposium on languages in biology and medicine (LBM 2007), CEUR Workshop Proceedings, pp 6.1–6.14
21. Schölkopf B, Herbrich R, Smola AJ (2001) A generalized representer theorem. In: Helmbold D, Williamson R (eds) Proceedings of the 14th annual conference on computational learning theory and 5th European conference on computational learning theory (COLT 2001). Springer, Berlin, Germany, pp 416–426

22. Schölkopf B, Mika S, Burges C, Knirsch P, Müller KR, Rätsch G, Smola A (1999) Input space versus feature space in kernel-based methods. IEEE Trans Neural Netw 10(5):1000–1017
23. Schölkopf B, Smola AJ (2002) Learning with kernels. MIT Press, Cambridge
24. Shawe-Taylor J, Cristianini N (2004) Kernel methods for pattern analysis. Cambridge University Press, Cambridge
25. Shwartz SS, Singer Y, Srebro N (2007) Pegasos: primal estimated sub-gradient solver for SVM. In: Ghahramani Z (ed) Proceedings of the 24th international conference on Machine learning (ICML 2007). ACM international conference proceeding series, vol 227. New York, pp 807–814. doi:10.1145/1273496.1273598
26. Smola AJ, Schölkopf B (2000) Sparse greedy matrix approximation for machine learning. In: Langley P (ed) Proceedings of the seventeenth international conference on machine learning (ICML 2000). Morgan Kaufmann Publishers Inc., San Francisco, pp 911–918
27. Smola AJ, Vishwanathan SVN, Le Q (2007) Bundle methods for machine learning. In: McCallum A (ed) Advances in neural information processing systems 20. MIT Press, Cambridge
28. Suykens JAK, Gestel TV, Brabanter JD, Moor BD, Vandewalle J (2003) Least squares support vector machines. World Scientific Publishing Company
29. Tsivtsivadze E, Pahikkala T, Airola A, Boberg J, Salakoski T (2008) A sparse regularized least-squares preference learning algorithm. In: Holst A, Kreuger P, Funk P (eds) Proceedings of the Tenth Scandinavian Conference on Artificial Intelligence (SCAI 2008). Frontiers in artificial intelligence and applications, vol 173. IOS Press, pp 76–83
30. Tsuda K (1999) Support vector classifier with asymmetric kernel functions. In: European symposium on artificial neural networks (ESANN 1999), pp 183–188
31. Williams CKI, Seeger M (2001) Using the Nyström method to speed up kernel machines. In: Leen TK, Dietterich TG, Tresp V (eds) Advances in neural information processing systems 13. MIT Press, Cambridge, pp 682–688
32. Xiong H, Swamy M, Ahmad MO (2005) Optimizing the kernel in the empirical feature space. IEEE Trans Neural Netw 16(2):460–474
33. Zhang K, Tsang IW, Kwok JT (2008) Improved Nyström low-rank approximation and error analysis. In: McCallum A, Roweis S (eds) Proceedings of the 25th international conference on Machine learning (ICML 2008). ACM international conference proceeding series, vol 307. New York, pp 1232–1239

Paper IV

An experimental comparison of cross-validation techniques for estimating the area under the ROC curve

Airola, A., Pahikkala, T., Waegeman, W., De Baets, B., and Salakoski, T. (2011). *Computational Statistics & Data Analysis*, 55(4):1828–1844.

Paper V

All-paths graph kernel for protein-protein interaction extraction with evaluation of cross-corpus learning

Airola, A., Pyysalo, S., Björne, J., Pahikkala, T., Ginter, F., and Salakoski, T. (2008). *BMC Bioinformatics*, 9 Suppl 11

BMC Bioinformatics

BioMed Central

Research

All-paths graph kernel for protein-protein interaction extraction with evaluation of cross-corpus learning

Antti Airola*, Sampo Pyysalo, Jari Björne, Tapio Pahikkala, Filip Ginter and Tapio Salakoski

Address: Turku Centre for Computer Science (TUCS) and the Department of IT, University of Turku, Joukahaisenkatu 3-5, 20520 Turku, Finland

Email: Antti Airola* - antti.airola@utu.fi; Sampo Pyysalo - sampo.pyysalo@utu.fi; Jari Björne - jari.bjorne@utu.fi; Tapio Pahikkala - tapio.pahikkala@utu.fi; Filip Ginter - filip.ginter@utu.fi; Tapio Salakoski - tapio.salakoski@utu.fi

* Corresponding author

from Natural Language Processing in Biomedicine (BioNLP) ACL Workshop 2008
Columbus, OH, USA. 19 June 2008

Published: 19 November 2008

BMC Bioinformatics 2008, **9**(Suppl 11):S2 doi:10.1186/1471-2105-9-S11-S2

This article is available from: http://www.biomedcentral.com/1471-2105/9/S11/S2

Abstract

Background: Automated extraction of protein-protein interactions (PPI) is an important and widely studied task in biomedical text mining. We propose a graph kernel based approach for this task. In contrast to earlier approaches to PPI extraction, the introduced all-paths graph kernel has the capability to make use of full, general dependency graphs representing the sentence structure.

Results: We evaluate the proposed method on five publicly available PPI corpora, providing the most comprehensive evaluation done for a machine learning based PPI-extraction system. We additionally perform a detailed evaluation of the effects of training and testing on different resources, providing insight into the challenges involved in applying a system beyond the data it was trained on. Our method is shown to achieve state-of-the-art performance with respect to comparable evaluations, with 56.4 F-score and 84.8 AUC on the Almed corpus.

Conclusion: We show that the graph kernel approach performs on state-of-the-art level in PPI extraction, and note the possible extension to the task of extracting complex interactions. Cross-corpus results provide further insight into how the learning generalizes beyond individual corpora. Further, we identify several pitfalls that can make evaluations of PPI-extraction systems incomparable, or even invalid. These include incorrect cross-validation strategies and problems related to comparing F-score results achieved on different evaluation resources. Recommendations for avoiding these pitfalls are provided.

Background

Information extraction from biomedical research publications has been a topic of intense research during recent years [1-3]. Literature databases such as PubMed offer access through online interfaces to records of millions of research articles from the biomedical domain, with abstracts made available for many, and full texts for some of the papers. Potentially, this offers a researcher direct

access to vast amounts of research knowledge. However, locating the useful information can be challenging, a simple keyword search may still return many more articles than a human being can process. This motivates the development of tools for automating the extraction of information from biomedical text.

A task of significant interest in biomedical natural language processing is the automated protein-protein interaction (PPI) extraction. The most commonly addressed problem has been the extraction of binary interactions, where the system identifies which protein pairs in a sentence have a biologically relevant relationship between them. Proposed solutions include both hand-crafted rule-based systems and machine learning approaches (see e.g. [4]). A wide range of results have been reported for the systems, but as we will show, differences in evaluation resources, metrics and strategies make direct comparison of the numbers presented problematic. Further, the results gained from the BioCreative II evaluation, where the best performing system achieved a 29% F-score [5], suggest that the problem of extracting binary protein-protein interactions is far from solved.

The public availability of large annotated PPI-corpora such as AImed [4], BioInfer [6] and GENIA [7], provides an opportunity for building PPI extraction systems automatically using machine learning. A major challenge is how to supply the learner with the contextual and syntactic information needed to distinguish between interactions and non-interactions. To address the ambiguity and variability of the natural language expressions used to state PPI, several recent studies have focused on the development, adaptation and application of NLP tools for the biomedical domain. Many high-quality domain-specific tools are now freely available, including full parsers such as that introduced by Lease and Charniak [8]. Additionally, a number of conversions from phrase structure parses to dependency structures that make the relationships between words more directly accessible have been introduced. These include conversions into representations such as the Stanford dependency scheme [9] that are explicitly designed for information extraction purposes. However, specialized feature representations and kernels are required to make learning from such structures possible.

Approaches such as subsequence kernels [10], tree kernels [11] and shortest path kernels [12] have been proposed and successfully used for relation extraction. However, these methods lack the expressive power to consider representations derived from general, possibly cyclic, dependency graph structures, such as those generated by the Stanford tools. The subsequence kernel approach does not consider parses at all, and the shortest path approach is limited to representing only a single path in the full dependency graph, which excludes relevant words even in many simple cases (Figure 1). Tree kernels can represent more complex structures, but are still restricted to tree representations.

Lately, in the framework of kernel-based machine learning methods there has been an increased interest in designing kernel functions for graph data. Building on the work of Gärtner et al. [13], graph representations tailored for the task of dependency parse ranking were proposed by Pahikkala et al. [14]. Though the proposed representations are not directly applicable to the task of PPI extraction, they offer insight in how to learn from dependency graphs. We develop a graph kernel approach for PPI extraction based on these ideas.

We next define a graph representation suitable for describing potential interactions and introduce a kernel which makes efficient learning from a general, unrestricted graph representation possible. Then we provide a short description of the sparse regularized least squares (sparse RLS) kernel-based machine learning method we use for PPI-extraction.

Further, we rigorously assess our method on five publicly available PPI corpora. In addition to purely intrinsic evaluation using cross-validation on single corpora, we pro-

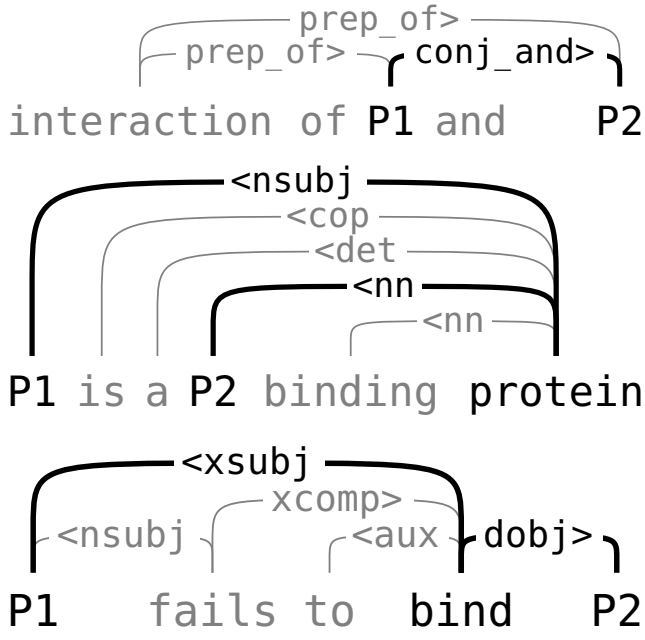

Figure 1
Shortest path example. Stanford dependency parses ("collapsed") representation where the shortest path, shown in bold, excludes important words.

vide a broad cross-corpus evaluation to test how well an extraction system trained on a given corpus will generalize to the other corpora. We thus provide, to our knowledge, the most comprehensive evaluation done with a machine learning approach to PPI-extraction. Finally, we discuss the effects that different evaluation strategies, choice of corpus and applied metrics have on measured performance, and provide conclusions.

Methods

We next present our graph representation, formalize the notion of graph kernels, and present our learning method of choice, the sparse RLS.

Graph encoding of sentence structure

As in most recent work on machine learning for PPI extraction, we cast the task as learning a decision function that determines for each unordered candidate pair of protein names occurring together in a sentence whether the two proteins interact. In the following, we first define the graph representation used to represent an interaction candidate pair. We then proceed to derive the kernel used to measure the similarities of these graphs.

We assume that the input of our learning method is a dependency parse of a sentence where a pair of protein names is marked as the candidate interaction for which an extraction decision must be made. Based on this, we form a weighted, directed graph that consists of two unconnected subgraphs. One represents the dependency structure of the sentence, and the other the linear order of the words (see Figure 2).

The first subgraph is built from the dependency analysis. One vertex and an associated set of labels is created in the graph for each token and for each dependency. The vertices that represent tokens have as labels the text and part-of-speech (POS) of the token. To ensure generalization of the learned extraction model, the labels of vertices that correspond to protein names are replaced with PROT1, PROT2 or PROT, where PROT1 and PROT2 are the pair of interest. The vertices that represent dependencies are labeled with the type of the dependency. The edges in the subgraph are defined so that each dependency vertex is connected by an incoming edge from the vertex representing its governor token, and by an outgoing edge to the vertex representing its dependent token. The graph thus represents the entire sentence structure.

It is widely acknowledged that the words between the candidate entities or connecting them in a syntactic representation are particularly likely to carry information regarding their relationship; Bunescu and Mooney [12] formalize this intuition for dependency graphs as the *shortest path hypothesis*. We apply this insight in two ways in the graph representation: the labels of the nodes on the shortest undirected paths connecting PROT1 and PROT2 are differentiated from the labels outside the paths using a special tag. Further, the edges are assigned weights; after limited preliminary experiments, we chose a simple weighting scheme where all edges on the shortest paths receive a weight of 0.9 and other edges receive a weight of 0.3. The representation thus allows us to emphasize the shortest path without completely disregarding potentially relevant words outside the path.

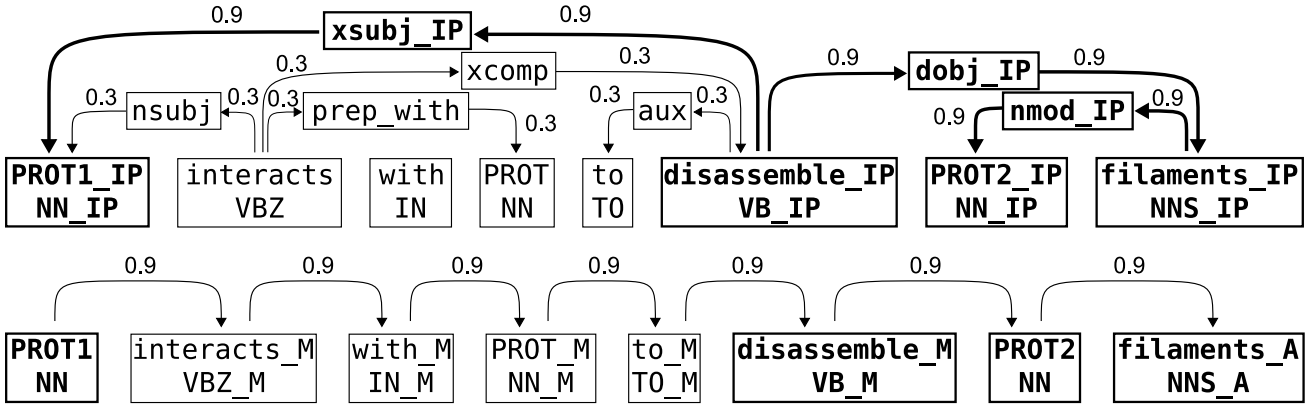

Figure 2

Graph representation. Graph representation generated from an example sentence. The candidate interaction pair is marked as PROT1 and PROT2, the third protein is marked as PROT. The shortest path between the proteins is shown in bold. In the dependency based subgraph all nodes in a shortest path are specialized using a post-tag (IP). In the linear order subgraph possible tags are (B)efore, (M)iddle, and (A)fter. For the other two candidate pairs in the sentence, graphs with the same structure but different weights and labels would be generated.

The second subgraph is built from the linear structure of the sentence. For each token, a second vertex is created and the labels for the vertices are derived from the texts, POS-tags and named entity tagging as above. The labels of each word are specialized to denote whether the word appears before, in-between, or after the protein pair of interest. Each word node is connected by an edge to its succeeding word, as determined by sentence order of the words. Each edge is given the weight 0.9.

The all-paths graph kernel

We next formalize the graph representation and present the all-paths graph kernel. This kernel can be considered as a practical instantiation of the theoretical graph kernel framework introduced by Gärtner et al. [13]. Let V be the set of vertices in the graph and $\mathcal{L}$ be the set of possible labels vertices can have. We represent the graph with an adjacency matrix $A \in \mathbb{R}^{|V| \times |V|}$, whose rows and columns are indexed by the vertices, and $[A]_{i,j}$ contains the weight of the edge connecting $v_i \in V$ and $v_j \in V$ if such an edge exists, and zero otherwise. Further, we represent the labels as a label allocation matrix $L \in \mathbb{R}^{|\mathcal{L}| \times |V|}$ so that $L_{i,j} = 1$ if the j-th vertex has the i-th label and $L_{i,j} = 0$ otherwise. Because only a very small fraction of all the possible labels are ever assigned to any single node, this matrix is extremely sparse.

It is well known that when an adjacency matrix is multiplied with itself, each element $[A^2]_{i,j}$ contains the summed weight of paths from vertex v_i to vertex v_j through one intervening vertex, that is, paths of length two. Similarly, for any length n, the summed weights from v_i to v_j can be determined by calculating $[A^n]_{i,j}$. Since we are interested not only in paths of one specific length, it is natural to combine the effect of paths of different lengths by summing the powers of the adjacency matrices. We calculate the infinite sum of the weights of all possible paths connecting the vertices using the Neumann Series, defined as

$$(I - A)^{-1} = I + A + A^2 + \ldots = \sum_{k=0}^{\infty} A^k$$

if $|A| < 1$ where $|A|$ is the spectral radius of A [15]. From this sum we can form a new adjacency matrix

$$W = (I - A)^{-1} - I.$$

The final adjacency matrix contains the summed weights of all possible paths connecting the vertices. The identity matrix is subtracted to remove the paths of length zero, which would correspond to self-loops. Next, we present the graph kernel that utilizes the graph representation defined previously. We define an instance G representing a candidate interaction as $G = LWL^{\mathrm{T}}$, where L and W are the label allocation matrix and the final adjacency matrix corresponding to the graph representation of the candidate interaction.

Following Gärtner et al. [13] the graph kernel is defined as

$$k(G', G'') = \sum_{i=1}^{|\mathcal{L}|} \sum_{j=1}^{|\mathcal{L}|} G'_{i,j} G''_{i,j},$$

where G' and G'' are two instances formed as defined previously. The features can be thought as combinations of labels from connected pairs of vertices, with a value that represents the strength of their connection. In practical implementations, the full G matrices, which consist mostly of zeroes, are never explicitly formed. Rather, only the non-zero elements are stored in memory and used when calculating the kernels.

Scalable learning with Sparse RLS

RLS, also known as the least squares support vector machine, is a state-of-the-art kernel-based machine learning method which has been shown to have comparable performance to standard support vector machines [16,17]. We choose the sparse version of the algorithm, also known as subset of regressors, as it allows us to scale up the method to very large training set sizes. Sparse RLS also has the property that it is possible to perform cross-validation and regularization parameter selection so that their time complexities are negligible compared to the training complexity. These efficient methods are analogous to the ones proposed by Pahikkala et al. [18] for the basic RLS regression.

We now briefly present the basic sparse RLS algorithm. Let m denote the training set size and $M = \{1, \ldots, m\}$ an index set in which the indices refer to the examples in the training set. Instead of allowing functions that can be expressed as a linear combination over the whole training set, as in the case of basic RLS regression, we only allow functions of the following restricted type:

$$f(\cdot) = \sum_{i \in B} a_i k(\cdot, x_i), \qquad (1)$$

where k is the kernel function, x_i are training data points, $a_i \in \mathbb{R}$ are weights, and the set indexing the basis vectors $B \subset M$ is selected in advance. The coefficients a_i that determine (1) are obtained by minimizing

$$\sum_{j=1}^{m} (y_j - \sum_{i \in B} a_i k(x_j, x_i))^2 + \lambda \sum_{j,i \in B} a_j a_i k(x_j, x_i), \qquad (2)$$

where the first term is the squared loss function, the second term is the regularizer, and $\lambda \in \mathbb{R}_+$ is a regularization parameter. All the training instances are used for determining the coefficient vector, but only a subset of them to represent the learned hypothesis. The minimizer of (2) is obtained by solving the corresponding system of linear equations, which can be performed in $O(m|B|^2)$ time.

We set the maximum number of basis vectors to 4000 in all experiments in this study. The subset is selected randomly when the training set size exceeds this number. Other methods for the selection of the basis vectors were considered by Rifkin et al. [17], who however reported that the random selection worked as well as the more sophisticated approaches.

Results and discussion

We next describe the evaluation resources and metrics used, provide a comprehensive evaluation of our method across five PPI corpora, and compare our results to earlier work. Further, we discuss the challenges inherent in providing a valid method evaluation and propose solutions.

Corpora and evaluation criteria

We evaluate our method using five publicly available corpora that contain PPI interaction annotation: AImed [4], BioInfer [6], HPRD50 [19], IEPA [20] and LLL [21]. All the corpora were processed to a common format using transformations [22] that we have introduced earlier [23]. We note that the version of the BioInfer used in this study differs from the one we considered in [23] and in [24]. This is due to the fact that these studies used an early version of the binarization rules [25] that transform the complex relations of BioInfer to binary ones.

We parse these corpora with the Charniak-Lease parser [8], which has been found to perform best among a number of parsers tested in recent domain evaluations [26,27]. The Charniak-Lease phrase structure parses are transformed into the collapsed Stanford dependency scheme using the Stanford tools [9]. We cast the PPI extraction task as binary classification, where protein pairs that are stated to interact are positive examples and other co-occurring pairs negative. Thus, from each sentence, $\binom{n}{2}$ examples are generated, where n is the number of occurrences of protein names in the sentence. Finally, we form the graph representation described earlier for each candidate interaction.

In the single corpus tests we evaluate the method with 10-fold document-level cross-validation on all of the corpora. This guarantees the maximal use of the available data, and also allows comparison to relevant earlier work. In particular, on the AImed corpus we apply the exact same 10-fold split that was used by Bunescu et al. [10], Giuliano et al. [28], Van Landeghem et al. [29], and possibly some of the other studies which do not explicitly state which split was used. In cross-corpus tests we use each of the corpora in turn to train an extraction system, and test the system on the four remaining corpora.

Performance is measured according to the following criteria: interactions are considered untyped, undirected pairwise relations between specific protein mentions, that is, if the same protein name occurs multiple times in a sentence, the correct interactions must be extracted for each occurrence. Further, we do not consider self-interactions as candidates and remove them from the corpora prior to evaluation. The majority of PPI extraction system evaluations use the balanced F-score measure for quantifying the performance of the systems. This metric is defined as $F = \frac{2pr}{p+r}$, where p is precision and r recall. Likewise, we provide F-score, precision, and recall values in our evaluation. It should be noted that F-score is very sensitive to the underlying positive/negative pair distribution of the corpus – a property whose impact on evaluation is discussed in detail below. As an alternative to F-score, we also evaluate the performance of our system using the *area under the receiver operating characteristics curve* (AUC) measure [30]. AUC has the important property that it is invariant to the class distribution of the used dataset. Due to this and other beneficial properties for comparative evaluation, the usage of AUC for performance evaluation has been recently advocated in the machine learning community (see e.g. [31]). Formally, AUC can be defined as

$$AUC = \frac{\sum_{i=1}^{m_+} \sum_{j=1}^{m_-} H(x_i - y_j)}{m_+ m_-},$$

where m_+ and m_- are the numbers of positive and negative examples, respectively, and $x_1, \ldots, x_{m_+}$ are outputs of the system for the positive, and $y_1, \ldots, y_{m_-}$ for the negative examples, and

$$H(r) = \begin{cases} 1, & \text{if } r > 0 \\ 0.5, & \text{if } r = 0 \\ 0, & \text{otherwise.} \end{cases}$$

The outputs are real valued and can be thought of as inducing a ranking, where the examples considered to be most likely to belong to the positive class should receive

the highest output values. The measure corresponds to the probability that given a randomly chosen positive and negative example, the system will be able to correctly distinguish which one is which.

Performance on the individual corpora

The performance of our method on the five corpora for the various metrics is presented in Table 1. For reference, we show also the performance of the co-occurrence (or *all-true*) baseline, which simply assigns each candidate into the interaction class. The recall of the co-occurrence method is trivially 100%, and in terms of AUC it has a performance of 50%, the random baseline. All the numbers in Table 1, including the co-occurrence results, are averages taken over the ten folds. One should note that because of the non-linearity of the F-score measure, the average precision and recall will not produce exactly the average F. Further, calculating the co-occurrence numbers as averages over the folds leads to results that differ slightly compared to the approach where the co-occurrence statistic is calculated over all the data pooled together.

The results hold several interesting findings. First, we briefly observe that on the AImed corpus, which has recently been applied in numerous evaluations [32] and can be seen as an emerging *de facto* standard for PPI extraction method evaluation, the method achieves an F-score performance of 56.4%. As we argue in more detail below, this level of performance is comparable to the state-of-the-art in machine learning based PPI extraction. For the other large corpus, BioInfer, F-score performance is somewhat higher, at 61%. Second, we observe that the F-score performance of the method varies strikingly between the different corpora, with results on IEPA and LLL approximately 20 percentage units higher than on AImed and 15 percentage units higher than on BioInfer, despite the larger size of the latter two. In our previous work we have observed similar results with a rule-based extraction method [23]. As a broad multiple corpus evaluation using a state-of-the-art machine learning method for PPI extraction, our results support and extend the key finding that F-score performance results measured on different corpora cannot, in general, be meaningfully compared.

The co-occurrence baseline numbers indicate one reason for the high F-score variance between the corpora. The F-score metric is not invariant to the distribution of positive and negative examples: for example, halving the number of negative test examples is expected to approximately halve the number of false positives at a given recall point. Thus, the greater the fraction of true interactions in a corpus is, the easier it is to reach high performance in terms of F-score. This is reflected in co-occurrence results, which range from 30% to 70% depending on the class distribution of the corpus.

This is a critical weakness of the F-score metric in comparisons involving different corpora as, for example, the fraction of true interactions out of all candidates is 50% in the LLL corpus but only 17% in AImed. By contrast to the large differences in performance measured using F-score, we find that for the distribution-invariant AUC measure the performance for all of the corpora falls in the range of 80–85%. The results provide an argument in favor of applying the AUC metric instead of, or in addition to, F-score. AUC is also more stable in terms of variance.

The similar performance in terms of AUC for corpora with as widely differing sizes as LLL and BioInfer allows for two alternative interpretations. First, it might be that past a relatively modest number of examples, increasing corpus size has little effect on the performance of the method. Alternatively, it might be the case that the larger corpora, while having more training data available, are also more difficult to learn than the smaller corpora. We explore the issue further by calculating learning curves on the corpora, using AUC as the performance measure (see Figure 3). For each corpus five folds are set aside as the test set, and the rest of the data is incrementally added to the training set to test how increase in training data affects the performance.

The learning curves support the latter interpretation. If the datasets all represented equally difficult tasks with respect to distinguishing randomly drawn positive instances from negatives, we would expect the curves to roughly overlap. The fact that they are to a large extent separate indicates that there are large differences in the difficulty of the

Table 1: Evaluation results. Counts of positive and negative examples in the corpora and (P)recision, (R)ecall, (F)-score and AUC for the graph kernel, with standard deviations provided for F and AUC.

	Statistics		Graph Kernel						co-occ	
Corpus	#POS.	#NEG.	P	R	F	σ_F	AUC	σ_{AUC}	P	F
AIMed	1000	4834	52.9%	61.8%	56.4%	5.0%	84.8%	2.3%	17.8%	30.1%
BioInfer	2534	7132	56.7%	67.2%	61.3%	5.2%	81.9%	4.9%	26.6%	41.7%
HPRD50	163	270	64.3%	65.8%	63.4%	11.4%	79.7%	6.3%	38.9%	55.4%
IEPA	335	482	69.6%	82.7%	75.1%	7.0%	85.1%	5.1%	40.8%	57.6%
LLL	164	166	72.5%	87.2%	76.8%	17.8%	83.4%	12.2%	55.9%	70.3%

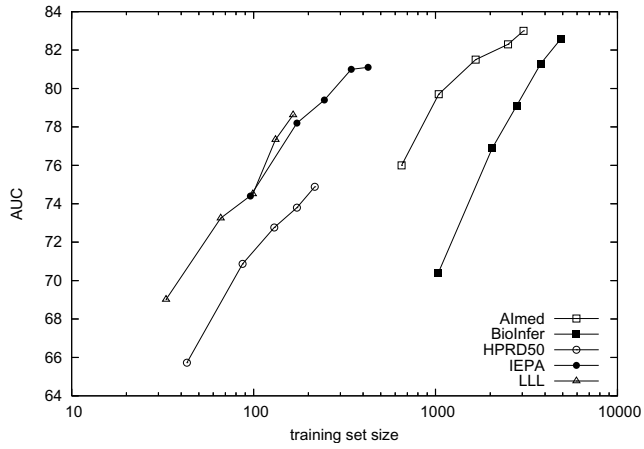

Figure 3
Learning curves. Learning curves for the five corpora. The scale is logarithmic with respect to the amount of training data.

learning tasks represented by the different corpora. On AImed and BioInfer it takes significantly more data to reach the same performance than on the three smaller corpora HPRD50, LLL and IEPA: For example, performance on the latter three with 100 training examples exceeds the performance on BioInfer with ten times as much training data.

Cross-corpus performance

The cross-corpus evaluation aims to shed light on a question of fundamental importance in training machine learning based PPI-extraction systems: Will the learned models generalize beyond the specific characteristics of the data they were trained on? The types of named entities annotated, the definition of what exactly constitutes an interaction and the relative positive/negative distributions of pairs can vary significantly over different corpora. Thus it is not obvious that a system trained on a given corpus will perform well on data which is not from the same corpus. As discussed in [33], applying text mining tools beyond the development data can lead to disappointing results.

We explore this issue through a cross-corpus evaluation of our method. Five extraction systems are trained, one on each corpus, and they are each tested on the four remaining corpora. Leave-one-document-out cross-validation on the training corpus is used for parameter value selection. Our evaluation extends the recent results of Van Landeghem et al. [29], who conducted cross-corpus experiments on four of the corpora considered in this study. Their finding was that models trained on a combination of three of the corpora often did not perform well on terms of F-score, when tested on the remaining corpus.

We start by considering the AUC results of the cross-corpus evaluation (see Table 2), as the metric normalizes away much of the differences resulting from differing positive/negative distributions and threshold selection strategies, thus providing a more stable view of performance. We notice that the performance varies significantly depending on the training and test corpus. Unlike in the single corpus evaluations the results are scattered, ranging from 61% to 83% AUC. On the large corpora the trained extraction systems in all cases perform clearly worse than the cross-validation performance. However, on the two smallest corpora this is not so. On HPRD50 systems trained on AImed and IEPA actually give better performance than the results from cross-validating on the corpus. On LLL the models trained on BioInfer and IEPA do almost as well as the cross-validation results on the corpus. These results suggest that a larger amount of training data can compensate for the differences in corpus annotation strategies to a large extent. Random chance may also be a factor here, as observed previously in the large variances in cross-validation results on the smallest corpora.

One relevant question that can be answered from the cross-corpus experiments is which of the corpora provides the best resource for training from a generalization perspective. However, it is not entirely straightforward to meaningfully summarize these results: simple averages over results on the very different resources carry little meaning. Instead, we provide a simple, rough indicator of generalization potential by ranking the corpora separately according to the results on each of the other corpora. The rankings are presented in Table 2. Though the rankings do differ over different test corpora, overall they roughly fol-

Table 2: Cross-corpus results measured with AUC. AUC results for cross-corpus testing. Rows correspond to training corpora and columns to test corpora.

	AImed	rank	BioInfer	rank	HPRD50	rank	IEPA	rank	LLL	rank	avg. rank
AImed	-	-	67.7%	2	82.4%	1	76.1%	2	77.8%	3	2
BioInfer	77.8%	1	-	-	75.2%	3	79.3%	1	83.3%	1	1.5
HPRD50	72.5%	2	61.8%	3	-	-	74.9%	3	64.0%	4	3
IEPA	70.2%	3	72.2%	1	80.0%	2	-	-	82.5%	2	2
LLL	61.8%	4	61.0%	4	69.4%	4	74.8%	4	-	-	4

low the size of the corpora. On average models trained on the largest corpus, BioInfer, perform best. Next in this ranking the second and third largest corpora, AImed and IEPA share a rank. The second worst performing models are trained on the second smallest corpus HPRD50, and the lowest performing ones on the smallest dataset, LLL. Unsurprisingly, the more training data available the better the performance is. A surprising result is the high performance of systems trained on IEPA, the corpus being an order of magnitude smaller than AImed or BioInfer.

Next, we consider the results using the F-score measure. In Table 3 results for which the threshold separarating positive and negative classes has been selected on the training corpus are shown. In some cases the results are on a similar level to those gained in the single corpus cross-validation experiments. This holds true for example with models trained on AImed or IEPA, and tested on the HPRD50 corpus. However, there are several cases where the performance is disastrously low. Most strikingly, three out of four results gained when using AImed for training fall below the results one would achieve with the naive co-occurrence baseline. We observe that even in these cases the AUC results are still competitive. This gives rise to the assumption that the problem is in the threshold selection. The learned models do have the property that they tend to assign higher values for the positive than for the negative examples, but the approach of selecting the suitable threshold on training data for separating the two classes fails utterly in some cases. We further observed that avoiding the task of threshold selection altogether by setting it simply to zero yielded no better results.

In Table 4 we provide the optimal F-score results, choosing the positions from the precision/recall curves that would lead to highest F-scores. Many of the results are now greatly increased, with no result falling below the naive co-occurrence baseline. Further, the relative ranking order of the results is the same as that induced by the AUC scores. It is now clear that one can not necessarily rely on the approach of choosing the threshold according to what works on the training set when doing cross-corpus learning. This is perhaps due to the large differences in the underlying positive/negative distributions of the corpora. The differences mean breaking the basic assumption made by the majority of machine learning methods, that the training and test examples are identically distributed. As can be seen from the statistics presented in Table 1, the examples are clearly not identically distributed over the corpora, at least with respect to outputs.

One approach for selecting which examples to assign to positive and which to negative classes could be selecting the threshold according to the the relative positive/negative distribution of the test set. To estimate this in a practical setting, one may have to sample and manually check examples from the test set. In Table 5 are presented the F-score results gained when assigning to positive class such a fraction of the test examples that corresponds to the relative frequency of positive examples in the test corpus. In all the cases the results are within a few percentage units of the optimal values, indicating that this simple heuristic allows the worst disasters observed in the cross-corpus tests to be avoided. However, there are several cases where the result achieved with this approach is lower than when choosing the threshold on the training data.

To conclude, the cross-corpus learning results support the assumption that the learned models generalize beyond the corpora they were trained on. Still, results are generally lower when testing a method against a corpus different from that on which it was trained. We observe that the systems trained on larger corpora tend to perform better than the ones trained on smaller ones, as is to be expected. The results achieved with the IEPA as a training corpus are surprisingly competitive, considering how much smaller it is than the two larger corpora. Choosing a threshold for separating the positive and negative classes proves to be a challenging issue, as a threshold chosen on the training corpus may not work at all on another.

Performance compared to other methods

We next discuss the performance of our method compared to other methods introduced in the literature and the challenges of meaningful comparison, where we identify three major issues.

Table 3: Cross-corpus results measured with F-score and threshold chosen on training set. F-score results for cross-corpus testing with the thresholds chosen on the training set. Rows correspond to training corpora and columns to test corpora. Δ denote the difference between the F-score result and the result achieved with the optimal threshold.

	AImed	Δ	BioInfer	Δ	HPRD50	Δ	IEPA	Δ	LLL	Δ
AImed	-	-	24.9%	22.2%	64.6%	4.4%	22.9%	44.5%	17.7%	56.8%
BioInfer	44.2%	3.0%	-	-	63.6%	0.3%	64.5%	3.5%	76.4%	1.6%
HPRD50	40.9%	1.3%	27.2%	15.3%	-	-	56.3%	8.8%	45.5%	22.4%
IEPA	38.4%	0.7%	47.0%	4.7%	65.6%	1.9%	-	-	77.0%	0.6%
LLL	32.6%	0.7%	42.2%	0.3%	58.3%	1.5%	63.9%	1.0%	-	-

Table 4: Cross-corpus results measures with F-score and optimal thresholds. F-score results for cross-corpus testing with the optimal thresholds. Rows correspond to training corpora and columns to test corpora.

	AImed	BioInfer	HPRD50	IEPA	LLL
AImed	-	47.1%	69.0%	67.4%	74.5%
BioInfer	47.2%	-	63.9%	68.0%	78.0%
HPRD50	42.2%	42.5%	-	65.1%	67.9%
IEPA	39.1%	51.7%	67.5%	-	77.6%
LLL	33.3%	42.5%	59.8%	64.9%	-

First, as indicated by the results above, differences in the makeup of different corpora render cross-corpus comparisons in terms of F-score essentially meaningless. As F-score is typically the only metric for which results are reported in the PPI extraction literature, we are limited to comparing against results on single corpora. We consider the AImed and BioInfer evaluations to be the most relevant ones, as these corpora are sufficiently large for training and reliably testing machine learning methods. As the present study is, to the best of our knowledge, the first to report machine learning method performance on BioInfer, we will focus on AImed in the following comparison.

Second, the cross-validation strategy used in evaluation has a large impact on measured performance. The pair-based examples can break the assumption of the training and test sets being independent of each other, as pairs generated from the same sentence, and to a lesser extent from the same document, are clearly not independent. This must be taken into account when designing the experimental setup (see e.g. [18] for further discussion). In earlier system evaluations, two major strategies for defining the splits used in cross-validation can be observed. The approach used by Bunescu and Mooney [10], which we consider the correct one, is to split the data into folds on the level of documents. This guarantees that all pairs generated from the same document are always either in the training set or in the test set. Another approach is to pool all the generated pairs together, and then randomly split them to folds. To illustrate the significance of this choice, consider two interaction candidates extracted from the same sentence, e.g. from a statement of the form "P_1 and P_2 [...] P_3", where "[...]" is any statement of interaction or non-interaction. Due to the near-identity of contexts, a machine learning method will easily learn to predict that the label of the pair (P_1, P_3) should match that of (P_2, P_3). However, such "learning" will clearly not generalize. This approach must thus be considered invalid, because allowing pairs generated from the same sentences to appear in different folds leads to an information leak between the training and test sets. Sætre et al. [32] observed that adopting the latter cross-validation strategy on AImed could lead *up to 18 F-score percentage unit overestimation of performance*. For this reason, we will not consider results listed in the "False 10-fold cross-validation" table (2b) of Sætre et al. [32].

With these restrictions in place, we now turn to comparison with relevant results reported in related research, summarized in Table 6. Among the work left out of the comparison we note the results of Bunescu and Mooney [10], who reported a performance of 54.2% F on AImed. Though they used the same cross-validation strategy as the one used in our experiments, their results are not comparable to the ones included in the Table 6. They applied evaluation criteria where it is enough to extract only one occurrence of each mention of an interaction from each abstract, while the results shown were evaluated using the same criteria as applied here. The former approach can produce higher performance: the evaluation of Giuliano et al. [28] includes both alternatives, and their method achieves an F-score of 63.9% under the former criterion, which they term One Answer per Relation in a given Document (OARD).

The best performing system, that of Miwa et al. [34], combines the all-paths graph kernel, implemented based on the description we provided in [24], together with other kernels. Their results can be considered as a further validation about the suitability of the graph kernel for PPI-extraction. Our implementation of the all-paths method outperforms most of the other studies using similar eval-

Table 5: Cross-corpus results measured with F-score and thresholds based on the distribution of test set. F-score results for cross-corpus testing with the thresholds chosen according to the positive/negative distribution of the test set. Rows correspond to training corpora and columns to test corpora. Δ denote the difference between the F-score result and the result achieved with the optimal threshold.

	AImed	Δ	BioInfer	Δ	HPRD50	Δ	IEPA	Δ	LLL	Δ
AImed	-	-	44.7%	2.4%	65.6%	3.4%	63.9%	3.5%	70.1%	4.4%
BioInfer	42.6%	4.6%	-	-	62.0%	1.9%	66.9%	1.1%	75.6%	2.4%
HPRD50	39.1%	3.1%	40.0%	2.5%	-	-	63.3%	1.8%	58.5%	9.4%
IEPA	33.5%	5.6%	48.4%	3.3%	66.3%	1.2%	-	-	77.4%	0.2%
LLL	26.5%	6.8%	38.7%	3.8%	54.0%	5.8%	63.0%	1.9%	-	-

Table 6: Comparison on AImed. (P)recision, (R)ecall, (F)-score and AUC results for methods evaluated on AImed with the correct cross-validation methodology. Note that the best performing method, introduced by Miwa et al. [34], also utilizes the all-paths graph kernel.

	P	R	F	AUC
Miwa et al. [34]	-	-	63.5%	87.9%
Miyao et al. [35]	54.9%	65.5%	59.5%	-
Giuliano et al. [28]	60.9%	57.2%	59.0%	-
All-paths graph kernel	52.9%	61.8%	56.4%	84.8%
Sætre et al. [32]	64.3%	44.1%	52.0%	-
Mitsumori et al. [39]	54.2%	42.6%	47.7%	-
Van Landeghem et al. [29]	49%	44%	46%	-
Yakushiji et al. [40]	33.7%	33.1%	33.4%	-

uation methodology, with the exceptions being the approaches Miyao et al. [35] and Giuliano et al. [28].

Miyao reports choosing in the experiments always the optimal point from the precision/recall curve, an approach we observe would raise our results around the same level. The results of Giuliano et al. are somewhat surprising, as their method does not apply any form of parsing but relies instead only on the sequential order of the words. This brings us to our third point regarding comparability of methods. As pointed out by Sætre et al. [32], the AImed corpus allows remarkably different "interpretations" regarding the number of interacting and non-interacting pairs. For example, where we have identified 1000 interacting and 4834 non-interacting protein pairs in AImed, in the data used by Giuliano there are eight more interacting and 200 fewer non-interacting pairs. The corpus can also be preprocessed in a number of ways. In particular we noticed that whereas protein names are always blinded in our data, in the data used by Giuliano protein names are sometimes partly left visible. As Giuliano has generously made his method implementation available [36], we were able to test the performance of his system on the data we used in our experiments. This resulted in an F-score of 52.4%.

Finally, there remains an issue of parameter selection. For sparse RLS the values of the regularization parameter λ and the decision threshold separating the positive and negative classes must be chosen, which can be problematic when no separate data for choosing them is available. Choosing from several parameter values the ones that give best results in testing, or picking the best point from a precision/recall curve when evaluating in terms of F-score, will lead to an over-optimistic evaluation of performance. This issue has often not been addressed in earlier evaluations that do cross-validation on a whole corpus. We choose the parameters by doing further leave-one-document-out cross-validation within each round of 10-fold-cross-validation, on the nine folds that constitute the training set.

As a conclusion, we observe the results achieved with the all-paths graph kernel to be state-of-the-art level. However, differences in evaluation strategies and the large variance exhibited in the results make it impossible to state which of the systems considered can be expected in general to perform best. We encourage future PPI-system evaluations to report AUC and F-score results over multiple corpora, following clearly defined evaluation strategies, to bring further clarity to this issue. For further discussion on resolving the challenges of comparing biomedical relation extraction results we refer to [37].

Conclusion

In this paper we have proposed a graph kernel approach to extracting protein-protein interactions, which captures the information in unrestricted dependency graphs to a format that kernel based learning algorithms can process. The method combines syntactic analysis with a representation of the linear order of the sentence, and considers all possible paths connecting any two vertices in the resulting graph. We demonstrate state-of-the-art performance for the approach. All software developed in the course of this study is made publicly available at [22].

A cross-corpus evaluation is performed to test whether an extraction system will work beyond the corpus it was trained on. We observe this to be the case, though results are generally worse than when training and testing on data from the same corpus. Having a larger amount of data available leads to better performance. Extraction systems trained on the largest corpora work on the smallest ones in some cases as well as systems trained on data directly from the smaller corpora themselves.

We identify a number of issues which make results achieved with different evaluation strategies and resources incomparable, or even incorrect. In our experimental design we consider the problems related to differences across corpora, the effects different cross-validation strategies have, and how parameter selection can be done. Our recommendation is to provide evaluations over different corpora, to use document-level cross-validation and to always select parameters on the training set.

We draw attention to the behavior of the F-score metric over corpora with differing pair distributions. The higher the relative frequency of interacting pairs is, the higher the performance can be expected to be. This is noticed both for the graph kernel method and for the naive co-occurrence baseline. Indeed, the strategy of just stating that all pairs interact leads to as high a result as 70% F-score on one of the corpora. We consider AUC as an alternative

measure that does not exhibit such behavior, as it is invariant to the distribution of pairs. The AUC metric is much more stable across all the corpora, and never gives better results than random for approaches such as the naive co-occurrence.

Though we only consider binary interactions in this work, the graph representations have the property that they could be used to represent more complex structures than pairs. The availability of corpora that annotate complex interactions, such as the full BioInfer and GENIA, makes training a PPI extraction system for extracting complex interactions an important avenue of future research (see [38] for further discussion). However, how to avoid the combinatorial explosion following from considering triplets, quartets etc. remains an open question. Also, the performance of the current approaches may need to be yet improved before extending them to recognize complex interactions.

Competing interests

The authors declare that they have no competing interests.

Authors' contributions

AA designed the graph kernel, implemented it with the help of JB, and had the main responsibility for experiments. AA, FG, JB and SP explored suitable features and their representations. TP provided the sparse RLS algorithms and advice on kernel design. AA was the main author of the manuscript with contributions from all other authors, all of whom read and approved the final version.

Acknowledgements

We would like to thank Razvan Bunescu, Claudio Giuliano and Rune Sætre for their generous assistance in providing us with data, software and information about their work on PPI extraction. Further, we thank CSC, the Finnish IT center for science, for providing us extensive computational resources. This work has been supported by the Academy of Finland and the Finnish Funding Agency for Technology and Innovation, Tekes.

This article has been published as part of *BMC Bioinformatics* Volume 9 Supplement 11, 2008: Proceedings of the BioNLP 08 ACL Workshop: Themes in biomedical language processing. The full contents of the supplement are available online at http://www.biomedcentral.com/1471-2105/9?issue=S11

References

1. Hirschman L, Park JC, Tsujii J, Wong L, Wu CH: **Accomplishments and challenges in literature data mining for biology.** *Bioinformatics* 2002, **18(12):**1553-1561.
2. Cohen KB, Hunter L: **Natural language processing and systems biology.** In *Artificial intelligence methods and tools for systems biology, Volume 5 of Computational Biology Springer*; 2004:147-173.
3. Zweigenbaum P, Demner-Fushman D, Yu H, Cohen KB: **Frontiers of biomedical text mining: current progress.** *Briefings in Bioinformatics* 2007, **8(5):**358-375.
4. Bunescu R, Ge R, Kate R, Marcotte E, Mooney R, Ramani A, Wong Y: **Comparative Experiments on Learning Information Extractors for Proteins and their Interactions.** *Artificial Intelligence in Medicine* 2005, **33(2):**139-155.
5. Hunter L, Lu Z, Firby J, Baumgartner WA, Johnson HL, Ogren PV, Cohen KB: **OpenDMAP: An open-source, ontology-driven concept analysis engine, with applications to capturing knowledge regarding protein transport, protein interactions and cell-specific gene expression.** *BMC Bioinformatics* 2008, **9:**78.
6. Pyysalo S, Ginter F, Heimonen J, Björne J, Boberg J, Järvinen J, Salakoski T: **BioInfer: A Corpus for Information Extraction in the Biomedical Domain.** *BMC Bioinformatics* 2007, **8(50):**.
7. Kim JD, Ohta T, Tsujii J: **Corpus annotation for mining biomedical events from literature.** *BMC Bioinformatics* 2008, **9:**10.
8. Lease M, Charniak E: **Parsing Biomedical literature.** In *Proceedings of the Second International Joint Conference on Natural Language Processing, Lecture notes in computer science Springer*; 2005:58-69.
9. de Marneffe MC, MacCartney B, Manning CD: **Generating Typed Dependency Parses from Phrase Structure Parses.** *Proceedings of the Fifth International Conference on Language Resources and Evaluation* 2006:449-454.
10. Bunescu R, Mooney R: **Subsequence Kernels for Relation Extraction.** In *Advances in Neural Information Processing Systems 18 MIT Press*; 2006:171-178.
11. Zelenko D, Aone C, Richardella A: **Kernel methods for relation extraction.** *Journal of Machine Learning Research* 2003, **3:**1083-1106.
12. Bunescu R, Mooney R: **A shortest path dependency kernel for relation extraction.** In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing Association for Computational Linguistics*; 2005:724-731.
13. Gärtner T, Flach PA, Wrobel S: **On Graph Kernels: Hardness Results and Efficient Alternatives.** In *Proceedings of the Sixteenth Annual Conference on Learning Theory and Seventh Annual Workshop on Kernel Machines, Lecture Notes in Artificial Intelligence Springer*; 2003:129-143.
14. Pahikkala T, Tsivtsivadze E, Boberg J, Salakoski T: **Graph Kernels versus Graph Representations: a Case Study in Parse Ranking.** *Proceedings of the Fourth Workshop on Mining and Learning with Graphs* 2006:181-188.
15. Meyer CD: *Matrix analysis and applied linear algebra Society for Industrial and Applied Mathematics*; 2000.
16. Suykens JAK, Vandewalle J: **Least Squares Support Vector Machine Classifiers.** *Neural Processing Letters* 1999, **9(3):**293-300.
17. Rifkin R, Yeo G, Poggio T: *Regularized Least-squares Classification, Volume 190 of NATO Science Series III: Computer and System Sciences Volume chap 7. IOS Press*; 2003:131-154.
18. Pahikkala T, Boberg J, Salakoski T: **Fast n-Fold Cross-Validation for Regularized Least-Squares.** *Proceedings of the Ninth Scandinavian Conference on Artificial Intelligence, Otamedia* 2006:83-90.
19. Fundel K, Kuffner R, Zimmer R: **RelEx-Relation extraction using dependency parse trees.** *Bioinformatics* 2007, **23(3):**365-371.
20. Ding J, Berleant D, Nettleton D, Wurtele E: **Mining MEDLINE: abstracts, sentences, or phrases?** *Proceedings of the Pacific Symposium on Biocomputing* 2002:326-337.
21. Nédellec C: **Learning language in logic – genic interaction extraction challenge.** *Proceedings of the 4th Learning Language in Logic Workshop* 2005:31-37.
22. **Conversions for five PPI corpora** [http://mars.cs.utu.fi/PPICorpora]
23. Pyysalo S, Airola A, Heimonen J, Björne J, Ginter F, Salakoski T: **Comparative Analysis of Five Protein-protein Interaction Corpora.** *BMC Bioinformatics* 2008, **9(Suppl 3):**S6.
24. Airola A, Pyysalo S, Björne J, Pahikkala T, Ginter F, Salakoski T: **A Graph Kernel for Protein-Protein Interaction Extraction.** *Proceedings of the Workshop on Current Trends in Biomedical Natural Language Processing* 2008:1-9.
25. Heimonen J, Pyysalo S, Ginter F, Salakoski T: **Complex-to-pairwise mapping of biological relationships using a semantic network representation.** *Proceedings of the Third International Symposium on Semantic Mining in Biomedicine* 2008:45-52.
26. Clegg AB, Shepherd A: **Benchmarking natural-language parsers for biological applications using dependency graphs.** *BMC Bioinformatics* 2007, **8:**24.
27. Pyysalo S, Ginter F, Laippala V, Haverinen K, Heimonen J, Salakoski T: **On the unification of syntactic annotations under the Stanford dependency scheme: A case study on BioInfer and GENIA.** In *Proceedings of the Workshop on Biological, translational and clinical language processing Association for Computational Linguistics*; 2007:25-32.

28. Giuliano C, Lavelli A, Romano L: **Exploiting Shallow Linguistic Information for Relation Extraction From Biomedical Literature.** *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics* 2006.
29. Van Landeghem S, Saeys Y, Peer Y Van de, De Baets B: **Extracting Protein-Protein Interactions from Text using Rich Feature Vectors and Feature Selection.** *Proceedings of the Third International Symposium on Semantic Mining in Biomedicine* 2008:77-84.
30. Hanley JA, McNeil BJ: **The meaning and use of the area under a receiver operating characteristic (ROC) curve.** *Radiology* 1982, **143:**29-36.
31. Bradley AP: **The use of the area under the ROC curve in the evaluation of machine learning algorithms.** *Pattern Recognition* 1997, **30(7):**1145-1159.
32. Sætre R, Sagae K, Tsujii J: **Syntactic features for protein-protein interaction extraction.** *Second International Symposium on Languages in Biology and Medicine short papers* 2007.
33. Caporaso JG, Deshpande N, Fink JL, Bourne PE, Cohen KB, Hunter L: **Intrinsic Evaluation of Text Mining Tools May Not Predict Performance on Realistic Tasks.** *Proceedings of Pacific Symposium on Biocomputing* 2008:640-651.
34. Miwa M, Sætre R, Miyao Y, Ohta T, Tsujii J: **Combining Multiple Layers of Syntactic Information for Protein-Protein Interaction Extraction.** *Proceedings of the Third International Symposium on Semantic Mining in Biomedicine* 2008:101-108.
35. Miyao Y, Sætre R, Sagae K, Matsuzaki T, Tsujii J: **Task-oriented Evaluation of Syntactic Parsers and Their Representations.** *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies* 2008:46-54.
36. **java Simple Relation Extraction** [http://tcc.itc.it/research/textec/tools-resources/jsre.html]
37. Pyysalo S, Sætre R, Tsujii J, Salakoski T: **Why Biomedical Relation Extraction Results are Incomparable and What to do about it.** *Proceedings of the Third International Symposium on Semantic Mining in Biomedicine* 2008:149-152.
38. Björne J, Pyysalo S, Ginter F, Salakoski T: **How Complex are Complex Protein-protein Interactions?** *Proceedings of the Third International Symposium on Semantic Mining in Biomedicine* 2008:125-128.
39. Mitsumori T, Murata M, Fukuda Y, Doi K, Doi H: **Extracting Protein-Protein Interaction Information from Biomedical Text with SVM.** *IEICE – Transactions on Information and Systems* 2006, **E89-D(8):**2464-2466.
40. Yakushiji A, Miyao Y, Tateisi Y, Tsujii J: **Biomedical information extraction with predicate-argument structure patterns.** *Proceedings of the First International Symposium on Semantic Mining in Biomedicine* 2005:60-69.

Errata for All-paths graph kernel for protein-protein interaction extraction with evaluation of cross-corpus learning

Antti Airola

February 27, 2009

This is an errata for the article All-paths graph kernel for protein-protein interaction extraction with evaluation of cross-corpus learning [1, 2].

The behavior of the implementation of the graph kernel used in our experiments deviates from how the method is described in the article. The difference is in the construction of the matrix G, which is introduced towards the end of the section "The all-paths graph kernel".

Let $L \in \mathbb{R}^{m \times n}$ be the label allocation matrix and $W \in \mathbb{R}^{n \times n}$ the final adjacency matrix of the graph. Then, the definition given for G in the article is

$$G = LWL^{\mathrm{T}}. \tag{1}$$

When written open as a sum this becomes

$$G_{i,j} = \sum_{k=1}^{n} \sum_{l=1}^{n} L_{i,k} W_{k,l} L_{j,l}.$$

However, when computing the values of the entries in matrix G, the implementation used in the experiments erroneously had a reassignment operation in place of the sum operation. The resulting behavior can be very closely approximated by the following definition for G.

$$G_{i,j} = \max_{1 \leq k,l \leq n} \{L_{i,k} W_{k,l} L_{j,l}\}. \tag{2}$$

The resulting difference is that instead of summing together the weights of all paths connecting two labels, we take the maximum over these.

The results reported in [1, 2] are based on using (2). Further experiments seem to indicate, that using the latter definition of G can lead to better

performance, than using the former definition (roughly, by order of 2–3 F-score units on the five corpora). The evidence is however not conclusive, as the differences fall within variance in performance estimation, and contrary results have also been observed in a re-implementation of the method. We note that both (1) and (2) lead to valid kernels, and thus we encourage anyone using the graph kernel to try both variants of the method.

The graph kernel implementation that was made available at `http://mars.cs.utu.fi/PPICorpora/GraphKernel.html` also originally used definition (2). An implementation of definition (1) has been added to the software to facilitate replication efforts.

Acknowledgments

We would like to thank Erik Fässler for bringing this matter to our attention.

References

[1] A. Airola, S. Pyysalo, J. Björne, T. Pahikkala, F. Ginter, and T. Salakoski. All-paths graph kernel for protein-protein interaction extraction with evaluation of cross-corpus learning. *BMC Bioinformatics, special issue*, 9(Suppl 11):S2, 2008.

[2] A. Airola, S. Pyysalo, J. Björne, T. Pahikkala, F. Ginter, and T. Salakoski. A graph kernel for protein-protein interaction extraction. In *Proceedings of the Workshop on Current Trends in Biomedical Natural Language Processing*, pages 1–9, 2008.

Turku Centre for Computer Science
TUCS Dissertations

111. **Camilla J. Hollanti**, Order-Theoretic Methods for Space-Time Coding: Symmetric and Asymmetric Designs
112. **Heidi Himmanen**, On Transmission System Design for Wireless Broadcasting
113. **Sébastien Lafond**, Simulation of Embedded Systems for Energy Consumption Estimation
114. **Evgeni Tsivtsivadze**, Learning Preferences with Kernel-Based Methods
115. **Petri Salmela**, On Communication and Conjugacy of Rational Languages and the Fixed Point Method
116. **Siamak Taati**, Conservation Laws in Cellular Automata
117. **Vladimir Rogojin**, Gene Assembly in Stichotrichous Ciliates: Elementary Operations, Parallelism and Computation
118. **Alexey Dudkov**, Chip and Signature Interleaving in DS CDMA Systems
119. **Janne Savela**, Role of Selected Spectral Attributes in the Perception of Synthetic Vowels
120. **Kristian Nybom**, Low-Density Parity-Check Codes for Wireless Datacast Networks
121. **Johanna Tuominen**, Formal Power Analysis of Systems-on-Chip
122. **Teijo Lehtonen**, On Fault Tolerance Methods for Networks-on-Chip
123. **Eeva Suvitie**, On Inner Products Involving Holomorphic Cusp Forms and Maass Forms
124. **Linda Mannila**, Teaching Mathematics and Programming – New Approaches with Empirical Evaluation
125. **Hanna Suominen**, Machine Learning and Clinical Text: Supporting Health Information Flow
126. **Tuomo Saarni**, Segmental Durations of Speech
127. **Johannes Eriksson**, Tool-Supported Invariant-Based Programming
128. **Tero Jokela**, Design and Analysis of Forward Error Control Coding and Signaling for Guaranteeing QoS in Wireless Broadcast Systems
129. **Ville Lukkarila**, On Undecidable Dynamical Properties of Reversible One-Dimensional Cellular Automata
130. **Qaisar Ahmad Malik**, Combining Model-Based Testing and Stepwise Formal Development
131. **Mikko-Jussi Laakso**, Promoting Programming Learning: Engagement, Automatic Assessment with Immediate Feedback in Visualizations
132. **Riikka Vuokko**, A Practice Perspective on Organizational Implementation of Information Technology
133. **Jeanette Heidenberg**, Towards Increased Productivity and Quality in Software Development Using Agile, Lean and Collaborative Approaches
134. **Yong Liu**, Solving the Puzzle of Mobile Learning Adoption
135. **Stina Ojala**, Towards an Integrative Information Society: Studies on Individuality in Speech and Sign
136. **Matteo Brunelli**, Some Advances in Mathematical Models for Preference Relations
137. **Ville Junnila**, On Identifying and Locating-Dominating Codes
138. **Andrzej Mizera**, Methods for Construction and Analysis of Computational Models in Systems Biology. Applications to the Modelling of the Heat Shock Response and the Self-Assembly of Intermediate Filaments.
139. **Csaba Ráduly-Baka**, Algorithmic Solutions for Combinatorial Problems in Resource Management of Manufacturing Environments
140. **Jari Kyngäs**, Solving Challenging Real-World Scheduling Problems
141. **Arho Suominen**, Notes on Emerging Technologies
142. **József Mezei**, A Quantitative View on Fuzzy Numbers
143. **Marta Olszewska**, On the Impact of Rigorous Approaches on the Quality of Development
144. **Antti Airola**, Kernel-Based Ranking: Methods for Learning and Performace Estimation

Turku Centre *for* Computer Science

Joukahaisenkatu 3-5 B, 20520 Turku, Finland | www. tucs.fi

University of Turku
Faculty of Mathematics and Natural Sciences
- Department of Information Technology
- Department of Mathematics

Turku School of Economics
- Institute of Information Systems Science

Åbo Akademi University
Division for Natural Sciences and Technology
- Department of Information Technologies

ISBN 978-952-12-2674-8
ISSN 1239-1883

Antti Airola

Antti Airola

Antti Airola

Kernel-Based Ranking

Kernel-Based Ranking

Kernel-Based Ranking: Methods for Learning and Performance Estimation