Pekka Naula

Sparse Predictive Modeling

A Cost-Effective Perspective

Turku Centre *for* Computer Science

TUCS Dissertations
No 224, November 2017

Sparse Predictive Modeling
A Cost-Effective Perspective

Pekka Naula

To be presented, with the permission of the Faculty of Mathematics and Natural Sciences of the University of Turku, for public criticism in Auditorium Natura X on November 10, 2017, at 12 noon.

University of Turku
Department of Future Technologies
20014 TURUN YLIOPISTO
Finland

2017

Supervisors

Assistant Professor Tapio Pahikkala
Department of Future Technologies
University of Turku
Finland

Doctor Antti Airola
Department of Future Technologies
University of Turku
Finland

Professor Tapio Salakoski
Department of Future Technologies
University of Turku
Finland

Reviewers

Professor Martti Juhola
Department of Computer Science
University of Tampere
Finland

Professor Willem Waegeman
Department of Mathematical Modelling, Statistics and Bioinformatics
Ghent University
Belgium

Opponent

Professor Tommi Kärkkäinen
Department of Mathematical Information Technology
University of Jyväskylä
Finland

ISBN 978-952-12-3590-0
ISSN 1239-1883

Abstract

Many real life problems encountered in industry, economics or engineering are complex and difficult to model by conventional mathematical methods. Machine learning provides a wide variety of methods and tools for solving such problems by learning mathematical models from data. Methods from the field have found their way to applications such as medical diagnosis, financial forecasting, and web-search engines. The predictions made by a learned model are based on a vector of feature values describing the input to the model. However, predictions do not come for free in real world applications, since the feature values of the input have to be bought, measured or produced before the model can be used. Feature selection is a process of eliminating irrelevant and redundant features from the model. Traditionally, it has been applied for achieving interpretable and more accurate models, while the possibility of lowering prediction costs has received much less attention in the literature.

In this thesis we consider novel feature selection techniques for reducing prediction costs. The contributions of this thesis are as follows. First, we propose several cost types characterizing the cost of performing prediction with a trained model. Particularly, we consider costs emerging from multi-target prediction problems as well as a number of cost types arising when the feature extraction process is structured. Second, we develop greedy regularized least-squares methods to maximize the predictive performance of the models under given budget constraints. Empirical evaluations are performed on numerous benchmark data sets as well as on a novel water quality analysis application. The results demonstrate that in settings where the considered cost types apply, the proposed methods lead to substantial cost savings compared to conventional methods.

Tiivistelmä

Monet käytännön ongelmat erityisesti teollisuudessa, taloudessa tai tekniikan alalla ovat vaikeita mallintaa perinteisin matemaattisin menetelmin analyyttisesti, koska ongelma-alueesta ei ole tarpeeksi tietoa tai ratkaisualgoritmia ei osata muodostaa. Koneoppiminen tarjoaa työkaluja ja menetelmiä, joilla tällaisia ongelmia voidaan ratkaista oppimalla matemaattisia malleja datasta. Koneoppimisalgoritmeja on jo menestyksekkäästi sovellettu useissa tosielämän sovelluksissa kuten esimerkiksi lääketieteellisessä diagnosoinnissa, talousennusteissa ja Internetin hakukoneissa. Koneoppimisella opittujen mallien ennusteet perustuvat piirteisiin, jotka kuvaavat mallille annetun syötteen ominaisuuksia. Käytännössä piirrearvot pitää ostaa, mitata tai tuottaa jollakin tavoin, eikä niitä yleensä saada ilmaiseksi. Piirrevalinnalla tarkoitetaan sellaisten piirteiden poistamista, jotka ovat joko hyödyttömiä ennustamisen kannalta, tai eivät tarjoa lisäinformaatiota muihin piirteisiin verrattuna. Perinteisesti piirrevalintaa on käytetty mallin tulkittavuuden ja tarkkuuden parantamiseen, kun taas piirrevalinnan käyttökelpoisuutta ennustamiskustannusten pienentämiseksi on hyvin vähän tutkittu.

Tässä väitöskirjassa tarkastelemme uudenlaisia piirrevalintamenetelmiä, joilla saavutetaan kustannussäästöjä, kun opittua mallia käytetään ennustamiseen. Aluksi esittelemme uusia kustannustyyppejä, joita esiintyy useita toisiinsa liittyviä tehtäviä samanaikaisesti ratkaistaessa. Lisäksi näytämme miten tällaisia ongelmia voidaan ratkoa kustannustehokkaasti maksimoimalla mallin ennustustarkkuutta annetun budjetin rajoissa. Tätä varten kehitämme ahneeseen pienimmän neliösumman menetelmään perustuvia algoritmeja. Menetelmiä testataan laajasti erilaisilla datajoukoilla, lisäksi uutena sovelluksena näytämme miten tutkimuksessa kehitettyä kustannustehokasta lähestymistapaa voidaan soveltaa vedenlaadun mittaukseen. Tulokset osoittavat, että kustannustyyppien huomioiminen piirrevalinnassa usein johtaa merkittäviin kustannussäästöihin.

Acknowledgements

First, I would like to thank my research director and supervisor Tapio Salakoski, for taking me in to his machine learning team and providing me funding for my doctoral studies. He has given me the freedom to concentrate on those research issues which I consider to be important in the area of machine learning, and has always been supportive and given good advice when needed. Without his important support, most of the work could not have been done. I owe my deepest appreciation to my excellent supervisors Tapio Pahikkala and Antti Airola, who gave me an introduction to machine learning, gave me tools and knowledge to conduct my research and motivated me to continue studies when needed over the years. They have acted as role models of scientific research to me, showing always high work ethics and trusting on hard work and they are co-authors of all my publications. So once again, thank you very much, Tapio and Antti.

During the years I have worked on this doctoral dissertation, I have been guided and helped by many people. First I owe a big appreciation to Sebastian Okser for taking the first steps in research on machine learning with me. Thank you for being both a colleague and a friend during my early years as a researcher. I would like to thank Paavo Nevalainen, Jonne Pohjankukka, Markus Viljanen and Filip Ginter for your valuable comments on my research and always such fascinating ideas and funny discussions. I owe a debt of gratitude to my fellow researchers, Sari Pihlasalo, Ileana Montoya Perez and Parisa Movahedi. You introduced several interesting application domains to me and provided help with the data when I was testing and validating my algorithms.

I express my gratitude to my former employer, Nokia Mobile Phones, for allowing me the opportunity to deepen my knowledge in the area of artificial intelligence, thus connecting my passion to smart computing and industrial applications. Special thanks go to my Nokia mentors and colleagues, Pekka Isomursu and Ari Siuvatti, who introduced the world of fuzzy logic and its applications to me. I would like to thank the leaders of the Nokia research, Urpo Tuomela and Pertti Huuskonen, for giving me freedom to study many fascinating future approaches such as context sensitivity. I also want to acknowledge some former colleagues for being part of the team which

gave birth to multiple smart innovations. Thank you Jakke Mäkelä, Niko Porjo, Terho Kaikuranta, Seppo Salminen, Mikko Juhola, Toni Östergård and Marko Parikka.

A number of organizations have provided infrastructure that made my research possible. The Turku Centre for Computer Science (TUCS) and the Department of Future Technologies at the University of Turku have been vital to my research, and I am grateful to the entire staff working there. I would like to thank CSC, the Finnish IT center for science, for providing me with extensive computational resources. Further, I would like to acknowledge Tomi Suovuo for his effort to inspect my thesis and Sami Nuuttila for his continuous help to solve my IT problems over the years.

I am greatful to Martti Juhola and Willem Waegeman for their valuable comments and criticisms on this dissertation. Moreover, I would like to thank Tommi Kärkkäinen for acting as my opponent.

Finally, I am deeply thankful to my parents Maila and Pertti, and my brother Mika and his family for their unwavering support, love and encouragement throughout my life.

List of original publications

I Tapio Pahikkala, Antti Airola, Pekka Naula, Tapio Salakoski. Greedy RankRLS: a Linear Time Algorithm for Learning Sparse Ranking Models. In: Evgeniy Gabrilovich, Alexander J. Smola, Naftali Tishby (Eds.), SIGIR 2010 Workshop on Feature Generation and Selection for Information Retrieval, 11–18, ACM, 2010.

II Pekka Naula, Tapio Pahikkala, Antti Airola, Tapio Salakoski. Learning Multi-Label Predictors under Sparsity Budget. In: Anders Kofod-petersen, Fredrik Heintz, Helge Langseth (Eds.), Proceedings of the Eleventh Scandinavian Conference on Artificial Intelligence (SCAI 2011), Frontiers in Artificial Intelligence and Applications 227, 30-39, IOS Press, 2011.

III Pekka Naula, Tapio Pahikkala, Antti Airola, Tapio Salakoski. Greedy Regularized Least-Squares for Multi-Task Learning. In: Myra Spiliopoulou, Haixun Wang, Diane Cook, Jian Pei, Wei Wang, Osmar Zaane, Xindong Wu (Eds.), 11th IEEE International Conference on Data Mining Workshops (ICDMW'11), 527–533, IEEE Computer Society, 2011.

IV Pekka Naula, Antti Airola, Tapio Salakoski, Tapio Pahikkala. Multi-Label Learning Under Feature Extraction Budgets. Pattern Recognition Letters 40, 56-65, 2014.

V Pekka Naula, Antti Airola, Tapio Salakoski, and Tapio Pahikkala. Learning low cost multi-target models by enforcing sparsity. In Moonis Ali, Young Sig Kwon, Chang-Hwan Lee, Juntae Kim, and Yongdai Kim, editors, The 28th International Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems (IEA/AIE 2015), Lecture Notes in Computer Science, pages 252–261. Springer, 2015.

VI Pekka Naula, Antti Airola, Sari Pihlasalo, Ileana Montoya Perez, Tapio Salakoski, Tapio Pahikkala. Assessment of metal ion concentration in water with structured feature selection. Chemosphere, 185:1063–1071, 2017.

Contents

Chapter 1

Introduction

1.1 Data and intelligent processing

There is a vast amount of data available nowadays. Data is been produced by human beings in their everyday lives when they are filling questionnaires, diagnosed in a hospital, buying groceries in super markets or making phone calls to friends and so on. Technology around us is more and more ubiquitous and wireless meaning that sensors are embedded in everywhere in small consumer electronics, traffic, vehicles and homes to support our lives. Sensors measure light, air pressure, acceleration, motion, temperature, moisture, blood pressure etc. However, data sources such as sensors are not free, and their use can also result in costs. These costs need to be taken into account when designing intelligent systems with limited budget.

Nowadays, such sensor data is used in many devices or industry applications. For example, consider a toaster that heats the bread until correct temperature is reached or a refrigerator that monitors the items inside the device and warns if a consumer is lacking of fresh milk or vegetables. Similarly, a sensor controlled vacuum cleaner is able to navigate in a room and clean up dirt. Moreover, automatic traffic lights may turn to green if an approaching car is the only vehicle in the crossing at the time. In addition to these small scale tasks, automatic cars are already now mature enough to navigate and drive automatically on streets without human interaction. In order to analyze data intelligently, turn data to knowledge, make decisions from data, or control devices automatically, we need intelligent methods to reason from data.

There are a numerous different approaches invented to extract information from the data, or create mathematical predictive models to classify, rank or regress things. *Data mining* (see e.g. Witten et al. (2011); Adriaans and Zantinge (1997)) contains tools and methods to extract patterns from the data. *Machine learning* (see e.g. Bishop (2006) is a related discipline to

build mathematical models by learning from past observations. The learned predictive models can then be used to predict on unseen data.

1.2 Thesis background

The motivation to write this thesis and investigate the issues considered here originates back to the 1990s when I was working in a big telecommunication company. Looking back at that time now those years can be seen as a time period when the world turned towards ubiquitous technology. It was a turning point when many digital innovations such as mobile phones, world wide web, and personal computers, were seen for the first time at every home. My motivation is based on to two central questions encountered in industry applications. First, how to automatically produce intelligent behavior in smart systems, based on data gathered from sensors. Second, how to lower the costs related to such systems.

During the industry years, I was working in many novel projects where our goal was to innovate, design and implement smart applications either into prototypes or real final products. Such innovations include for example lightning detector (Mäkelä, 2009), ghost keyboard, touch-based data transfer, context sensitive devices and so on just to mention a few.

Brand new innovations often require concept testing before the designing of final products can be started. One needs a working prototype in order to demonstrate the functionality of the device for the user experience team. Additionally, some innovations are too complex to be modeled and implemented without intelligent techniques such as machine learning. The machine learning methods are dependent on the data, that is used in training, testing and prediction phase. While research and development require resources, when products are mass produced the most important costs are the manufacturing and product use costs. Each component included in a manufacture device have a price. Performing each measurement for gathering data may also cost time or money. A central question in innovative R&D projects is how to implement the acceptable quality products when subject to hard budget constraints on the cost of the solution.

1.3 Problem setting

Machine learning methods can be used to model a wide range of problems that are difficult or even impossible to solve analytically with conventional mathematics or statistics. In this thesis, we restrict our considerations to the supervised machine learning setting, where both the inputs and the outputs to be predicted are available in the training data while learning the model.

The final model can then be used to predict outputs for new inputs, for which they are unknown.

Feature selection is a process that is usually used in the machine learning literature in order to improve predictive performance and understandability of learned models. This is accomplished by means of reducing redundant and irrelevant features from the model. However, the perspective that feature selection can also provide cost sensitive models has been less studied. In this thesis, we study how one can take account of costs related to prediction using feature selection. Further, we introduce practical machine learning algorithms that optimize these costs.

While some previous work (Xu et al., 2012; Huang et al., 2009) has been done on cost-effective learning for simple classification and regression problems, we focus in this work on more complex structured costs that often occur in real-world problems. For example, in many applications one encounters problems, where multiple prediction tasks have to be solved together. These include multi-class and multi-label learning problems that are typically solved by training several binary classifiers on shared inputs. In such settings prediction costs can be lowered by using such feature selection methods that find a common set of features suitable for all the tasks. This and other structured feature selection problems, such as those concerning grouped and Cartesian features, are considered in this thesis.

1.4 Research objectives

The main focus of this thesis is to develop novel machine learning methods for learning accurate predictive models when subject to budget constraints. We restrict our considerations to the family of linear models, while most of the introduced ideas can also be extended to non-linear models. Specifically, we consider the following two research questions:

1 What type of feature costs should one be aware of when doing predictions with a trained model?

2 How to maximize predictive performance of the machine learning model given a budget limit on the costs?

1.5 Organization of the thesis

This thesis consists of two separate parts that both together form the contribution of the research. Part I of the thesis (Chapters 1-5) gives an introduction to the topic by presenting the background for machine learning, as well as our main findings concerning the research questions. Part II presents

the six original research publications that were written during the doctoral studies at the university.

In Chapter 2 we give an introduction to machine learning, shortly explaining the main concepts, and providing the theory for key methods such as ridge regression and multi-task learning. The main contribution of the thesis is presented in the following two chapters. Chapter 3 covers feature cost types that occur when making predictions with machine learning models. Chapter 4 concerns training algorithms that take account of the feature costs. Chapter 5 concludes Part I of the thesis and draws guidelines for future work.

Chapter 2

Supervised machine learning

In this section we introduce the most essential linear modeling methods that we will apply in the thesis. We will provide a formal presentation for such well-known approaches as least-squares, ridge regression and lasso. We also present a variety of different problem types, one can encounter in the context of multiple tasks.

We use the following notations through the whole thesis along with the vectors and matrices. Vectors are denoted with a bold lowercase letter, that is $\mathbf{v} \in \Re^d$ for the vector, whose i:th component is v_i. Equivalently, we reserve a bold uppercase representation for matrices, $\mathbf{A} \in \mathbb{R}^{n \times d}$. Now, $a_{i,j}$ denotes the element of the matrix $\mathbf{A}$ in its i:th row and j:th column. Further, let R and C denote index sets containing a subset of possible row indices or column indices, respectively. Now, we denote the sub matrix that contains only those rows that are indexed in R as $\mathbf{A}_R$, only those columns indexed in C as $\mathbf{A}_{:,C}$, or both as $\mathbf{A}_{R,C}$. Particularly, a sub matrix that contains only one row or column, is denoted as $\mathbf{A}_i$ and $\mathbf{A}_{:,j}$, where i corresponds to i:th row and j corresponds to j:th column, respectively. Finally, we may refer to multiple related matrices by using upper index notation, so that $\mathbf{A}^i$ refers to the i:th matrix.

2.1 Introduction

Given a set of training data, a machine learning algorithm builds a mathematical model that can be used to perform predictions on unseen data. In this work we assume a standard supervised learning setting, where we are given an input space $\mathcal{X} = \Re^d$ and output space $\mathcal{Y} = \Re^e$ and a random sample D of examples $\{(\mathbf{x}_i, \mathbf{y}_i) : i = 1 \dots n\}$ generated independently and identically (i.i.d) from some unknown joint probability distribution P over $\mathcal{X} \times \mathcal{Y}$. Training inputs are multi-dimensional vectors in which each component represents feature values that have been collected, measured or

produced. The output for each example might be a single real or binary value, or a vector of such values.

The aim of learning is to infer such a hypothesis that is optimal with respect to a given error measure L and the probability measure P. In this work, we restrict our considerations to the set of linear functions of the form $\mathbf{Wx}$, where $\mathbf{W} \in \mathbb{R}^{e \times d}$. Finding the optimal linear functions can be expressed as the following minimization problem:

$$\mathbf{W}^* = \underset{\mathbf{W} \in \mathbb{R}^{e \times d}}{\operatorname{argmin}} \int_{\mathcal{X} \times \mathcal{Y}} L(\mathbf{Wx}, \mathbf{y}) dP(\mathcal{X} \times \mathcal{Y}), \tag{2.1}$$

In practice we can never evaluate this, but rather use instead the empirical risk measured on the training set.

$$\mathbf{W}^* = \underset{\mathbf{W} \in \mathbb{R}^{e \times d}}{\operatorname{argmin}} \sum_{i=1}^{n} L(\mathbf{Wx}_i, \mathbf{y}_i). \tag{2.2}$$

As an important special case of this setting, we will consider also single task prediction problems, where $e = 1$ and instead of the matrix $\mathbf{W}$ we store the coefficients of the linear model in column vector $\mathbf{w} \in \mathbb{R}^d$.

2.2 Least-squares

First, let us consider single-task regression with the least-squares loss $(y - \mathbf{x}^T\mathbf{w})^2$. One of the most popular methods used to estimate the coefficients (see e.g. (Hastie et al., 2001)) is the method of *least-squares* (LS), where the aim is to minimize the residual sum of the squares. Let $\mathbf{X} \in \mathbb{R}^{n \times d}$ be a matrix containing the feature representation of the inputs in the training set, where n is the number of inputs and d is the number of features. Moreover, let $\mathbf{y} \in \mathbb{R}^n$ be a vector containing the labels in the training set. The least-squares error is

$$J(\mathbf{w}) = (\mathbf{y} - \mathbf{Xw})^T(\mathbf{y} - \mathbf{Xw}). \tag{2.3}$$

The residual error can now easily to be minimized by differentiating $J(\mathbf{w})$ with respect to $\mathbf{w}$ and setting the derivative to zero, resulting in the solution

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \tag{2.4}$$

Linear regression can also be used for classification. For example, let y be a binary output so that -1 and +1 encode the two classes. Further, let the predicted output be $\hat{y}$. Now the predicted real-value output can be categorized to the class value according to the formula:

$$K = \begin{cases} -1 & \text{if } \hat{y} < 0 \\ +1 & \text{if } \hat{y} \geq 0 \end{cases},$$

where K denotes the output of the classification task.

2.3 Norms

Norms are often used to regularize the risk in order to achieve lower complexity models that avoid over-fitting.

Definition 1 *The norm of vector* $\mathbf{w}$ *is a real number, denoted* $\|\mathbf{w}\|$*, that satisfies the following properties (generalizes to matrices):*

1. *Non-negative:* $\|\mathbf{w}\| \geq 0$.
2. *Positive:* $\|\mathbf{w}\| = 0 \iff \mathbf{w} = 0$.
3. *Homogeneous:* $\|\alpha \cdot \mathbf{w}\| = |\alpha| \cdot \|\mathbf{w}\|$ *for all complex scalars* α.
4. *Triangle inequality:* $\|\mathbf{w}_1 + \mathbf{w}_2\| \leq \|\mathbf{w}_1\| + \|\mathbf{w}_2\|$.

A related concept popular in machine learning, that is however not a proper norm, is the so-called zero-norm $\|\cdot\|_0$, that returns the number of non-zero components of the vector. It is defined as

$$\|\mathbf{w}\|_0 = |\{w_i \neq 0, i = 0, \ldots, d\}| \,. \tag{2.5}$$

2.3.1 Vector norm(s)

Vector norms are often used in single task problems to introduce sparsity in the coefficients of the model. The most used vector norms are Euclidean norm also known as l_2-norm, denoted by $\|\cdot\|_2$, which is defined as

$$\|\mathbf{w}\|_2 = \sqrt{\mathbf{w}^{\mathrm{T}} \cdot \mathbf{w}}, \tag{2.6}$$

Manhattan norm or l_1-norm

$$\|\mathbf{w}\|_1 = \sum_{i=1}^{d} |w_i| \,, \tag{2.7}$$

and l_∞-norm

$$\|\mathbf{w}\|_\infty = \max_{1 \leq i \leq d} |w_i| \,. \tag{2.8}$$

2.3.2 Matrix norm(s)

The matrix norm is an extension to the vector norm. Matrix norms are applicable as regularizers when predicting multiple outputs. One of the most useful matrix norms is the Frobenius norm:

$$\|\mathbf{W}\|_F = \sqrt{\sum_{i=1}^{e} \sum_{j=1}^{d} w_{ij}^2}. \tag{2.9}$$

The combined norm (matrix norm) $l_{1,\infty}$ is used to encourage sparsity simultaneously for the multiple tasks in many of our baseline experiments.

$$l_{1,\infty} = \sum_{j=1}^{d} \|\mathbf{W}_{:,j}\|_{\infty}. \tag{2.10}$$

The other popular combined norm to be used to couple tasks together and thus introduce sparsity in matrices is $l_{1,2}$-norm, which can be presented equivalently as follows:

$$l_{1,2} = \sum_{j=1}^{d} \|\mathbf{W}_{:,j}\|_{2}. \tag{2.11}$$

2.4 Regularized least-squares

Ordinary least-squares is prone to overfitting. It's well known in the theory of machine learning that shrinking the coefficients provides a remedy for this problem. This can be achieved by using norms to penalize the model coefficients. Two of the most popular regularized methods are ridge regression and lasso, which are regularized by the l_2 and l_1 norms, respectively.

2.4.1 Ridge regression

By adding a new term into the objective function J for LS, we get the l_2-regularized least-squares problem also known as *ridge regression* (see e.g. Hoerl and Kennard (1970))

$$J(\mathbf{w}) = (\mathbf{y} - \mathbf{X}\mathbf{w})^{\mathrm{T}}(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda \mathbf{w}^{\mathrm{T}}\mathbf{w}, \tag{2.12}$$

where λ is the regularization parameter which controls the trade-off of empirical risk and the amount of shrinking.

The minimizer to this objective can easily be found by taking the derivative and setting it to zero analogously to unregularized least-squares. Thus, we get the solution

$$\mathbf{w} = (\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda \mathbf{I})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}, \tag{2.13}$$

where $\mathbf{I}$ is an identity matrix.

2.4.2 Lasso

Lasso is an l_1-constrained method, introduced by Tibshirani (1994), for sparse variable selection, also known as *basis pursuit* (Chen et al., 2001) in the area of signal processing. Lasso shrinks the coefficients of the linear model similarly to ridge regression, additionally it also tends to increase the

number of zero coefficients in the model thus introducing sparsity. Objective function J for the lasso is

$$J(\mathbf{w}) = (\mathbf{y} - \mathbf{X}\mathbf{w})^{\mathrm{T}}(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda\|\mathbf{w}\|_1. \tag{2.14}$$

Unlike ridge regression lasso has no closed form solution. Computation of the solution to the Eq. (2.14) is a quadratic programming problem. There exist efficient algorithms to solve this problem such as least angle regression (Efron et al., 2004), and coordinate descent, for instance.

2.5 Multiple tasks

Many of the machine learning methods in academic research address problems which contain only one binary label or real-valued target. In general, real-world problems are more complex in nature. The outputs can consist of vectors of target values, and more generally, one may want to solve several related yet separate prediction tasks simultaneously. Further, sometimes a problem is easier to solve by dividing it into smaller pieces and then solve all the sub problems separately. We present different transformation approaches that allow dividing problems with multiple tasks into several simple single-task problems that can be solved with conventional methods.

2.5.1 Multi-class problem

Multi-class classification refers to the setting where each example belongs to exactly one of several different classes. For example, hand-written digits have to be classified to exactly one class (0-9). The goal is to learn a function which will correctly predict the classes for new inputs. Formally, let L denote the set of classes. If $|L| = 2$, then the learning problem is called binary classification problem. In case of $|L| > 2$, we are referring to multi-class classification problem.

Multi-class problems can be transformed to multiple binary classification problems by using the one-vs-all method. Other methods also exist, but one-vs-all has been shown to be a very competitive approach (Rifkin and Klautau, 2004). As a result of transformation, there are $|L|$ independent binary classification problems to be solved. For new data, the class with the highest predicted value is selected.

2.5.2 Multi-label problem

In real-world situations, one often encounters settings where several labels may be associated with each input. For example, the same document can belong to many different categories (sport, news, animal). Such prediction

problem is called *multi-label classification* (Tsoumakas et al., 2010). Analogously, one may consider prediction tasks with real-valued targets, called *multi-target regression* (also known as multivariate or multi-output regression).

There are a wide variety of applications in the area of multi-label classification. For instance, Trohidis et al. (2011) consider the multi-label task of identifying emotions in music; in scene classification (Boutell et al., 2004) the task is to categorize images into semantic classes such as beaches or sunsets; genes are associated with a set of several functional classes in bioinformatics (Elisseeff and Weston, 2001).

There are two main approaches to solving multi-label problems, problem transformation methods and algorithm adaptation methods. Two of the most popular transformation methods are binary relevance (BR) and label power-set (LP) method, both of which transform the original problem to smaller binary problems. Whereas BR divides multi-label problems into single-task problems, one task per label, LP creates binary single-task problems for each label combination. BR ignores possible interactions between the labels due to independently learned predictors, whereas LP is computationally expensive due to the exponential number of possible label combinations, and it is sensitive to a small amount of data. While learning the model for each label or label combination, the data matrix is the same for each of the tasks. All the tasks can be solved by using conventional methods developed for single-task classification. Given new data, each model predicts labels separately. The other major approach to solving multi-label problems is called algorithm adaptation. As examples of such algorithms, we mention multi-label extensions for the k-nearest neighbors (Zhang and Zhou, 2005) and decision trees (Clare and King, 2001).

Multi-label classification requires different evaluation measures than conventional single-label classification, because predictions can be partly correct meaning that a few of the labels can be associated correctly with an input and the rest are not. The evaluation measures are considered to be divided into two categories, example-based (Hamming loss, accuracy, etc.) and label-based (micro-precision, micro-recall, etc.) measures (Tsoumakas and Vlahavas, 2007; Nowak et al., 2010; Madjarov et al., 2012).

2.5.3 Multi-task problem

Multi-task learning (Caruana, 1997) is a setting, that concerns several separate tasks to be solved simultaneously, meaning that all the tasks have their own training data but they share the feature space. Multi-task problem is a generalization of the multi-label problem and the multi-class problem. Formally, we are given the input space $\mathcal{X}$ and the output space $\mathcal{Y}_l$, and several random samples $D_k, k = 1 \dots e$ of examples $\{(\mathbf{x}_i^k, \mathbf{y}_i^k) : i = 1 \dots n_k\}$ gener-

ated from some unknown joint probability distributions $P_1, \ldots, P_e$, where n_k denotes the number of examples in sample D_k and e denotes the number of tasks. The empirical risk over the training sets is defined as follows:

$$\sum_{l=1}^{e} \sum_{i=1}^{n_l} L(\mathbf{W}^l \mathbf{x}_i^l, \mathbf{y}_i^l). \tag{2.15}$$

In this thesis we build linear models for each of the tasks so that each model shares the same subset of features.

Chapter 3

Costs of predictive models

This chapter motivates the importance of cost-sensitive machine learning by presenting several different cost types (see Turney (2000); Krishnapuram et al. (2011)) that should be considered during a construction of a mathematical model from data and using it to predict on unseen data. We consider application scenarios that result in different types of prediction costs. Further, we show what type of sparsity patterns minimizing these costs leads to, when using linear models.

3.1 Multiple linear models

In this thesis, we consider linear models that relate the outputs to be predicted to input features (see Section 2.1). Various types of multi-task learning problems are considered, including multi-class, multi-label or multi-target problems that are transformed in our considerations into multiple single-task problems. Thus the word *task*, corresponds to a single value to be predicted, either a binary class label or a real number. The coefficients of the linear models are stored in the matrix $\mathbf{W}$ so that rows and columns correspond to tasks and features, respectively (see below).

$$\mathbf{W} = \begin{array}{c} \text{coefficients} \\ task_1 \\ task_2 \\ \vdots \\ task_e \end{array} \begin{array}{c} \begin{array}{cccc} f_1 & f_2 & \dots & f_d \end{array} \\ \begin{pmatrix} w_{11} & w_{12} & \dots & w_{1d} \\ w_{21} & w_{22} & \dots & w_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ w_{e1} & w_{e2} & \dots & w_{ed} \end{pmatrix} \end{array}$$

A linear model containing just a few non-zero coefficients, is said to be sparse. Sparsity of the linear model is often seen to be a desirable property because having fewer coefficients makes the model easier to interpret. Moreover, sparse models are less prone to overfitting than dense ones. In this thesis, we however concentrate on the third advantage of sparsity, that

is, the cost reduction. All redundant or irrelevant features in the model increase the cost of prediction. Next, we consider different types of costs associated with making predictions, and the types of *sparsity patterns* in $\mathbf{W}$ that allow lowering such costs.

3.2 Prediction cost

Once the linear models are learned and the relevant features are selected from the set of candidate features, all coefficients in $\mathbf{W}$ corresponding to the selected features are fit to the training data and the rest are set to zero. The final models can then be employed for prediction by multiplying them with the feature vector of a given input $\mathbf{x}$. The feature values required in the input vector $\mathbf{x}$ may originate, for instance, from some device, such as a temperature sensor embedded into a mobile phone, a blood tester in a hospital or a satellite up in the sky sending humidity data down to Earth for some weather forecast system, et cetera. Accordingly, one may have to pay a price for acquiring the feature values in prediction time. We refer to this cost as *feature cost* and the total prediction cost is the sum of feature costs of all selected features. These costs can be reduced by selecting a small number of features or by selecting only inexpensive features. However, removing relevant but expensive features from the model may lead to a lower prediction performance. Therefore, one has to find a suitable trade-off between prediction performance and cost (Yang and Honavar, 1998; Shalev-Shwartz et al., 2010; Min et al., 2011).

Example 1 *Consider a smart device that calculates calorie consumption of the user during exercise. Possible data sources include sensors embedded to the device (accelerometer, barometer, GPS, temperature sensor, heart rate sensor etc.) or components for inputting questionnaire data (sex, age, weight, etc.). The task is to determine such (software and hardware) components that allow predicting calorie consumption as accurately as possible, while staying under a given budget limit on the total cost of the components.*

Example 2 *Consider a doctor providing diagnosis for a patient. There are many possible tests to choose from, each having some cost. These include questionnaires, where the doctor asks some preliminary questions from the patient, blood tests, heart rate monitoring, medical imaging scans etc. The doctor has to choose which tests would be the most useful for making a reliable diagnosis, while avoiding unnecessary and too costly tests. Moreover, a doctor might want to screen for several possible diseases based on same tests.*

In Example 1, the features are derived from the sensors embedded into mobile devices, whereas in Example 2 the features are measured in the medi-

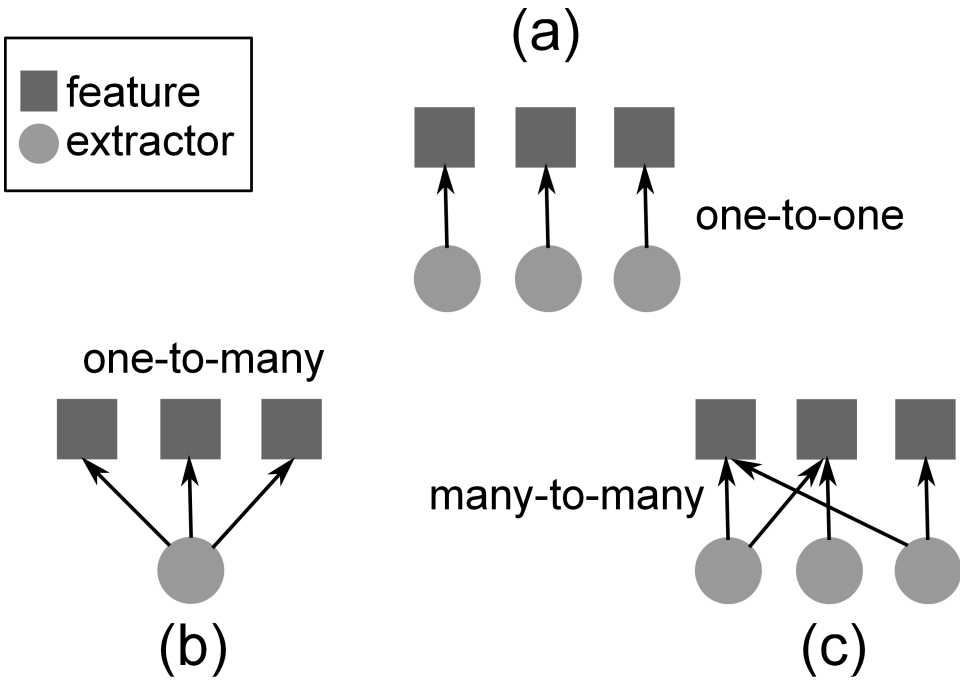

Figure 3.1: Three settings

cal tests done for the patient. We refer to a source that generates features as *feature extractor* and by *feature extraction* we refer to the procedure, where such extractors are used to obtain feature values. Feature extractors may be mapped to the generated features in various ways (three such scenarios are presented in Figure 3.1). In the first setting (a), each extractor generates exactly one feature (one-to-one mapping). Setting (b) presents a setup where one extractor produces two or more features (one-to-many mapping). As an example, consider a group of related features that are derived from the same sensor signal. The last setting (c) shows an approach (fusion, many-to-many mapping) where several extractors may produce each feature. The many-to-one mapping is not separately considered, as it can be mapped to the one-to-one setting. The prices of the extractors are referred to as *extractor costs*, whereas the costs of using the extractors are referred to as *extraction costs*. The extractor cost has to be paid only once, so it can be considered as an initial investment to be able to make predictions. In contrast, the extraction cost must be paid every time a prediction is done.

In this thesis we assume that there always exists a budget, k, restricting the allowed cost of the prediction. Depending on the application, the budget may be set on extractor or extraction cost. However, the training budget is assumed to be unlimited. It is common in industrial applications that development costs are marginal compared to the costs that will occur when

the produced model is deployed. Considering Example 1, the cost of creating prototypes that may include many expensive sensors most of which are later discarded is insignificant compared to the costs that occur when the final products are mass produced. Similarly, considering Example 2, when designing a new diagnostic method we may perform a large array of tests, whereas costs must be managed when applying the model in hospitals.

3.3 Separate cost

Let us recall (see Section 3.1), that we have trained multiple linear models, whose coefficients are stored as rows in matrix $\mathbf{W}$. *Separate cost* represents the cost corresponding to the average number of non-zero coefficients over the tasks, that is, the average number of features needed in prediction. Formally, the separate cost is defined as

$$C_{\mathsf{separate}}(\mathbf{W}) = \frac{1}{e} \sum_{t=1}^{e} \|\mathbf{W}_t\|_0, \tag{3.1}$$

where $e \in \mathbb{N}$ is the number of the tasks and $\|\cdot\|_0$ is the l_0-norm of the matrix row $\mathbf{W}_t$.

Figure 3.2 presents four different distributions of the coefficients over the tasks in $\mathbf{W}$. Clearly, non-zero coefficients are distributed freely in (a) compared to non-zero coefficients in (b), where one can recognize some type of ordering. Despite of different distributions of the non-zero coefficients, both matrices have the same separate cost ($C_{\mathsf{separate}}(\mathbf{W}) = 2$).

Separate cost is a reasonable assumption, whenever extraction cost is considered, and predictions are needed only for a single task at a time. The cost can be easily optimized, by solving each task independently as a separate problem. However, the cost is not realistic when making simultaneous prediction with multiple models, or when considering extractor costs. For these cases, we introduce next the concept of a joint cost.

3.4 Joint cost

Joint cost represents the cost corresponding to the number of columns of which contain at least one non-zero coefficient. Formally, the joint cost is defined by means of combined $l_{0,\infty}$-norm as

$$C_{\mathsf{joint}}(\mathbf{W}) = \left|\{j \mid \exists i, w_{i,j} \neq 0\}\right|, \tag{3.2}$$

where $\mathbf{W}$ is the coefficient matrix over tasks.

Again, consider the matrices in (a) and (b) in Figure 3.2, where both matrices contain the same number of non-zero coefficients but the joint cost

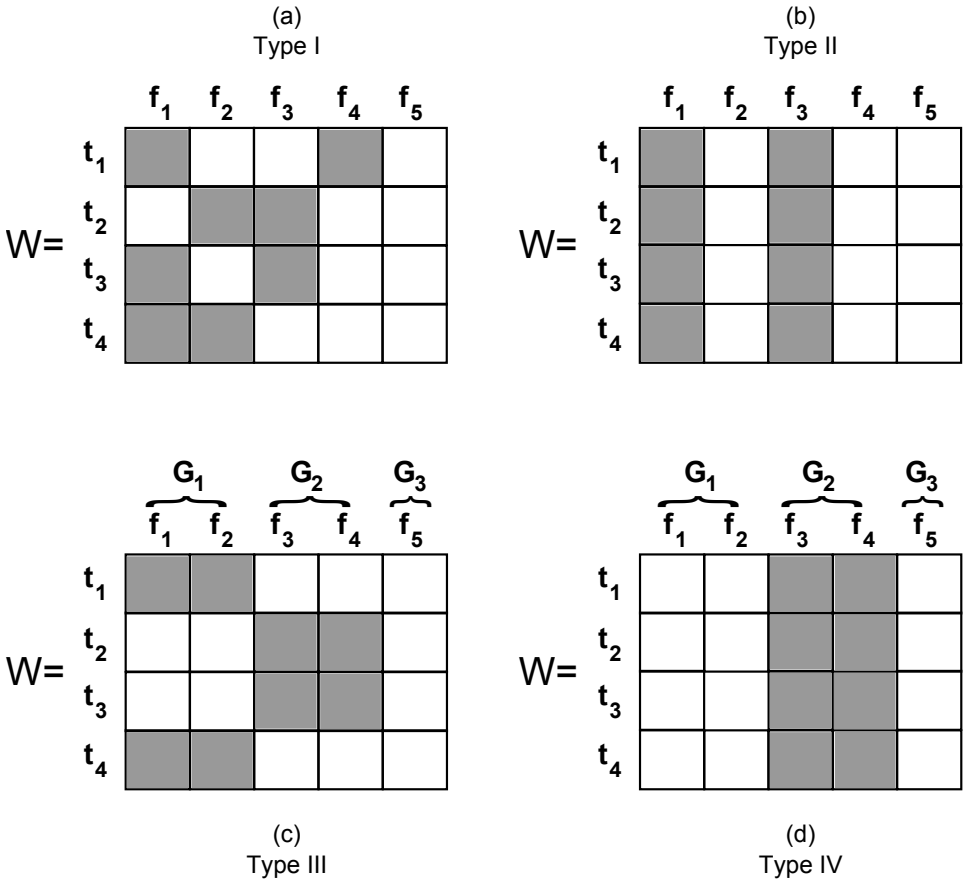

Figure 3.2: Four alternative ways to select features from the candidate set of $\{f_1, f_2, f_3, f_4, f_5\}$ for four tasks. The group structure of the features is known in advance in (c) and (d), that is $\{G_1, G_2, G_3\}$. The rows correspond to tasks and the columns correspond to features. Shaded squares represent non-zero coefficients, whereas empty squares represent zero coefficients.

is different. In Figure 3.2 the joint cost for the matrix in (a) is $C_{\mathsf{joint}}(\mathbf{W}) = 4$, whereas the cost of the matrix in (b) is $C_{\mathsf{joint}}(\mathbf{W}) = 2$ by means of the joint cost. In general, the joint cost is minimized by selecting the same features for all the tasks.

The joint cost is a natural choice when considering the extractor costs of prediction, since each feature that needs to be generated for at least one task needs also an extractor. For extraction costs, joint cost corresponds to applications where predictions are always made simultaneously for all the tasks. This setting appears for example in standard multi-class, and multi-label classification, as well as multi-target regression settings.

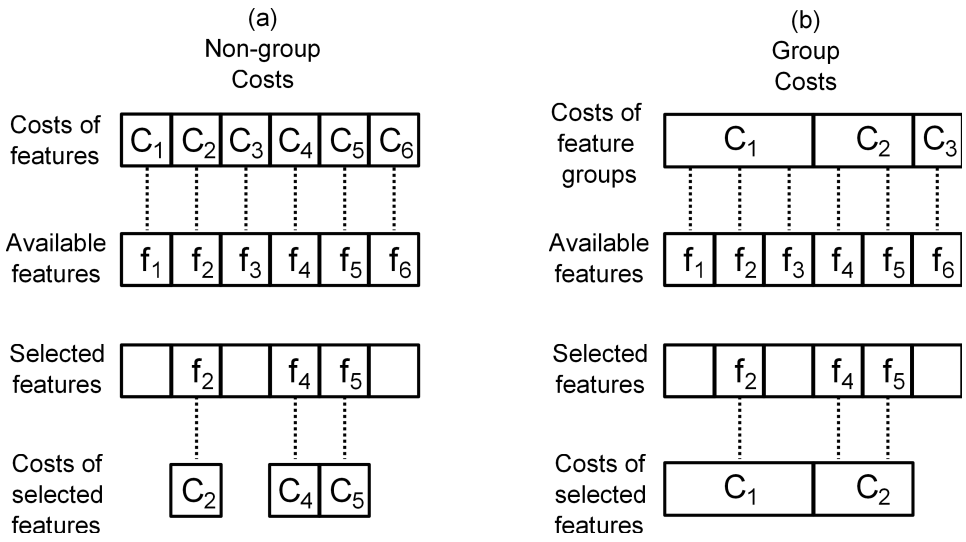

Figure 3.3: Figure shows an example of costs for six features. On the left (a), every single feature is associated with a cost, whereas on the right (b) each feature group $\{f_1, f_2, f_3\}$, $\{f_4, f_5\}$ and $\{f_6\}$ is associated with a cost.

3.5 Group cost

In some applications several features may be derived from the same extractor resulting in a shared cost for the whole group of features (see Figure 3.3). We extend the definitions of separate cost for the group structure by defining *group separate cost* as the average number of non-zero groups of coefficients over the tasks. Equivalently, *group joint cost* refers to the cost corresponding to the number of column groups that contain at least one non-zero coefficient. In order to formalize the above notions, we assume that features are divided into disjoint groups $G_i \subset \{1, \ldots, d\}$, $i = 1, \ldots, J$ and $G_i \cap G_j = \emptyset, i \neq j$, where d is the number of features and J is the number of the groups. Now, group separate cost is given as follows:

$$C_{\text{group separate}}(\mathbf{W}) = \frac{1}{e} \sum_{t=1}^{e} \left| \{ \mathbf{W}_{t,G_i} \neq \mathbf{0}, i = 1, \ldots, J \} \right| . \tag{3.3}$$

and group joint cost as follows:

$$C_{\text{group joint}}(\mathbf{W}) = \left| \{ G_i \mid \exists t, \mathbf{W}_{t,G_i} \neq \mathbf{0} \} \right| , \tag{3.4}$$

where $\mathbf{W}$ is the matrix of coefficients over the tasks.

The average group separate cost for both matrices in (c) and (d) in Figure 3.2 is $C_{\text{group separate}}(\mathbf{W}) = 1$. On the other hand, the group joint cost in (c) is $C_{\text{group joint}}(\mathbf{W}) = 2$ and in (d) it is only $C_{\text{group joint}}(\mathbf{W}) = 1$.

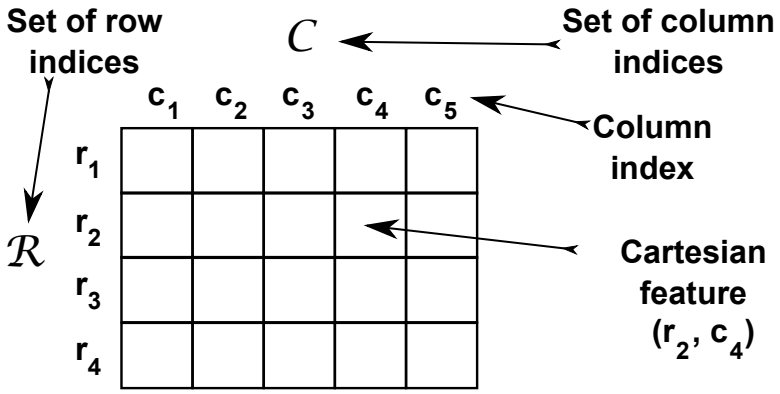

Figure 3.4: An example grid consisting of $4 \times 5 = 20$ elements referred to as Cartesian features (Figure from Publication VI).

3.6 Cartesian cost

In this section, we consider a novel cost type (Publication VI) for linear predictive models that has to be paid in settings where features are associated with the Cartesian product of extractor sets. We assume that there are two disjoint extractor sets, say the row set and column set as illustrated in Figure 3.4. The idea can also be generalized to more than two sets but this is not required in our practical consideration. It requires one extractor from both sets to extract a single feature value and each feature is associated with a different pair of feature extractors. These features are referred here to as Cartesian features.

Definition 2 *The Cartesian product refers to a mathematical operation that creates a product set from two or multiple sets. The Cartesian product* $\mathcal{R} \times \mathcal{C}$ *for two sets* $\mathcal{R}$ *and* $\mathcal{C}$ *is given as*

$$\mathcal{R} \times \mathcal{C} = \{(r, c) | r \in \mathcal{R}, c \in \mathcal{C}\},$$

where (r, c) *is an ordered pair.*

The terminology related to Cartesian cost is explained in the example in Figure 3.4. The grid plots two separate sources, where the elements of the set $\mathcal{R}$ corresponding rows and the elements of set $\mathcal{C}$ corresponding columns. Moreover, the elements of the sets $\mathcal{R}$ and $\mathcal{C}$ are referred to as row indices and column indices, respectively. The ordered pairs (r_i, c_j), where $i = 1 \ldots 4$ and $j = 1 \ldots 5$ in the grid we refer to as Cartesian features. Thus, the number of Cartesian features is $|\mathcal{R}||\mathcal{C}|$, resulting in $4 \times 5 = 20$ features in the example.

This setting gives rise to the Cartesian feature selection problem, whose search space consists of tuples (P, Q) such that $P \subseteq \mathcal{R}$, $Q \subseteq \mathcal{C}$ and $|P|+|Q| \leq k$ that satisfy budget k. Alternatively, a budget constraint could be given

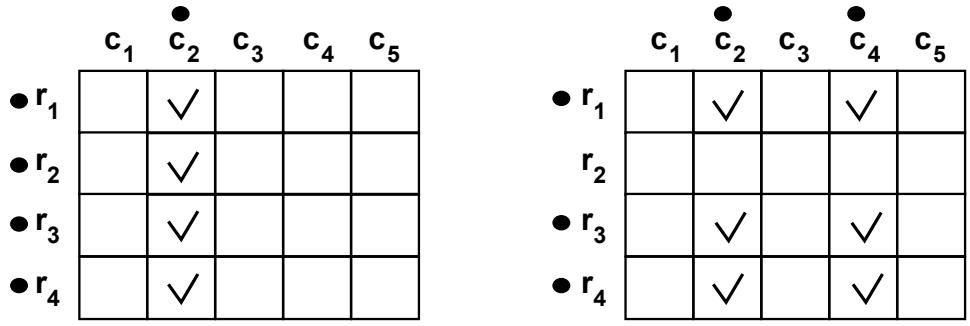

Figure 3.5: Two Cartesian cost profiles.

separately for both of the sources, that is $|P| \leq k_1$ and $|Q| \leq k_2$. The problem is to find such a tuple, whose corresponding feature set determined by $P \times Q \subseteq \mathcal{R} \times \mathcal{C}$ maximizes the prediction performance. The Cartesian cost for the predictor is defined as $|P| + |Q|$, thus assuming equal costs for all the indices.

Example 3 *Let us consider the problem of identifying the concentration of different metal ions in water. Our goal is to minimize the number of different modulator types and the number of different dilution ratios, that both are associated with different costs. Thus, the aim is to find, subject to pre-given budget constraints, a subset of both modulators and dilution ratios to allow predicting the concentration as accurately as possible.*

More generally, Cartesian feature selection applies to pairwise learning problems, where joint features for pairs of objects are often constructed from the tensor product of their feature vectors (see e.g. Pahikkala et al. (2013, 2015)).

Figure 3.5 shows two Cartesian cost profiles as an example. On the left, all the row indices are selected and one column index, thus giving a set of four features as a result of Cartesian product $\{r_1, r_2, r_3, r_4\} \times \{c_2\} = \{(r_1, c_2), (r_2, c_2), (r_3, c_2), (r_4, c_2)\}$. Similarly, for the profile on the right we get Cartesian product $\{r_1, r_3, r_4\} \times \{c_2, c_4\} = \{(r_1, c_2), (r_3, c_2), (r_4, c_2), (r_1, c_4), (r_3, c_4), (r_4, c_4)\}$. The costs of both profiles are $|4| + |1| = |3| + |2| = 5$. While the profiles have equal costs, the number of features in the final predictors are different. In most cases larger number of features results in better prediction performance thus favoring the latter cost profile.

3.7 Related costs

In addition to prediction costs, costs related to obtaining training data have also been considered in related literature. Here we give a brief overview of such cost types, while the topic is left out of scope for this thesis.

The more representative training data are available, the better the model in general can be induced. However, training data does not come for free, because training inputs typically consist of a number of feature values that may be expensive to acquire. We refer to such costs as *training input costs*. Sometimes the outputs are missing as well, and need to be acquired with a cost. We refer to such costs as *training output costs*. We refer to input and output costs together as *training data costs*. In many applications there is no need to retrain or adjust the final model once it has been built. In such settings training costs can be considered as one-time costs that are marginal compared to prediction costs, which have to be paid every time the model is used to predict. Therefore, our focus in this thesis is only on the prediction costs.

Instance completion active feature-value acquisition focuses on the setting where whole feature vectors are acquired with a cost during learning. Saar-Tsechansky and Provost (2004) proposes an algorithm that finds most informative training samples based on the variance in probability estimates. Zheng and Padmanabhan (2002) try to estimate which inputs and how many should be acquired to optimize the accuracy of the model. Active feature acquisition tries to select individual feature values for training inputs such that most improve the predictive accuracy of trained model (Saar-Tsechansky et al., 2009; Melville et al., 2005). Lizotte et al. (2003) consider a setting related to active feature-value acquisition but the cost of selected feature values is restricted by a given budget while active feature acquisition algorithms use some other stopping criteria such as accuracy.

Sometimes outputs rather than feature values of inputs are missing and it is costly to acquire them. *Active learning* (Cohn et al., 1994) is a technique which focuses on solving this kind of problems. Learning algorithm can ask outputs for the examples but it has to pay a price for each of them. As an example application, we may consider a pattern recognition problem where animals are identified from photos. For a human being labeling the data, it takes time to identify the object from the picture and then correctly identify the species of the animal. Often in tasks, where automatic labeling is not possible but rather human interaction is needed, labeling can be very expensive. In these cases active learning offers a way to find those examples whose labeling is the most important to obtain.

Chapter 4

Novel cost-sensitive feature selection algorithms

In this chapter we present an overview of the feature selection problem, and the most important conventional approaches for solving it (filter, wrapper and embedded). We present several novel greedy search-based methods that are designed to solve the cost-sensitive feature selection problems introduced in Chapter 3. In addition, we provide lasso solutions that are employed as a baseline in our experiments. Moreover, we discuss about the findings of our experiments, analyzing the weaknesses and strengths of the algorithms. Next, we introduce feature selection algorithms for the Cartesian greedy forward selection. Finally, we briefly consider the problem of cost-sensitive feature selection for learning-to-rank problems.

4.1 Introduction to feature selection

Feature selection (see John et al. (1994); Blum and Langley (1997); Guyon and Elisseeff (2003)) also known as feature subset selection is a process that aims at selecting relevant features from the pool of candidate features, thus discarding irrelevant or redundant features. Reducing the feature space for a given problem is seen to be important for a number of reasons. First, it may prevent *over-fitting* (see e.g. Reunanen (2003)) that is a common problem when the model is too complex containing too many features compared to the size of the training data. In general, overly complicated models tend to fit the training data very accurately but fail to generalize to new unseen data. Second, the large number of features means that the model is more difficult to interpret for human experts. Finally, due to the costs associated with the features, the model should be kept as compact as possible when it comes to the number of features in order to minimize the prediction costs, which is our main focus in this thesis. Feature selection methods are commonly

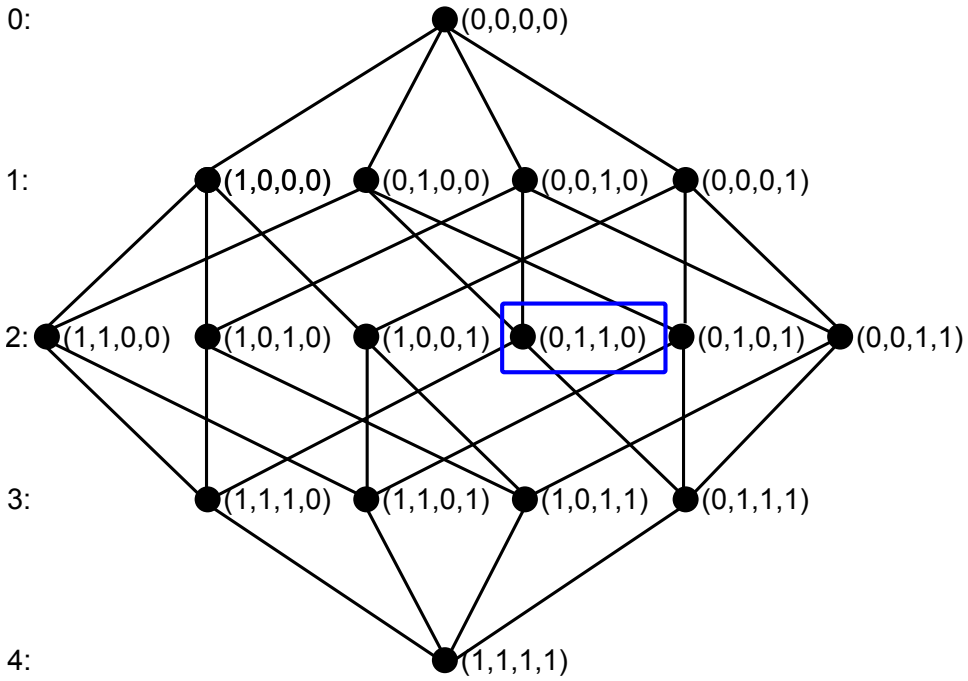

Figure 4.1: An example of feature selection lattice.

divided into three categories (Saeys et al., 2007): filter, wrapper (Kohavi and John, 1997) and embedded methods.

Formally, in a conventional feature selection task, we are given a set of available features Z from which to identify the subset of features $S \subseteq Z$. Let d be the number of available features in the set Z and $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ be the feature vector of an example. Each of the subset of the features can now be encoded into binary vector $\mathbf{f}$ of size d (we call it as a feature boolean vector) so that $f_i = 1$ if i:th feature is selected and $f_i = 0$ otherwise. The search space of the feature selection problem can be represented in the form of a lattice, where subsets of the set Z are marked as nodes and there is a link between the nodes if nodes are immediate subsets of each other. Figure 4.1 shows an example search lattice for four features, where the best subset in terms of given evaluation measure is indicated as a rectangle meaning that the node in question gives the best prediction performance over all the subsets. Each node at the same level in the lattice contains an equal number of features, and the number of different combinations of subsets in level i is $\binom{d}{i}$. Therefore the total number of nodes (or subsets) in the lattice is:

$$\sum_{i=0}^{d} \binom{d}{i} = 2^d.$$

The best subset can be found by applying an exhaustive search that travels through all the nodes in the lattice. For example in Figure 4.1, the number of features is four ($d = 4$) and thus there are only $2^4 = 16$ subsets in the superset. In this case, it is convenient to go through all the nodes and determine which one results in the best performance. However, listing all the subsets is an infeasible task already for very modest values of d because the number of subsets increases exponentially with respect to d. The search problem of subsets is said to be NP-complete (Ullman, 1975), meaning that there is no known polynomial time algorithm for solving the feature selection problem in general. Branch and bound (Narendra and Fukunaga, 1977) is an optimal algorithm if a given evaluation measure J is *monotonic*, meaning that $J(S_1) \geq J(S_2)$ for any two subsets S_1 and S_2 and $S_1 \subseteq S_2$. Still, the worst case runtime of branch and bound is also exponential.

As pointed out in Chapter 3, features do not usually come for free but there is a cost associated with them, such as measuring cost or acquisition cost. We assign a cost to every feature in the set Z and thus define a feature cost vector $\mathbf{c}$ so that c_i corresponds to the cost of i:th feature. Whereas in conventional feature selection problems one aims at optimizing just predictive performance of the model, cost-sensitive feature selection heuristics are used to search such solutions where not just the predictive performance is maximized but also the feature costs are minimized. In cost-sensitive problems there is often given a budget constraint, meaning that the sum of the costs of selected features needs to be below a given budget. That is, $\mathbf{f} \cdot \mathbf{c} < k$, where k is the given budget limit. In this thesis, we make the simplifying assumption that the costs are equal for every feature ($c_1 = c_2 = \ldots = c_d$). In practice, the assumption of equal costs means that the budget constraint corresponds to the number of features allowed in the learned model.

4.1.1 Filter methods

In the filter approach, the selection is done as a pre-processing step before learning, typically by computing univariate statistics such as information gain, mutual information or χ^2 on feature-by-feature basis. The main advantages of the approach are efficiency and ease of implementation. The main disadvantages are the inability to take account of interactions between the features, remove redundant features, or take into account properties of the learning algorithm which is subsequently trained on the features.

More advanced multivariate filter methods have also been developed (see e.g. Hira and Gillies (2015)). Correlation-based feature selection methods assume that features in a relevant subset correlate with the class label, but are uncorrelated with each other (Hall, 2000), whereas information gain methods determine how common a feature is in a class compared to other classes.

4.1.2 Wrapper methods

In the wrapper model (Kohavi and John, 1997), features are selected through interaction with a learner training method. Here, the power set of features is searched over. A new predictor is trained during each search step using the corresponding feature subset and an estimate of its predictive performance is used to measure the quality of the subset. Wrapper techniques consist of three major components:

(i) Base learning algorithm around which the feature selection algorithm is wrapped

(ii) Search strategy over the power set of features

(iii) Heuristic for assessing the goodness of the feature subsets.

Wrapper methods have been proposed as a way to overcome the limitations of the filter approach. On the other hand, the biggest disadvantage of the wrapper methods is considered to be their computationally inefficiency.

Many different approaches are used as the base learning algorithm that works as a black-box method in wrappers. One of the most popular categories is a wide variety of different kinds of neural networks (see e.g. Hornik et al. (1989)) that are easy to use though they are computationally not that efficient or easy to interpret. Common choices include also random forests (Breiman, 2001), k-nearest neighbors (Zhang and Zhou, 2005), and linear regression.

The wrapper type of feature selection methods require a search heuristic when traversing through possible feature subsets. The most commonly used search heuristic is greedy forward selection (see e.g. Zhang (2009)) that adds one feature at a time to the set of selected features, but never removes any features from the set. Greedy backward elimination starts with all features in the selected feature set and then drops off the worst feature at each step. Genetic algorithms (Holland, 1987) provide more sophisticated approaches for searching for the best subset of features.

In addition to the search strategy, wrapper methods require an approach for assessing how good the feature subsets under consideration are. This is done by estimating the prediction performance of a model trained on the considered features. Measuring the prediction performance on the training set is known to be unreliable because of overfitting. If there are lots of data available, one can divide data in training, validation and test sets, where the test set is used to assess the test error. In many practical problems, data is not available or producing it can be expensive. Therefore, statistical re-sampling techniques are often used in model validation, such as *cross-validation* or *the bootstrap* (Kohavi, 1995).

Cross-validation is an approach where data is divided into P parts about equal size folds. Each fold in turn is used as validation set and the rest of the folds form a training set. The P-fold cross-validation estimate for the prediction error is an average of these P errors. Leave-one-out (LOO) cross-validation is a special case where each example in turn is left out of the training set and used for testing. When using cross-validation error as a selection criterion, one cannot use the same cross-validation estimate to evaluate the prediction performance of the learned model. This effect has been called in the literature *selection bias*. Using a separate test set, or so-called *nested cross-validation*, where inner cross-validation is used to select features and outer for performance evaluation, are the standard approaches for addressing this problem (Varma and Simon, 2006). We follow these approaches in all our experiments.

4.1.3 Embedded methods

Embedded methods refer to such feature selection approaches that are learning algorithm specific (Lal et al., 2006). Lasso (see Section 2.4.2) is among the most popular embedded methods. Due to the incorporation of the l_1 norm regularization in its objective function, lasso tends to set some of the coefficients down to zero. Thus the method not only shrinks the coefficients like ridge regression but also carries out feature selection at the same time.

Many variants of lasso are proposed in order to improve some of its weaknesses and restrictions. Zou and Hastie (2005) introduced a method called *elastic net* that combines lasso and ridge regression in order to create sparsity and shrink the coefficients of the linear models. Elastic net encourages a grouping effect that leads to a model where highly correlating features are either all selected or discarded. Elastic net outperforms lasso particularly in the settings that contain a lot of features compared to the number of examples. *Group lasso* (Simon et al., 2013; Meier et al., 2008) is applicable in such settings where a feature grouping is known in advance. Sometimes the groups are overlapping, a setting for which the the so-called *overlap group lasso* (Jacob et al., 2009) may be applied.

4.1.4 Related methods

Dimensionality reduction methods (Roweis and Saul, 2000), such as principal component analysis (Tipping and Bishop, 1999), singular value decomposition (Rokhlin et al., 2009) and Sammon filtering (Kim et al., 2009) are related to feature selection because they transform the original feature space to a lower-dimensional one. However, when making predictions one still needs the original features to transform the new data to the reduced representation. Therefore, using dimensional reduction methods does not

lower the cost of the prediction because all the original features are required anyway.

4.2 Feature selection for multiple tasks

In this section we introduce algorithms for performing feature selection under separate, joint and group cost constraints that were introduced in Sections 3.3, 3.4, and 3.5. Further, we assume that multi-class (see e.g. Section 2.5.1) and multi-label (see e.g. Section 2.5.2) problems are transformed in a pre-processing step into multiple single-task problems using the one-vs-all method. The distributions of non-zero elements in the coefficient matrix $\mathbf{W}$, whose rows are indexed by the tasks, and columns by the features, are called *sparsity patterns* (see Figure 3.2 and Publication V).

4.2.1 Separate feature selection

Separate feature selection is a process where feature selection is performed separately for all the tasks so that the individual training processes do not share any knowledge between each other. Therefore, all the tasks can be solved separately using conventional methods created for single-task problems such as greedy forward selection for least-squares or lasso. In this thesis, we place our main emphasis on greedy forward type solutions, though lasso is considered as a baseline method in many of our experiments.

Greedy solution

First, we show how one can solve tasks separately thus creating sparsity Type I (see Section 3.3) in the coefficient matrix $\mathbf{W}$. This can be achieved by using Algorithm 1 separately for each task. The algorithm implements the heuristic called *greedy forward selection*. The algorithm corresponds to a wrapper type feature selection heuristic that starts from the empty set of features and adds features one by one into the set of selected features until the budget constraint is reached.

Algorithm 1 greedy forward selection

1: $\mathcal{S} \leftarrow \emptyset$ ▷ The set of selected features.
2: **while** $|\mathcal{S}| < k$ **do**
3: $\quad b := \operatorname{argmin}_{r \in \{1,\ldots,d\} \setminus \mathcal{S}} \left\{ \mathcal{L}(\mathbf{X}_{:,\mathcal{S} \cup \{r\}}, \mathbf{y}) \right\}$ ▷ Find the best new feature.
4: $\quad \mathcal{S} \leftarrow \mathcal{S} \cup \{b\}$
5: $\mathbf{w} \leftarrow \mathcal{A}(\mathbf{X}_{:,\mathcal{S}}, \mathbf{y})$ ▷ Update coefficients.

Let $\mathcal{S}$ be the set of selected features, initialized as an empty set. The size of $\mathcal{S}$ will never decrease because the greedy forward search heuristic

always adds a new feature into the set, but never removes selected features. Moreover, let k denote the budget limit. By $\mathcal{L}(\cdot)$ we denote a procedure that computes the cross-validation error for a model trained using the data matrix and output vector given as arguments. Finally, let $\mathcal{A}(\cdot)$ be the procedure that trains a final predictor using the selected features. It is assumed, that the same learning algorithm is used both when computing the cross-validation error, and when learning the final predictor. The **while**-loop is repeated until budget constraint is fulfilled. In every iteration, the algorithm goes through every feature, that has not yet been added into the set of selected features, and calculates the cross-validation error. The feature whose addition leads to the lowest error is selected to be included in the set of selected features S.

The greedy forward selection algorithm requires a lot of computational resources because the model has to be trained many times in order to estimate the predictive performance during the cross-validation. Thus a naive implementation will not scale to large problems. However, for ridge regression this can be done much faster. The greedy-RLS algorithm (Pahikkala et al., 2010, 2012), combines a number of matrix algebra shortcuts such as the Woodbury matrix identity for speeding up cross-validation and feature addition, leading to a linear time algorithm. Further, Okser et al. (2013) introduce a parallel version of greedy-RLS for selecting the most predictive features from large datasets.

Lasso solution

Alternatively, type I sparsity can be induced by using the lasso approach (see Section 2.4.2) that encourages sparse solutions by shrinking some of the coefficients of the linear models down to zero depending on the adjustable regularization parameter. Similarly as for the greedy method, separate feature selection problems can be solved by using the conventional lasso method for every single-task problem separately, and then collecting the coefficients into one sparse matrix.

There is no closed form solution to the lasso optimization problem, but several efficient algorithms have been proposed to solve this objective. Wu (1998) presented a cyclical coordinate descent algorithm as a solution to this problem. The more effective algorithm that can be used to calculate the entire regularization path faster than the earlier method was presented by Friedman et al. (2007).

Previously Okser et al. (2014) have compared lasso and greedy-RLS for solving single task feature selection problems on genome wide association studies. They showed that for small feature set sizes greedy-RLS produced more accurate models, while for larger feature sets the effect disappeared.

4.2.2 Joint feature selection

Next, we consider a heuristic called *joint feature selection* where the aim is to select a common set of features for a group of tasks simultaneously. We introduce a feature selection algorithm that outputs models whose nonzero coefficients are distributed according to sparsity pattern II, thus minimizing the joint cost.

If the tasks are closely related, the joint feature selection may in some cases even improve the prediction performance. However, it is not straightforward to formally define what is meant by "related" (Dembczyński et al., 2012). For example, previously the empirical findings in (Baxter, 1995; Thrun, 1996) have validated the advantages of sharing knowledge between the tasks while solving the problems. This is in the literature referred to as transfer learning. Ben-David and Schuller (2003) consider a notation of relatedness between learning tasks, thus attempting to present a formal framework for the relatedness of the tasks. However, if the tasks are too dissimilar, transfer learning may lower the prediction performance.

Joint greedy forward selection

Joint greedy forward selection presented in Algorithm 2 is an extension to the above-defined greedy forward selection. The algorithm selects a set of features that is shared among all tasks. The only difference is in *line* 3 where we calculate aggregated cross-validation error over all tasks and select the feature whose addition minimizes it. The feature is then added into the selected features set S and this is repeated until budget limit k is reached.

The idea of greedy joint feature selection for multi-label data was first introduced in Publication II, where it was compared to separate greedy selection based baseline methods. In Publication III, the approach was extended to genuine multi-task learning problems, where different tasks may have also different training sets. Publication IV extends the results of Publication II by introducing a highly efficient linear time embedded training method for the multi-label greedy-RLS algorithm, for performing joint greedy selection. Further, a detailed experimental comparison to the multi-task lasso method is provided.

Multi-task lasso

Multi-task lasso is an extension of single-task lasso, where the l_1 regularization is extended to a multi-task setting (Turlach et al., 2005). The objective for multi-task lasso can be given as follows:

$$\text{argmin}_{\mathbf{W}} \sum_{t=1}^{e} \|\mathbf{y}^t - \mathbf{X}^t {\mathbf{W}_t}^{\mathrm{T}}\|_2^2 + \lambda \sum_{j=1}^{d} \|\mathbf{W}_{:,j}\|_\infty. \tag{4.1}$$

Algorithm 2 joint greedy selection

1: $S \leftarrow \emptyset$ ▷ The set of selected features.
2: **while** $|S| < k$ **do**
3: $b := \text{argmin}_{r \in \{1,\ldots,d\} \setminus \mathcal{S}} \left\{ \sum_t \mathcal{L}(\mathbf{X}^t_{:,\mathcal{S}\cup\{r\}}, \mathbf{y}^t) \right\}$ ▷ Find the best new feature.
4: $\mathcal{S} \leftarrow \mathcal{S} \cup \{b\}$
5: $\mathbf{W}_t \leftarrow \mathcal{A}(\mathbf{X}^t_{:,\mathcal{S}}, \mathbf{y}^t), t = 1, \ldots, e$ ▷ Update coefficients.

A variety of optimization methods have been proposed for minimizing the objective function (4.1). Among the most efficient is the blockwise coordinate descent method of Liu et al. (2009), that was also used in our experiments in Publication IV.

The regularizer in Equation 4.1 is referred to as combined norm $l_{1,\infty}$. Alternatively, the $l_{1,2}$ norm has also been used (see e.g Obozinski et al. (2006)). More generally, any combined norm $l_{1,q}$, where the choice of q defines the amount of coupling effect among the tasks, may be used. The larger q is, the more the tasks are tied to together (see Vogt and Roth (2012)).

Experimental findings

In Publication II, we compared joint greedy forward selection (Method 3 in Figure 4.2) to two greedy forward selection baseline methods. Method 1 corresponds to separate feature selection, where the budget is divided evenly between the tasks, and then features are selected independently for each task. Method 2 selects features in the same way as Method 1, but after selection all the models are re-trained on the union of features selected for each task. In Figure 4.2, we present experimental comparison of the three methods on the Scene multi-label data set (see Publication II). We plot the budget size against the area under ROC curve (AUC), averaged over the tasks and computed on a separate test set.

Clearly, in Figure 4.2 the joint feature selection (Method 3) outperforms the separate feature selection (Method 1), and also slightly outperforms Method 2. Similar results were seen on the other data sets, where joint feature selection also outperformed separate one, while differences between Method 2 and joint feature selection were not seen to be as substantial. The results indicate that when subject to joint costs, one should enforce a common set of features in order to obtain cost-wise predictors. With the same cost one can have more accurate predictors, or conversely, equally accurate predictors for lower cost.

A similar trend was also observed in Publication III, where we extended

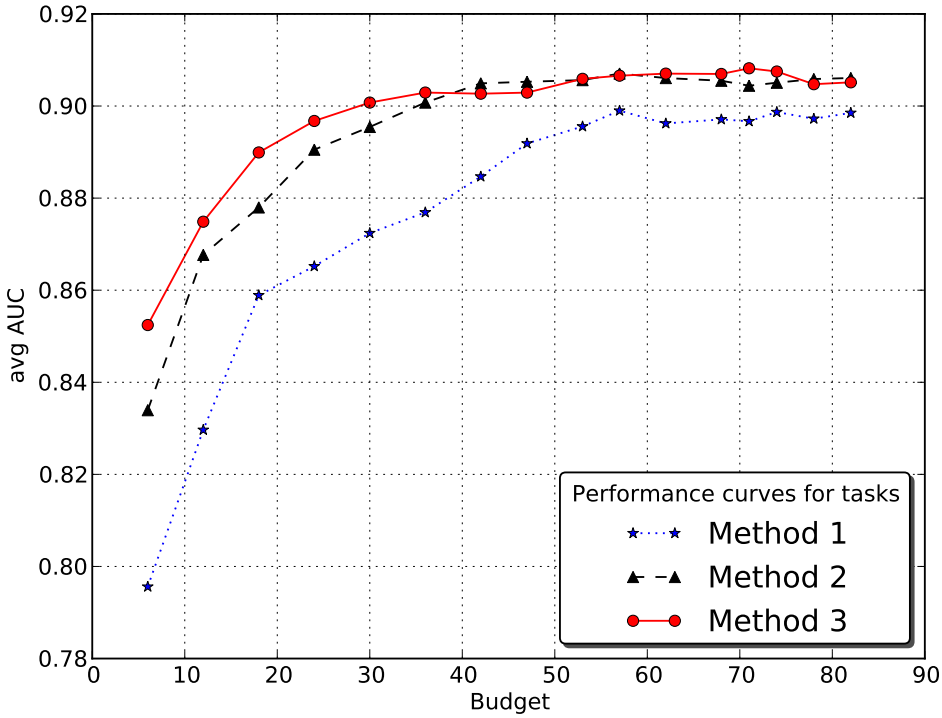

Figure 4.2: Performance curves on the Scene data (Figure from Publication II).

the joint greedy selection algorithm to the more general multi-task setting. In this setting the training data may differ for each of the tasks, whereas the aforementioned multi-label problems share the training inputs. We evaluated the performance of both joint and separate greedy methods on two real-world datasets. Joint greedy selection outperformed both separate selection methods clearly over all budget values.

In Publication IV we compared the multi-label greedy RLS approach to the multi-task lasso, that is one of the most popular methods to induce sparse multi-task models via $l_{1,\infty}$-regularization. Our method outperformed lasso on the data sets over the whole budget range, and it clearly outperformed lasso particularly for small budget sizes.

Figure 4.3 (taken from Publication IV) plots performance curves on the Scene data in terms of the Hamming loss (above) and the macro averaged AUC (below) comparing multi-task lasso and multi-label greedy RLS algorithms. First, one can see that the performance curves do not change much after a small portion of the total budget is used. This indicates that the Scene data set includes several redundant features that do not improve the performance, that is, feature selection results in cost savings by means

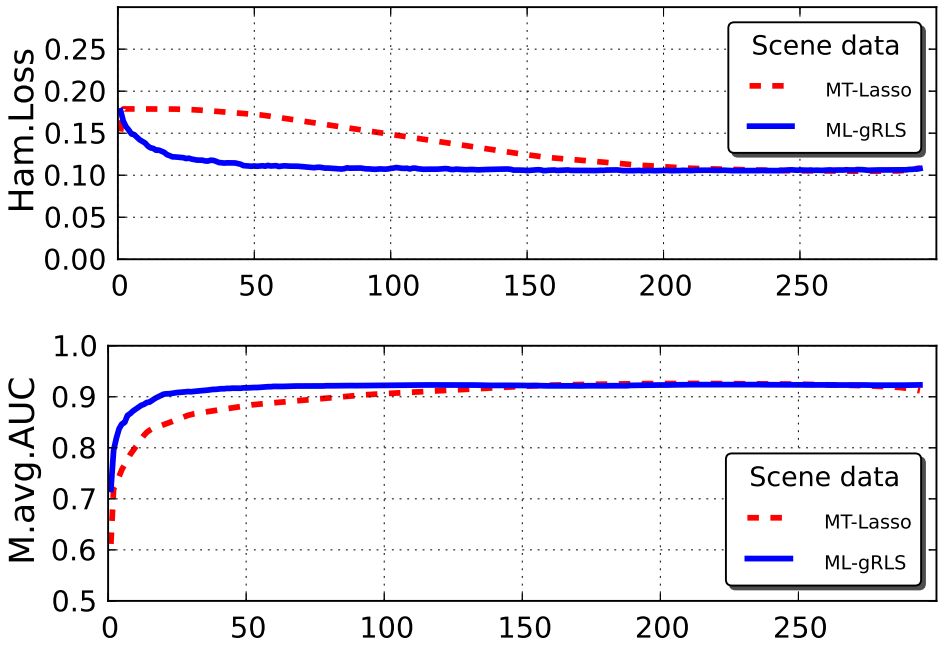

Figure 4.3: Performance curves on the Scene data over feature budgets (number of features) with respect to Hamming Loss and Macro-averaged AUC (Figure from Publication IV).

of fewer features in the predictive model. Secondly, the greedy algorithm reaches the maximum performance earlier than the lasso algorithm, which means that the greedy algorithm outperforms the lasso significantly with the small budget values. Our results with the other datasets considered further confirmed our hypothesis that one should favor greedy instead of lasso particularly with small budgets.

We encountered a few practical problems when working with the lasso approach. First, in order to find a sparse model corresponding to budget constraint k (and the whole path from 1,2, $\ldots k-1$), one has to define the correct grid of regularization parameters λ_i. For single-task problems, there exists the LARS-algorithm (Efron et al., 2004) that allows solving 2.14 for all values $\lambda \in [0, \infty]$ that result in different number of non-zero coefficients. However, for multi-task problems, we are not aware of similar efficient algorithm. Warm start optimization methods can be used to slightly speed up regularization grid search (Liu et al., 2009), but there is no known path solution to exactly find the sparse models corresponding to a given budget. Compared to lasso, the greedy algorithm has the advantage that it provides the models corresponding to all the given budget sizes up to the given budget constraint.

4.2.3 Group feature selection

Group feature selection defines a heuristic where features are selected as a whole group instead of individual features based on predefined division of features into groups. In this thesis the groups are defined from the cost perspective (see Section 3.5). Feature groups can be selected for each task either separately or jointly.

Analogously to the consideration in Section 4.2.1, feature groups can be selected separately for each task resulting in Type III sparsity. The costs are associated with groups of features as described in Section 3.5. Separate group selection analogously to the separate selection (Algorithm 1) given in Section 4.2.1 is shown in Algorithm 3. The main difference to Algorithm 1 is that Algorithm 3 keeps track of the indices of the selected groups (F).

Algorithm 3 greedy group separate feature selection

1: $G_i \subset \{1, \ldots, d\}, i = 1, \ldots, J$ ▷ Init the disjoint feature groups
2: $F \leftarrow \emptyset$ ▷ The set of selected feature groups.
3: $S \leftarrow \emptyset$ ▷ The set of selected features.
4: **while** $|F| < k$ **do**
5: $b := \text{argmin}_{r \in \{1,\ldots,J\} \setminus F} \{\mathcal{L}(\mathbf{X}_{:,\mathcal{S} \cup G_r}, \mathbf{y})\}$ ▷ Find the best feature group.
6: $F \leftarrow F \cup \{b\}$
7: $\mathcal{S} \leftarrow \mathcal{S} \cup G_b$
8: $\mathbf{W} \leftarrow \mathcal{A}(\mathbf{X}_{:,\mathcal{S}}, \mathbf{y})$ ▷ Update coefficients.

There are many l_1-norm solutions that encourage Type III sparsity group-wisely. Yuan and Lin (2006) propose a group lasso, group LARS and group Non-negative Garrote that provide a model where some blocks of coefficients are exactly zero. Kim et al. (2006) extended group lasso for generalized linear models and it works for general loss functions. Further extensions for lasso is presented in (Meier et al., 2008) which proposes group lasso for logistic regression.

Joint group feature selection is done analogously to joint feature selection. This will lead to Type IV sparsity (Figure 3.2) in Section 3.5. Algorithm 4 presents a greedy forward method for selecting features group-wisely jointly for several tasks. It is an adaptation of Algorithm 2 thus extending it to grouped features. There is no known implementation for lasso that induces sparsity jointly in group-level for several tasks simultaneously.

4.3 Cartesian feature selection

The Cartesian feature selection problem, that is based on a real-world chemistry application, is considered in Publication VI. Instead of selecting directly

Algorithm 4 greedy group joint feature selection

1: $G_i \subset \{1, \ldots, d\}, i = 1, \ldots, J$ ▷ Init the disjoint feature groups
2: $F \leftarrow \emptyset$ ▷ The set of selected feature groups.
3: $S \leftarrow \emptyset$ ▷ The set of selected features.
4: **while** $|F| < k$ **do**
5: $b := \text{argmin}_{r \in \{1,\ldots,J\} \setminus F} \left\{ \sum_t \mathcal{L}(\mathbf{X}^t_{:,\mathcal{S} \cup G_r}, \mathbf{y}^t) \right\}$ ▷ Best new feature group.
6: $F \leftarrow F \cup \{b\}$
7: $\mathcal{S} \leftarrow \mathcal{S} \cup G_b$
8: $\mathbf{W}_t \leftarrow \mathcal{A}(\mathbf{X}^t_{:,\mathcal{S}}, \mathbf{y}^t), t = 1, \ldots, e$ ▷ Update coefficients.

features, the key idea is to select feature indices from two distinct index sets; the final features selected correspond to the Cartesian product of these sets. The Cartesian cost is presented in detail in Chapter 3.6.

Algorithm 5 presents the Cartesian greedy forward selection method proposed. Variables $P \subseteq \mathcal{R}$ and $Q \subseteq \mathcal{C}$ denote the selected indices for sources 1 and 2, respectively. The first feature is selected from the set of Cartesian product, $R \times C$, based on the lowest cross-validation error. Function J returns the cross-validation error, taking as input the features that are formed as a result of Cartesian product of candidate indices. After that, the algorithm selects remaining feature indices either from set $\mathcal{R} \setminus P$ or $\mathcal{C} \setminus Q$. On each iteration, the index which gives the lowest cross-validation error will be selected and added into selected feature set. Selection continues until given budget k limit is reached.

The functionality of the Cartesian feature selection heuristic is described with an example in Figure 4.4. The elements in the matrix present candidate Cartesian features determined by the Cartesian product of two index sets, R and C. At the beginning, the algorithm is initialized by searching the feature that returns the lowest cross-validation error. In order to define the first feature, one has to pay a cost equal to two because a feature index has to be selected from both index sets at the beginning. While the following steps always increase the cost by one, the number of new features varies depending on which indices have already been selected. This can be seen in step 3 (bottom left in the Figure 4.4), where selecting index c_3 results in selecting two features at the cost of one. Equivalently, in the bottom right of the figure, selecting index c_4 increases the cost budget by one, but the number of the features increases by two. In general, the set of selected features can be determined as a result of the Cartesian product of subsets P and Q as $\{r_1, r_3\} \times \{c_1, c_3, c_4\}$. Adding a new index to set Q increases the number of features by the factor $|P|$, and vice versa.

In Publication VI, we apply Cartesian feature selection to a novel chem-

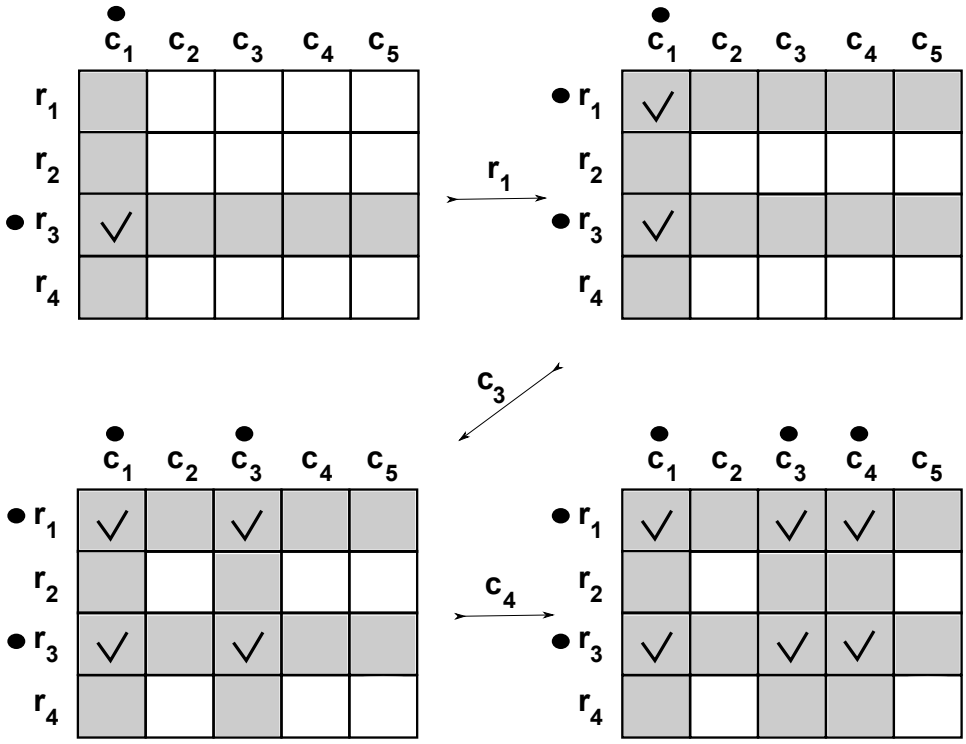

Figure 4.4: Top left: The algorithm is initialized by selecting first the feature providing the lowest cross-validation error, in this case (r_3,c_1). Top right: one row index is selected, now altogether two features have been selected. Bottom left: one column is selected, now four features have been selected e.g. the size of the set is increased by 2 for the price of a single index addition. Bottom right: one additional column is selected, now six features have been selected (Figure from Publication VI).

istry application of identifying metal ion concentration in drinking water. This is seen as an important research problem, because contaminated water is a significant health problem in developing countries. The proposed system combines a number of modulators and liquid sample dilutions, and a machine learning algorithm for assessing metal ion concentrations. The problem was formulated as a Cartesian feature selection problem because the features correspond to modulator-dilution combinations, both of which are associated with costs. The goal was to minimize both the number of modulators and dilutions while maintaining high prediction performance, in order to find a low-cost solution. The findings in the Publication VI showed that our Cartesian feature selection method was able to find relevant features without reducing much the predictive performance of the model even with low budgets.

Algorithm 5 Cartesian greedy forward selection

$\mathcal{R} \neq \emptyset$ ▷ Index set 1
$\mathcal{C} \neq \emptyset$ ▷ Index set 2
$(a, b) \leftarrow \text{argmin}_{(r,c)\in\mathcal{R}\times\mathcal{C}} J((r, c))$ ▷ Initialize first index pair
$P \leftarrow \{a\}$ ▷ Set of selected indices from $\mathcal{R}$
$Q \leftarrow \{b\}$ ▷ Set of selected indices from $\mathcal{C}$
while $|P| + |Q| < k$ **do**
    $a \leftarrow \text{argmin}_{r\in\mathcal{R}\setminus P} J((P \cup \{r\}) \times Q)$
    $b \leftarrow \text{argmin}_{c\in\mathcal{C}\setminus Q} J(P \times (Q \cup \{c\}))$
    if $J((P \cup \{a\}) \times Q) < J(P \times (Q \cup \{b\}))$ **then**
        $P \leftarrow P \cup \{a\}$
    else
        $Q \leftarrow Q \cup \{b\}$
Return P, Q

Figure 4.5 plots heat maps for metals ions (Cu and Ni) to be identified, where the axes correspond to the modulator and dilution budgets, and the color to cross-validated predictive performance. The bottom left corner indicates the start of the feature selection process, whereas the top right corner corresponds to the end of the process where all features have been selected. The heat map is based on averaged results from a large number of cross-validation experiments with different randomized splits for the data. On different rounds of cross-validation, the selection process can take a different route. Still, due to the greedy search criterion most of the budget space is not covered.

At first, the more features are included into model, the better accuracy is reached (the red color refers to higher accuracy values, whereas the blue color refers the lower accuracy values). The best performance is reached before the upper right corner where all the features are included. Thus, in these experiments including too many features is seen to actually reduce the predictive performance.

4.4 Feature selection for learning to rank

Traditionally, the most popular machine learning tasks have been classification and regression in the area of supervised learning, clustering in unsupervised learning, but lately *ranking* has gained popularity particularly in the area of applications of information retrieval (Frakes and Baeza-Yates, 1992) and recommender systems (Ricci et al., 2010).

Learning to rank problems contain a training set of lists of items that are ordered, and the learning algorithm that fits the ranking model to the

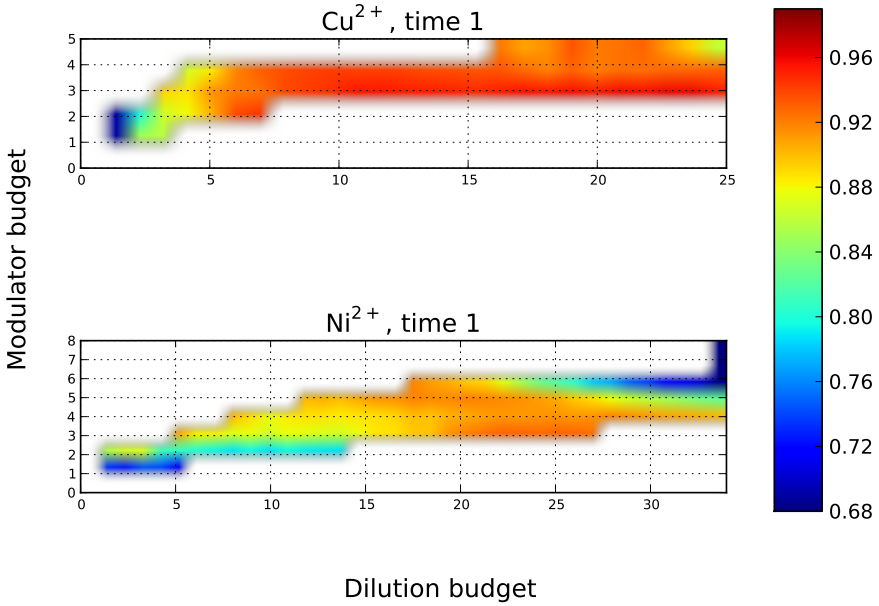

Figure 4.5: Heat maps for Cartesian feature selection on two metal ion data sets (Figure adapted from Publication VI).

training data in a learning process. The goal is to find such a ranking model that is able to rank accurately every unseen item list. In document retrieval training data consist of a set of documents and queries and a relevance degree that defines the quality of each match. Query-document pairs are represented typically in the form of feature vectors. Due to real-time response demands faced by search engines, one should limit the number of computed features to be as small as possible. Therefore feature selection is seen as an important step in order to create accurate and cost-sensitive ranking models, where consumed time or computation is the most natural cost.

Many learning to rank algorithms have been developed, such as RankSVM (Joachims, 2002) or RankRLS (Pahikkala et al., 2007) but embedded or wrapper type feature selection methods for learning to rank are not that common. We present a greedy forward selection method to the task of learning to rank by extending RankRLS in Publication I to the greedy RankRLS method. In the cross-validation step we use the so-called leave-query-out heuristic that guarantees that data points in a same query are used either in the training or test test but not in both at the same time.

Chapter 5

Conclusions

This section summarizes the contributions given in the six original publications included in the thesis. We also consider extensions of the algorithms and methods for more general settings in future work.

Our answer to the first research question is explorative, in the sense that we formulated a number of cost profiles that we encountered in our applications. Namely, we provide mathematical characterization for the separate, joint, separate group and joint group costs as well as Cartesian costs, based on budget constraints on the model coefficients.

For each cost profile we have developed a greedy feature selection algorithm. The proposed methods were shown to outperform both simpler greedy baselines, as well as lasso type methods.

The methods developed in this thesis have already shown their impact in a well established scientific challenge, which recently compared cost effective feature selective methods. Specifically, the greedy joint feature selection implemented in multi-label greedy RLS algorithm (Publication IV), provided the best results in the 2014 Broad-DREAM Gene Essentiality Prediction Sub-Challenge 3 (Gönen et al., 2017).

5.1 Overview of the research articles

Publication I presents an efficient algorithm for learning to rank with regularized least-squares, focusing on the task of document retrieval. We showed in experimental evaluation that the algorithm induced sparse ranking models. The considered cost is the number of features that a web search engine needs to compute in order to rank documents for web queries.

We extended our considerations to such classification and regression problems (Publications II, III and IV) where multiple targets have to be predicted under a given budget. We showed in experimental evaluation for multi-task and multi-label problems that in order to induce cost-effective

models one should favor joint feature selection. We presented an efficient joint greedy forward selection algorithm for both multi-label and multi-task problems, which outperformed lasso in most of our experiments, especially with low budget values.

In Publication V we provided a unified overview of sparsity patterns, which characterize the solutions to the separate, joint, and group costs. We further showed in detail how two types of cost settings, extractor and extracting cost, map to feature costs of predictors.

Publication VI introduced a novel feature selection problem referred to as Cartesian feature selection. The features are formed as a Cartesian product of two separate sets of indices that each are associated with cost. We presented a greedy forward selection heuristic that cost-efficiently selects feature indices from two sources one by one until the Cartesian budget constraint is satisfied. A problem of predicting metal ion concentration from drinking water promotes the need of Cartesian feature selection in real life applications. The experimental results show that the Cartesian feature selection method finds relevant features efficiently with respect to cost and better than simpler greedy and lasso type methods.

5.2 Future work

So far we have assumed equal costs for features. However, the presented algorithms can be extended to the variable cost setting. This is a natural direction to future studies since often in real life applications features are associated with variable costs. For example, in health applications one may use sensors that monitor for instance users blood pressure, location, heart rate or movement patterns to make conclusions about the state of one's health. All the sensors, however, come with a variable cost ranging from few cents to several euros.

References

Adriaans, P. and Zantinge, D. (1997). *Data Mining*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Baxter, J. (1995). Learning internal representations. In *Proceedings of the Eighth Annual Conference on Computational Learning Theory (COLT '95)*, pages 311–320, New York, NY, USA. ACM.

Ben-David, S. and Schuller, R. (2003). Exploiting task relatedness for multiple task learning. In *Proceedings of the 16th Annual Conference of Learning Theory (COLT '03)*, pages 825–830.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Blum, A. L. and Langley, P. (1997). Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97(1-2):245–271.

Boutell, M. R., Luo, J., Shen, X., and Brown, C. M. (2004). Learning multi-label scene classification. *Pattern Recognition*, 37(9):1757 – 1771.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

Caruana, R. (1997). Multitask learning. *Machine Learning*, 28(1):41–75.

Chen, S. S., Donoho, D. L., and Saunders, M. A. (2001). Atomic decomposition by basis pursuit. *SIAM Review*, 43(1):129–159.

Clare, A. and King, R. D. (2001). Knowledge discovery in multi-label phenotype data. In *Proceedings of the 5th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD '01)*, pages 42–53.

Cohn, D., Atlas, L., and Ladner, R. (1994). Improving generalization with active learning. *Machine Learning*, 15(2):201–221.

Dembczyński, K., Waegeman, W., Cheng, W., and Hüllermeier, E. (2012). On label dependence and loss minimization in multi-label classification. *Machine Learning*, 88(1):5–45.

Efron, B., Hastie, T., Johnstone, I., and Tibshirani, R. (2004). Least angle regression. *Annals of Statistics*, 32:407–499.

Elisseeff, A. and Weston, J. (2001). A kernel method for multi-labelled classification. In *Proceedings of the Advances in Neural Information Processing Systems 14 (NIPS '01)*, pages 681–687. MIT Press.

Frakes, W. B. and Baeza-Yates, R., editors (1992). *Information Retrieval: Data Structures and Algorithms*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Friedman, J., Hastie, T., Hofling, H., and Tibshirani, R. (2007). Pathwise coordinate optimization. *Annals of Applied Statistics*, 1(2):302–332.

Gönen, M., Weir, B. A., Cowley, G. S., Guan, Y., Jaiswal, A., Karasuyama, M., Uzunangelov, V., Vazquez, F., Wang, T., Tsherniak, A., Howell, S., Marbach, D., Hoff, B., Norman, T. C., Airola, A., Bivol, A., Bunte, K., Carlin, D., Chopra, S., Deran, A., Ellrott, K., Gopalacharyulu, P., Graim, K., Kaski, S., Khan, S. A., Newton, Y., Ng, S., Pahikkala, T., Paull, E., Sokolov, A., Tang, H., Tang, J., Wennerberg, K., Xie, Y., Zhan, X., Zhu, F., Aittokallio, T., Boehm, J. S., Mamitsuka, H., Root, D. A., Stuart, J. M., Xiao, G., Stolovitzky, G., Hahn, W. C., and Margolin, A. A. (2017). A community-based challenge for building predictive models of gene essentialities over a large-scale functional screen of cancer cell lines. *Cell Systems*. In Press.

Guyon, I. and Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182.

Hall, M. A. (2000). Correlation-based feature selection for discrete and numeric class machine learning. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML '00)*, pages 359–366, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc.

Hira, Z. M. and Gillies, D. F. (2015). A review of feature selection and feature extraction methods applied on microarray data. In *Advances in Bioinformatics*, volume 2015.

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

Holland, J. H. (1987). Genetic algorithms and classifier systems: Foundations and future directions. In *Proceedings of the 2nd International Conference on Genetic Algorithms (ICGA '87), Cambridge, MA, USA, July 1987*, pages 82–89.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359 – 366.

Huang, J., Zhang, T., and Metaxas, D. (2009). Learning with structured sparsity. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML '09)*, pages 417–424, New York, NY, USA. ACM.

Jacob, L., Obozinski, G., and Vert, J.-P. (2009). Group lasso with overlap and graph lasso. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML '09)*, pages 433–440, New York, NY, USA. ACM.

Joachims, T. (2002). Optimizing search engines using clickthrough data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '02)*, pages 133–142, New York, NY, USA. ACM.

John, G. H., Kohavi, R., and Pfleger, K. (1994). Irrelevant features and the subset selection problem. In *Proceedings of the International Conference on Machine Learning (ICML '94)*, pages 121–129. Morgan Kaufmann.

Kim, Y., Kim, J., and Kim, Y. (2006). Blockwise sparse regression. *Statistica Sinica*, 16:375–390.

Kim, Y.-H., Lee, K. H., and Yoon, Y. (2009). Visualizing the search process of particle swarm optimization. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCO '09)*, pages 49–56, New York, NY, USA. ACM.

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2 (IJCAI'95)*, pages 1137–1143.

Kohavi, R. and John, G. H. (1997). Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324.

Krishnapuram, B., Yu, S., and Rao, R. B. (2011). *Cost-Sensitive Machine Learning*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition.

Lal, T. N., Chapelle, O., Weston, J., and Elisseeff, A. (2006). *Embedded Methods*, pages 137–165. Springer Berlin Heidelberg, Berlin, Heidelberg.

Liu, H., Palatucci, M., and J.Zhang (2009). Blockwise coordinate descent procedures for the multi-task lasso, with applications to neural semantic basis discovery. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML '09)*, pages 649–656.

Lizotte, D. J., Madani, O., and Greiner, R. (2003). Budgeted learning of naïve-Bayes classifiers. In *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence (UAI '03)*, pages 378–385.

Madjarov, G., Kocev, D., Gjorgjevikj, D., and Deroski, S. (2012). An extensive experimental comparison of methods for multi-label learning. *Pattern Recognition*, 45(9):3084–3104.

Mäkelä, J. (2009). *Electromagnetic signatures of lightning near the HF frequency band*. PhD thesis, University of Helsinki, Finnish Meteorological Institute Contributions 80.

Meier, L., Geer, S. V. D., Bühlmann, P., and Zrich, E. T. H. (2008). The group lasso for logistic regression. *Journal of the Royal Statistical Society, Series B*, 70(1):53–71.

Melville, P., Provost, F. J., and Mooney, R. J. (2005). An expected utility approach to active feature-value acquisition. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM '05), 27-30 November 2005, Houston, Texas, USA*, pages 745–748.

Min, F., He, H., Qian, Y., and Zhu, W. (2011). Test-cost-sensitive attribute reduction. *Information Sciences*, 181(22):4928–4942.

Narendra, P. and Fukunaga, K. (1977). A branch and bound algorithm for feature subset selection. *IEEE Transactions on Computers*, 26(9):917–922.

Nowak, S., Lukashevich, H., Dunker, P., and Rüger, S. (2010). Performance measures for multilabel evaluation: A case study in the area of image classification. In *Proceedings of the International Conference on Multimedia Information Retrieval (MIR '10)*, pages 35–44.

Obozinski, G., Taskar, B., and Jordan, M. (2006). Multi-task feature selection. In *Proceedings of ICML'06 workshop of structural Knowledge Transfer for Machine Learning*.

Okser, S., Airola, A., Salakoski, T., Aittokallio, T., and Pahikkala, T. (2013). Parallel feature selection for regularized least-squares. In Manninen, P. and Öster, P., editors, *Proceedings of the 11th international conference on Applied Parallel and Scientific Computing (PARA '12), Revised Selected Papers*, volume 7782 of *Lecture Notes in Computer Science*, pages 280–294. Springer.

Okser, S., Pahikkala, T., Airola, A., Salakoski, T., Ripatti, S., and Aittokallio, T. (2014). Regularized machine learning in the genetic prediction of complex traits. *PLOS Genetics*, 10(11):e1004754.

Pahikkala, T., Airola, A., Pietil, S., Shakyawar, S., Szwajda, A., Tang, J., and Aittokallio, T. (2015). Toward more realistic drug-target interaction predictions. *Briefings in Bioinformatics*, 16(2):325–337.

Pahikkala, T., Airola, A., and Salakoski, T. (2010). Speeding up greedy forward selection for regularized least-squares. In *Proceedings of ninth International Conference on Machine Learning and Applications (ICMLA '10)*, pages 325–330. IEEE.

Pahikkala, T., Airola, A., Stock, M., Baets, B. D., and Waegeman, W. (2013). Efficient regularized least-squares algorithms for conditional ranking on relational data. *Machine Learning*, 93(2-3):321–356.

Pahikkala, T., Okser, S., Airola, A., Salakoski, T., and Aittokallio, T. (2012). Wrapper-based selection of genetic features in genome-wide association studies through fast matrix operations. *Algorithms for Molecular Biology*, 7(1):11.

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

Reunanen, J. (2003). Overfitting in making comparisons between variable selection methods. *Journal of Machine Learning Research*, 3:1371–1382.

Ricci, F., Rokach, L., Shapira, B., and Kantor, P. B. (2010). *Recommender Systems Handbook*. Springer-Verlag New York, Inc., New York, NY, USA, 1st edition.

Rifkin, R. M. and Klautau, A. (2004). In defense of one-vs-all classification. *Journal of Machine Learning Research*, 5:101–141.

Rokhlin, V., Szlam, A., and Tygert, M. (2009). A randomized algorithm for principal component analysis. *SIAM Journal on Matrix Analysis and Applications*, 31(3):1100–1124.

Roweis, S. T. and Saul, L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290:2323–2326.

Saar-Tsechansky, M., Melville, P., and Provost, F. (2009). Active feature-value acquisition. *Management Science*, 55(4):664–684.

Saar-Tsechansky, M. and Provost, F. (2004). Active sampling for class probability estimation and ranking. *Machine Learning*, 54(2):153–178.

Saeys, Y., Inza, I. n., and Larrañaga, P. (2007). A review of feature selection techniques in bioinformatics. *Bioinformatics*, 23(19):2507–2517.

Shalev-Shwartz, S., Srebro, N., and Zhang, T. (2010). Trading accuracy for sparsity in optimization problems with sparsity constraints. *SIAM Journal on Optimization*, 20(6):2807–2832.

Simon, N., Friedman, J., Hastie, T., and Tibshirani, R. (2013). A sparse-group lasso. *Journal of Computational and Graphical Statistics*, 22(2):231–245.

Thrun, S. (1996). Is learning the n-th thing any easier than learning the first? In *Proceedings in the Advances in Neural Information Processing Systems (NIPS '96)*, pages 640–646. The MIT Press.

Tibshirani, R. (1994). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288.

Tipping, M. E. and Bishop, C. M. (1999). Probabilistic principal component analysis. *Journal of the Royal Statistical Society, Series B*, 61:611–622.

Trohidis, K., Tsoumakas, G., Kalliris, G., and Vlahavas, I. (2011). Multi-label classification of music by emotion. *EURASIP Journal on Audio, Speech, and Music Processing*, 2011(1):4.

Tsoumakas, G., Katakis, I., and Vlahavas, I. (2010). Mining multi-label data. In Maimon, O. and Rokach, L., editors, *Data Mining and Knowledge Discovery Handbook*, pages 667–685. Springer.

Tsoumakas, G. and Vlahavas, I. (2007). Random k-labelsets: An ensemble method for multilabel classification. In *Proceedings of the 18th European Conference on Machine Learning (ECML '07)*, pages 406–417.

Turlach, B. A., Venables, W. N., and Wright, S. J. (2005). Simultaneous variable selection. *Technometrics*, 47(3):349–363.

Turney, P. D. (2000). Types of cost in inductive concept learning. In Dietterich, T., Margineantu, D., Provost, F., and Turney, P. D., editors, *Proceedings of the ICML 2000 Workshop on Cost-Sensitive Learning*.

Ullman, J. D. (1975). Np-complete scheduling problems. *Journal of Computer and System Sciences*, 10(3):384–393.

Varma, S. and Simon, R. (2006). Bias in error estimation when using cross-validation for model selection. *BMC bioinformatics*, 7(1):91+.

Vogt, J. E. and Roth, V. (2012). A complete analysis of the group lasso. In *Proceedings of the 27th International Conference on Machine Learning (ICML '12)*.

Witten, I. H., Frank, E., and Hall, M. A. (2011). *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition.

Wu, W. J. (1998). Penalized regressions: The bridge versus the lasso. *Journal of Computational and Graphical Statistics*.

Xu, Z., Weinberger, K., and Chapelle, O. (2012). The greedy miser: Learning under test-time budgets. In Langford, J. and Pineau, J., editors, *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 1175–1182, New York, NY, USA. ACM.

Yang, J. and Honavar, V. G. (1998). Feature subset selection using a genetic algorithm. *IEEE Intelligent Systems*, 13(2):44–49.

Yuan, M. and Lin, Y. (2006). Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67.

Zhang, M.-L. and Zhou, Z.-H. (2005). A k-Nearest Neighbor Based Algorithm for Multi-label Classification. In *Proceedings of the IEEE International Conference on Granular Computing (GrC '05)*, pages 718–721 Vol. 2.

Zhang, T. (2009). On the consistency of feature selection using greedy least squares regression. *Journal of Machine Learning Research*, 10:555–568.

Zheng, Z. and Padmanabhan, B. (2002). On active learning for data acquisition. In *Proceedings of the International Conference on Data Mining (ICDM '02)*, pages 562–.

Zou, H. and Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society, Series B*, 67:301–320.

Publication Reprints

Publication I

Greedy RankRLS: a Linear Time Algorithm for Learning Sparse Ranking Models

Tapio Pahikkala, Antti Airola, Pekka Naula, Tapio Salakoski. In: Evgeniy Gabrilovich, Alexander J. Smola, Naftali Tishby (Eds.), SIGIR 2010 Workshop on Feature Generation and Selection for Information Retrieval, 11–18, ACM, 2010.

Greedy RankRLS: a Linear Time Algorithm for Learning Sparse Ranking Models

Tapio Pahikkala
University of Turku and Turku Centre for Computer Science (TUCS), Turku, Finland
firstname.lastname@utu.fi

Antti Airola
University of Turku and Turku Centre for Computer Science (TUCS), Turku, Finland
firstname.lastname@utu.fi

Pekka Naula
University of Turku
Turku, Finland
firstname.lastname@utu.fi

Tapio Salakoski
University of Turku and Turku Centre for Computer Science (TUCS), Turku, Finland
firstname.lastname@utu.fi

ABSTRACT

Ranking is a central problem in information retrieval. Much work has been done in the recent years to automate the development of ranking models by means of supervised machine learning. Feature selection aims to provide sparse models which are computationally efficient to evaluate, and have good ranking performance. We propose integrating the feature selection as part of the training process for the ranking algorithm, by means of a wrapper method which performs greedy forward selection, using leave-query-out cross-validation estimate of performance as the selection criterion. We introduce a linear time training algorithm we call greedy RankRLS, which combines the aforementioned procedure, together with regularized risk minimization based on pairwise least-squares loss. The training complexity of the method is $O(kmn)$, where k is the number of features to be selected, m is the number of training examples, and n is the overall number of features. Experiments on the LETOR benchmark data set demonstrate that the approach works in practice.

Keywords

feature selection, learning to rank, ranking, RankRLS, regularized least-squares, variable selection

1. INTRODUCTION

Learning to rank for information retrieval has been a topic of intense research during the recent years. The possible benefits of automatically inducing ranking models from data, compared to purely handcrafted systems, include reduced manual labor, increased ranking performance, and adaptivity to individual user preferences. A number of supervised machine learning methods have been proposed, and successfully applied for this task. These include both pairwise approaches such as RankSVM [8], RankNet [2], and RankRLS [16, 17], as well approaches which optimize multivariate loss functions defined over queries, also known as the listwise approach [3, 4, 25].

In this article we consider the task of feature selection for learning to rank, specifically concentrating on the task of document retrieval. The task is to recognize an informative subset of the features, such that a machine learning method trained on the subset achieves good ranking performance on unseen future data. Perhaps the most fundamental advantage of this approach is that it leads to sparse models, as only a limited subset of features is used for prediction. Since applications such as web-search engines are typically constrained by strict real-time response demands, being able to restrict the number of features that need to be calculated can be quite useful. Further, feature selection can also provide feedback on the quality of different features, which can be very useful when developing and testing new ones.

Feature selection methods are typically divided into three categories [6]. In the filter approach the features are selected as a pre-processing step before applying a learning algorithm, wrapper methods select features through interaction with a learning algorithm, and embedded methods perform the selection as part of the learning process itself. Feature selection for ranking is not as of yet a well studied area, but a number of approaches have been introduced during the past few years. Geng et al. [5] proposed a filter method for selecting such features which produce good rankings, while at the same time aiming to minimize the redundancy in the set of selected features. The work was further improved upon by Yu et al. [24]. Metzler [12] and Pan et al. [19] considered feature selection for ranking with Markov random fields and boosted trees, respectively.

Since the final goal of the feature selection process is to produce a sparse ranking model with good performance, we argue that the most natural selection criterion is the so-called wrapper approach [6, 10, 11]. We select such features, which result in maximal ranking performance for the supervised learning method that we are actually using to learn our model. A standard approach for estimating the generalization performance of a model trained on a subset of features

SIGIR Workshop on Feature Generation and Selection for Information Retrieval 2010 Geneva, Switzerland

is to use cross-validation, as proposed by [10] for wrapper based feature selection. More specifically, we propose using leave-query-out (LQO) cross-validation. This allows one to make maximal use of training data, and guarantees that data points related to the same query are never split between the training and test folds. Further, to make the search over the power set of features feasible, we propose to use greedy forward selection, where on each iteration the feature whose addition yields best cross-validation performance is selected.

In this article we propose the greedy RankRLS algorithm, that is able to efficiently perform the aforementioned selection procedure. The algorithm is equivalent to a wrapper method that for each tested feature set, and each round of cross-validation would train the RankRLS method [16, 17], which minimizes the pairwise regularized least-squares loss. It can also be considered as an embedded method, since the proposed training algorithm for greedy RankRLS training is far more efficient than the straightforward approach of using RankRLS as a black-box method within the selection process would be. Previously, RankRLS has been shown to produce good ranking results in document retrieval, and in general achieve ranking performance similar to that of RankSVM [16, 17].

To achieve computational efficiency for the wrapper method, we combine the training algorithm with matrix algebra based shortcuts. These are made possible by the fact that RankRLS has a closed form solution, which can be fully expressed in terms of matrix operations. Firstly, the developed shortcuts allow efficient update of the solution when new features are added, without having to recompute the solution from scratch. We have previously proposed similar shortcuts for the greedy RLS algorithm [13,14], which allows one to train sparse regressors and classifiers in linear time. Second, based on the results in [1,15] we derive a formula for the exact LQO estimate that is more efficient than the one previously proposed in [16], and combine it with the update operation for feature addition. The resulting complexity of greedy RankRLS training is $O(kmn)$, where k is the number of features to be selected, m is the number of training examples, and n is the overall number of features. The memory complexity of the method is $O(mn)$. We are not aware of as efficient greedy forward selection methods with cross-validation based selection criterion for other state-of-the-art learning to rank methods.

2. SETTING

We start by introducing some notation. Let $\mathbb{R}^m$ and $\mathbb{R}^{n\times m}$, where $n, m \in \mathbb{N}$, denote the sets of real valued column vectors and $n \times m$-matrices, respectively. To denote real valued matrices and vectors we use bold capital letters and bold lower case letters, respectively. Moreover, index sets are denoted with calligraphic capital letters. By denoting $\mathbf{M}_i$, $\mathbf{M}_{:,j}$, and $\mathbf{M}_{i,j}$, we refer to the ith row, jth column, and i,jth entry of the matrix $\mathbf{M} \in \mathbb{R}^{n\times m}$, respectively. Similarly, for index sets $\mathcal{R} \subseteq \{1,\ldots,n\}$ and $\mathcal{L} \subseteq \{1,\ldots,m\}$, we denote the submatrices of $\mathbf{M}$ having their rows indexed by $\mathcal{R}$, the columns by $\mathcal{L}$, and the rows by $\mathcal{R}$ and columns by $\mathcal{L}$ as $\mathbf{M}_{\mathcal{R}}$, $\mathbf{M}_{:,\mathcal{L}}$, and $\mathbf{M}_{\mathcal{R},\mathcal{L}}$, respectively. We use an analogous notation also for column vectors, that is, $\mathbf{v}_i$ refers to the ith entry of the vector $\mathbf{v}$.

We assume, that we are given a training set in a form of data matrix $\mathbf{X} \in \mathbb{R}^{n\times m}$ and a label vector $\mathbf{y} \in \mathbb{R}^m$. The rows of the data matrix are indexed by the n features and the columns by the m training examples. Thus, by $\mathbf{X}_{:,i}$ we denote the column vector containing the features of the ith example, sliced from the data matrix and by $\mathbf{y}_i$ we denote its corresponding real value label. Let $\mathcal{I} = \{1\ldots m\}$ denote the index set for the training set. The index set is divided into a number of disjoint queries, where $Q = \{\mathcal{Q}^{(1)},\ldots,\mathcal{Q}^{(|Q|)}\}$ is the set of queries, and $\mathcal{Q}^{(i)} \subset \mathcal{I}$, $\bigcup_{i=1}^{|Q|} \mathcal{Q}^{(i)} = \mathcal{I}$ and $\mathcal{Q}^{(i)} \cap \mathcal{Q}^{(j)} = \emptyset$, if $i \neq j$. Each example represents a query-document pair. The features are a joint feature representation for the query the example is associated with, and a candidate document, and the label denotes how relevant the document is with respect to the query. The ranking of the documents associated with a query can be obtained by sorting them according to the values of their labels.

In feature selection, the task is to select a subset $\mathcal{S} \subset \{1\ldots n\}$, $|\mathcal{S}| = k$, of the n available features, such that the resulting predictor is sparse, but still produces a good ranking performance on new data. The number of selected features k may be decided in advance, or selected against a validation set, according to application specific criteria. We consider linear predictors of type

$$f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x}_{\mathcal{S}}, \tag{1}$$

where $\mathbf{w}$ is the k-dimensional vector representation of the learned predictor and $\mathbf{x}_{\mathcal{S}}$ can be considered as a mapping of the data point x into k-dimensional feature space. Note that the vector $\mathbf{w}$ only contains entries corresponding to the features indexed by $\mathcal{S}$. The rest of the features of the data points are not used in prediction phase. The computational complexity of making predictions with (1) and the space complexity of the predictor are both $O(k)$, provided that the feature vector representation $\mathbf{x}_{\mathcal{S}}$ for the data point x is given.

The pairwise ranking error for a learned predictor f can be defined as

$$\frac{1}{|Q|}\sum_{\mathcal{Q}^{(i)}\in Q}\frac{1}{N^{(i)}}\sum_{j,k\in\mathcal{Q},\mathbf{y}_j<\mathbf{y}_k} H(f(\mathbf{X}_{:,j}) - f(\mathbf{X}_{:,k})), \tag{2}$$

where H is the Heaviside step function defined as

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

and $N^{(i)}$ is the number of pairs in the i:th query, for which $\mathbf{y}_j < \mathbf{y}_k$ holds true. In this definition, the error is normalized so that each considered query has the same importance, regardless of size. Since (2) is non-convex, successful pairwise approaches for learning to rank typically minimize convex approximations instead.

The RankRLS algorithm [16, 17] is based on regularized risk minimization, where a least-squares based approximation of (2) is minimized, together with a quadratic regularizer. We approximate (2) with the pairwise least-squares loss, where step function H is replaced with the pairwise squared loss

$$l(i,j) = (\mathbf{y}_i - \mathbf{y}_j - f(\mathbf{X}_{:,i}) + f(\mathbf{X}_{:,j}))^2.$$

Compared to the discrete pairwise loss, this loss also enforces the magnitudes of the prediction differences. For the purpose of simplifying the derivation and implementation of the learning algorithm, we modify the normalizers, and also

include tied predictions within the same query in the loss. The linear RankRLS solution is found by solving

$$\underset{\mathbf{w}\in\mathbb{R}^{|\mathcal{S}|}}{\operatorname{argmin}} \left\{ \sum_{\mathcal{Q}\in Q} \frac{1}{2|\mathcal{Q}|} \sum_{i,j\in\mathcal{Q}} l(i,j) + \lambda\|\mathbf{w}\|^2 \right\} \tag{3}$$

where the first term is the empirical risk measuring how well the model determined by $\mathbf{w}$ fits to the training data, and the second term called regularizer penalizes complex models. The regularization parameter $\lambda > 0$ controls the tradeoff between these terms.

Let

$$\mathbf{L}_l = \mathbf{I} - \frac{1}{l}\mathbf{1}\mathbf{1}^{\mathrm{T}}$$

be the $l \times l$-centering matrix with $l \in \mathbb{N}$. The matrix $\mathbf{L}$ is an idempotent matrix and multiplying it with a vector removes the mean of the vector entries from all elements of the vector. Moreover, the following equality can be shown

$$\frac{1}{2l}\sum_{i,j=1}^{l}(c_i - c_j) = \mathbf{c}^{\mathrm{T}}\mathbf{L}_l\mathbf{c}\,,$$

where c_i are the entries of any vector $\mathbf{c}$. Without loss of generality, we can assume that the training data is ordered according to the queries, so that first come the examples belonging to the first query, next to the second, etc. Now, let us consider the following quasi-diagonal matrix:

$$\mathbf{L} = \begin{pmatrix} \mathbf{L}_{l_1} & & \\ & \ddots & \\ & & \mathbf{L}_{l_{|Q|}} \end{pmatrix},$$

where $l_i = |\mathcal{Q}_i|$ for $i \in \{1,\ldots,|Q|\}$. The matrix $\mathbf{L}$ is again symmetric and idempotent, and can be interpreted as a query-wise centering matrix, that removes the mean from the prediction errors for each query [18]. It is also a normalized version of the Laplacian matrix encoding the structure of the preference graph induced by the queries, which has been used in previous derivations of RankRLS [16, 17].

Now, (3) can be re-written in matrix notation as

$$\underset{\mathbf{w}\in\mathbb{R}^{|\mathcal{S}|}}{\operatorname{argmin}} \left\{ ((\mathbf{w}^{\mathrm{T}}\mathbf{X}_{\mathcal{S}})^{\mathrm{T}} - \mathbf{y})^{\mathrm{T}}\mathbf{L}((\mathbf{w}^{\mathrm{T}}\mathbf{X}_{\mathcal{S}})^{\mathrm{T}} - \mathbf{y}) + \lambda\mathbf{w}^{\mathrm{T}}\mathbf{w} \right\}. \tag{4}$$

Analogously to the results in [16, 17], a minimizer of (4) is

$$\mathbf{w} = (\mathbf{X}_{\mathcal{S}}\mathbf{L}(\mathbf{X}_{\mathcal{S}})^{\mathrm{T}} + \lambda\mathbf{I})^{-1}\mathbf{X}_{\mathcal{S}}\mathbf{L}\mathbf{y}.$$

Due to the symmetry and idempotence of $\mathbf{L}$, this can be re-written as

$$\mathbf{w} = (\widehat{\mathbf{X}}_{\mathcal{S}}(\widehat{\mathbf{X}}_{\mathcal{S}})^{\mathrm{T}} + \lambda\mathbf{I})^{-1}\widehat{\mathbf{X}}_{\mathcal{S}}\widehat{\mathbf{y}}, \tag{5}$$

where $\widehat{\mathbf{X}}_{\mathcal{S}} = \mathbf{X}_{\mathcal{S}}\mathbf{L}$ and $\widehat{\mathbf{y}} = \mathbf{L}\mathbf{y}$. Using the sparse decomposition of $\mathbf{L}$, the first multiplication can be performed in $O(m|\mathcal{S}|)$, and the second in $O(m)$ time. We note (see e.g. [7]) that equivalently, the RankRLS solution can be obtained from

$$\mathbf{w} = \widehat{\mathbf{X}}_{\mathcal{S}}((\widehat{\mathbf{X}}_{\mathcal{S}})^{\mathrm{T}}\widehat{\mathbf{X}}_{\mathcal{S}} + \lambda\mathbf{I})^{-1}\widehat{\mathbf{y}}. \tag{6}$$

The overall complexity of solving (5) is $O(m|\mathcal{S}|^2 + |\mathcal{S}|^3)$, and that of solving (6) $O(m^2|\mathcal{S}| + m^3)$. We note that these are solutions to the ordinary regularized least squares (RLS) problem. Thus, by the query-wise centering of the data matrix the RankRLS problem can be mapped to that of solving the ordinary RLS problem.

Finally, we consider the issue of cross-validation. As discussed before, we aim to perform LQO cross-validation, where in turn each query is left out of the training set, and used for testing. Let $\mathcal{Q}$ be the index set of a query, and let $\overline{\mathcal{Q}} = \mathcal{I} \setminus \mathcal{Q}$ be the complement of this index set.

Let us consider the centered data matrix from which the rows corresponding to the $\mathcal{Q}$ have been removed, $\widehat{\mathbf{X}}_{\mathcal{S}\overline{\mathcal{Q}}}$. Due to the quasi-diagonal structure of $\mathbf{L}$, the submatrices $\mathbf{L}_{\mathcal{Q}\overline{\mathcal{Q}}}$ have only zero entries. Therefore, we have

$$\mathbf{X}_{\mathcal{S}\overline{\mathcal{Q}}}\mathbf{L}_{\overline{\mathcal{Q}}\overline{\mathcal{Q}}} = \widehat{\mathbf{X}}_{\mathcal{S}\overline{\mathcal{Q}}} - \mathbf{X}_{\mathcal{S}\mathcal{Q}}\mathbf{L}_{\mathcal{Q}\overline{\mathcal{Q}}} = \widehat{\mathbf{X}}_{\mathcal{S}\overline{\mathcal{Q}}}.$$

The significance of this result is that when removing a query from the training set, we can recover the centered representation of the remaining data simply by slicing $\widehat{\mathbf{X}}$. The centering operation does not need to be re-calculated. The feature representation for the test query is also centered, if we recover it from the centered training data matrix. Since using centered data does not affect the relative ordering or relative differences in prediction values, as long as ranking based performance measures are used, this makes no difference.

These results considerably simplify the development of efficient cross-validation methods for RankRLS. As long as folds are defined along query lines, the task of performing cross-validation with RankRLS is identical to that of performing cross-validation with RLS using centered training data. In problem settings where $\mathbf{L}$ does not have quasi-diagonal structure, such as when learning from a single global ranking, such results do not exist, making development of cross-validation shortcuts more challenging.

3. ALGORITHM DESCRIPTION

Here, we present the computational short-cuts that enable the efficient feature subset search strategy for RankRLS with LQO error as a heuristic. First, we recall an approach for computing the hold-out error for the RLS algorithm. By hold-out, we indicate the method that is used to estimate the performance of the learning algorithm by holding a part of the given data set as a test set and training a learner with the rest of the data. Our hold-out formulation assumes that the whole data set is used to train a RLS predictor and the hold-out set is then "unlearned" afterwards. The formulation can then be used, for example, to perform a N-fold CV by holding out a different part of the data set at a time and averaging the results. In this paper, we use it specifically for LQO-CV, that is, each query is held out from the training set at a time, and for a particular query, the corresponding hold-out set consists of all the training examples associated with the query.

Now, let us define

$$\mathbf{G} = ((\widehat{\mathbf{X}}_{\mathcal{S}})^{\mathrm{T}}\widehat{\mathbf{X}}_{\mathcal{S}} + \lambda\mathbf{I})^{-1}$$

and

$$\mathbf{a} = \mathbf{G}\widehat{\mathbf{y}}.$$

The following theorem can be straightforwardly inferred from the results presented by [1, 15].

THEOREM 3.1. *The predictions for the data points indexed by $\mathcal{Q}$ made by a RLS predictor trained using the features indexed by $\mathcal{S}$ and with the whole training set except the examples indexed by $\mathcal{Q}$ can be obtained from*

$$\widehat{\mathbf{y}}_{\mathcal{Q}} - (\mathbf{G}_{\mathcal{Q}\mathcal{Q}})^{-1}\mathbf{a}_{\mathcal{Q}}.$$

According to the above theorem, the result of LQO-CV with squared error as a performance measure can be obtained from

$$\sum_{\mathcal{Q} \in Q} (\mathbf{p}_{\mathcal{Q}})^{\mathrm{T}} \mathbf{p}_{\mathcal{Q}}, \tag{7}$$

where

$$\mathbf{p}_{\mathcal{Q}} = (\mathbf{G}_{\mathcal{Q}\mathcal{Q}})^{-1} \mathbf{a}_{\mathcal{Q}}.$$

It is quite straightforward to show that the vector of hold-out errors is centered query-wise, that is, $\mathbf{p} = \mathbf{L}\mathbf{p}$, because we use $\widehat{\mathbf{X}}$ and $\widehat{\mathbf{y}}$ in place of $\mathbf{X}$ and $\mathbf{y}$. Therefore, the sum of squared hold-out errors (7) is, in fact, the sum of squared query-wise centered hold-out errors. As shown earlier, this corresponds to the sum of pairwise squared losses, calculated for each query separately.

Algorithm 1: Greedy RankRLS

Input: $\widehat{\mathbf{X}} \in \mathbb{R}^{n \times m}$, $\widehat{\mathbf{y}} \in \mathbb{R}^m$, k, λ
Output: $\mathcal{S}$, $\mathbf{w}$

$\mathbf{a} \leftarrow \lambda^{-1}\widehat{\mathbf{y}}$;
$\mathbf{C} \leftarrow \lambda^{-1}\widehat{\mathbf{X}}^{\mathrm{T}}$;
$\mathbf{U} \leftarrow \widehat{\mathbf{X}}^{\mathrm{T}}$;
$\mathbf{p} \leftarrow \widehat{\mathbf{y}}$;
$\mathcal{S} \leftarrow \emptyset$;
while $|\mathcal{S}| < k$ **do**
  $e \leftarrow \infty$;
  $b \leftarrow 0$;
  foreach $i \in \{1, \ldots, n\} \setminus \mathcal{S}$ **do**
    $c \leftarrow (1 + \widehat{\mathbf{X}}_i \mathbf{C}_{:,i})^{-1}$;
    $d \leftarrow c\mathbf{C}^{\mathrm{T}}{}_i \widehat{\mathbf{y}}$;
    $e_i \leftarrow 0$;
    foreach $\mathcal{Q} \in Q$ **do**
      $\gamma \leftarrow (-c^{-1} + \mathbf{C}^{\mathrm{T}}{}_{i,\mathcal{Q}} \mathbf{U}_{\mathcal{Q},i})^{-1}$;
      $\tilde{\mathbf{p}}_{\mathcal{Q}} \leftarrow \mathbf{p}_{\mathcal{Q}} - d\mathbf{U}_{\mathcal{Q},i} - \gamma \mathbf{U}_{\mathcal{Q},i}(\mathbf{U}^{\mathrm{T}}{}_{i,\mathcal{Q}}(\mathbf{a}_{\mathcal{Q}} - d\mathbf{C}_{\mathcal{Q},i}))$;
      $e_i \leftarrow e_i + (\tilde{\mathbf{p}}_{\mathcal{Q}})^{\mathrm{T}} \tilde{\mathbf{p}}_{\mathcal{Q}}$;
    if $e_i < e$ **then**
      $e \leftarrow e_i$;
      $b \leftarrow i$;
  $c \leftarrow (1 + \widehat{\mathbf{X}}_b \mathbf{C}_{:,b})^{-1}$;
  $d \leftarrow c\mathbf{C}^{\mathrm{T}}{}_b \widehat{\mathbf{y}}$;
  $\mathbf{t} \leftarrow c\widehat{\mathbf{X}}_b \mathbf{C}$;
  foreach $\mathcal{Q} \in Q$ **do**
    $\gamma \leftarrow (-c^{-1} + \mathbf{C}^{\mathrm{T}}{}_{b,\mathcal{Q}} \mathbf{U}_{\mathcal{Q},b})^{-1}$;
    $\mathbf{p}_{\mathcal{Q}} \leftarrow \mathbf{p}_{\mathcal{Q}} - d\mathbf{U}_{\mathcal{Q},b} - \gamma \mathbf{U}_{\mathcal{Q},b}(\mathbf{U}^{\mathrm{T}}{}_{b,\mathcal{Q}}(\mathbf{a}_{\mathcal{Q}} - d\mathbf{C}_{\mathcal{Q},b}))$;
    $\mathbf{U}_{\mathcal{Q}} \leftarrow \mathbf{U}_{\mathcal{Q}} - \mathbf{U}_{\mathcal{Q},b}\mathbf{t} - \gamma \mathbf{U}_{\mathcal{Q},b}(\mathbf{U}^{\mathrm{T}}{}_{b,\mathcal{Q}}(\mathbf{C}_{\mathcal{Q}} - \mathbf{C}_{\mathcal{Q},b}\mathbf{t}))$;
  $\mathbf{a} \leftarrow \mathbf{a} - d\mathbf{C}_{:,b}$;
  $\mathbf{C} \leftarrow \mathbf{C} - \mathbf{C}_{:,b}\mathbf{t}$;
  $\mathcal{S} \leftarrow \mathcal{S} \cup \{b\}$;
$\mathbf{w} \leftarrow \widehat{\mathbf{X}}_{\mathcal{S}} \mathbf{a}$;

Next, we go through the actual feature selection algorithm whose pseudo code is presented in Algorithm 1. Let us first define the following quasi-diagonal matrix:

$$\mathbf{Q} = \begin{pmatrix} (\mathbf{G}_{\mathcal{Q}_1,\mathcal{Q}_1})^{-1} & & \\ & \ddots & \\ & & (\mathbf{G}_{\mathcal{Q}_{|Q|},\mathcal{Q}_{|Q|}})^{-1} \end{pmatrix}.$$

In order to take advantage of the computational short-cuts, the feature selection algorithm maintains the current set of selected features $\mathcal{S} \subseteq \{1, \ldots, n\}$, the vectors $\mathbf{a}, \mathbf{p} \in \mathbb{R}^m$, and the matrices $\mathbf{C}, \mathbf{U} \in \mathbb{R}^{m \times n}$ whose values are defined as

$$\begin{aligned} \mathbf{a} &= \mathbf{G}\widehat{\mathbf{y}}, \\ \mathbf{C} &= \mathbf{G}\widehat{\mathbf{X}}^{\mathrm{T}}, \\ \mathbf{U} &= \mathbf{Q}\mathbf{G}\widehat{\mathbf{X}}^{\mathrm{T}}, \\ \mathbf{p} &= \mathbf{Q}\mathbf{G}\widehat{\mathbf{y}}. \end{aligned}$$

In the initialization phase of the greedy RankRLS algorithm the set of selected features is empty, and hence the values of $\mathbf{a}$, $\mathbf{C}$, $\mathbf{U}$, and $\mathbf{p}$ are initialized to $\lambda^{-1}\widehat{\mathbf{y}}$, $\lambda^{-1}\widehat{\mathbf{X}}^{\mathrm{T}}$, $\widehat{\mathbf{X}}^{\mathrm{T}}$, and $\widehat{\mathbf{y}}$, respectively. The computational complexity of the initialization phase is dominated by the $O(mn)$ time required for storing the matrices $\mathbf{C}$, $\mathbf{U}$, and $\widehat{\mathbf{X}}$ in memory. Thus, the initialization phase is no more complex than one pass through the training data.

Let us now consider the computation of the LQO performance for the modified feature set $\mathcal{S} \cup \{i\}$, where i is the index of the feature to be added. Recall that the hold-out prediction for the examples that are associated with query $\mathcal{Q}$ can be computed from $\mathbf{p}_{\mathcal{Q}} = (\mathbf{G}_{\mathcal{Q},\mathcal{Q}})^{-1}\mathbf{a}_{\mathcal{Q}}$, where $\mathbf{a} = \mathbf{G}\widehat{\mathbf{y}}$. However, since a new feature is temporarily added into the set of selected features, we must use the matrix

$$\widetilde{\mathbf{G}} = ((\widehat{\mathbf{X}}_{\mathcal{S}})^{\mathrm{T}}\widehat{\mathbf{X}}_{\mathcal{S}} + (\widehat{\mathbf{X}}_i)^{\mathrm{T}}\widehat{\mathbf{X}}_i + \lambda\mathbf{I})^{-1}$$

in place of $\mathbf{G}$. Due to the well-known Sherman-Morrison-Woodbury (SMW) formula, the matrix $\widetilde{\mathbf{G}}$ can be rewritten as

$$\begin{aligned} \widetilde{\mathbf{G}} &= \mathbf{G} - \mathbf{G}(\widehat{\mathbf{X}}_i)^{\mathrm{T}}(1 + \widehat{\mathbf{X}}_i\mathbf{G}(\widehat{\mathbf{X}}_i)^{\mathrm{T}})^{-1}\widehat{\mathbf{X}}_i\mathbf{G} \\ &= \mathbf{G} - c\mathbf{C}_{:,i}\mathbf{C}^{\mathrm{T}}{}_i, \end{aligned}$$

where

$$c = (1 + \widehat{\mathbf{X}}_i\mathbf{C}_{:,i})^{-1}.$$

Accordingly, the updated vector of dual variables $\tilde{\mathbf{a}}$ can be written as

$$\begin{aligned} \tilde{\mathbf{a}} &= \widetilde{\mathbf{G}}\widehat{\mathbf{y}} \\ &= (\mathbf{G} - c\mathbf{C}_{:,i}\mathbf{C}^{\mathrm{T}}{}_i)\widehat{\mathbf{y}} \\ &= \mathbf{a} - d\mathbf{C}_{:,i}, \end{aligned}$$

where

$$d = c\mathbf{C}^{\mathrm{T}}{}_i\widehat{\mathbf{y}}.$$

Now, concerning $(\widetilde{\mathbf{G}}_{\mathcal{Q},\mathcal{Q}})^{-1}$, we have

$$\begin{aligned} (\widetilde{\mathbf{G}}_{\mathcal{Q},\mathcal{Q}})^{-1} &= ((\mathbf{G} - c\mathbf{C}_{:,i}\mathbf{C}^{\mathrm{T}}{}_i)_{\mathcal{Q},\mathcal{Q}})^{-1} \\ &= (\mathbf{G}_{\mathcal{Q},\mathcal{Q}} - c\mathbf{C}_{\mathcal{Q},i}\mathbf{C}^{\mathrm{T}}{}_{i,\mathcal{Q}})^{-1} \\ &= (\mathbf{G}_{\mathcal{Q},\mathcal{Q}})^{-1} \\ &\quad -\gamma(\mathbf{G}_{\mathcal{Q},\mathcal{Q}})^{-1}\mathbf{C}_{\mathcal{Q},i}\mathbf{C}^{\mathrm{T}}{}_{i,\mathcal{Q}}(\mathbf{G}_{\mathcal{Q},\mathcal{Q}})^{-1} \\ &= (\mathbf{G}_{\mathcal{Q},\mathcal{Q}})^{-1} - \gamma\mathbf{U}_{\mathcal{Q},i}\mathbf{U}^{\mathrm{T}}{}_{i,\mathcal{Q}}, \end{aligned}$$

where

$$\gamma = (-c^{-1} + \mathbf{C}^{\mathrm{T}}{}_{i,\mathcal{Q}}\mathbf{U}_{\mathcal{Q},i})^{-1}$$

and the equality between the second and third rows are again due to the SMW formula. Finally, we can compute the hold-out predictions $\tilde{\mathbf{p}}_{\mathcal{Q}}$ for the updated feature set as

$$\begin{aligned} \tilde{\mathbf{p}}_{\mathcal{Q}} &= (\widetilde{\mathbf{G}}_{\mathcal{Q},\mathcal{Q}})^{-1}\tilde{\mathbf{a}}_{\mathcal{Q}} \\ &= (\mathbf{G}_{\mathcal{Q},\mathcal{Q}})^{-1}\tilde{\mathbf{a}}_{\mathcal{Q}} - \gamma\mathbf{U}_{\mathcal{Q},i}(\mathbf{U}^{\mathrm{T}}{}_{i,\mathcal{Q}}\tilde{\mathbf{a}}_{\mathcal{Q}}) \\ &= \mathbf{p}_{\mathcal{Q}} - d\mathbf{U}_{\mathcal{Q},i} - \gamma\mathbf{U}_{\mathcal{Q},i}(\mathbf{U}^{\mathrm{T}}{}_{i,\mathcal{Q}}(\mathbf{a}_{\mathcal{Q}} - d\mathbf{C}_{\mathcal{Q},i})). \end{aligned}$$

The calculation of the last row requires only $O(|\mathcal{Q}|)$ time provided that we have all the required caches available. Since the sizes of the query index sets sum up to m, the overall complexity of LQO-CV for the updated feature set becomes $O(m)$. Further, as the greedy forward selection approach tests each of the order of n unselected features before the best of them is added into the set of selected features, the complexity of the selection step is $O(mn)$.

What is still left in our consideration is the phase in which the caches are updated after an new feature is added into the set of selected features. The vectors $\mathbf{a}$ and $\mathbf{p}$ are updated in the same way as they were temporarily updated in the LQO-CV computations. The update processes of the matrices $\mathbf{U}$ and $\mathbf{C}$ are analogous to those of the vectors $\mathbf{p}$ and $\mathbf{a}$ except that the matrix $\mathbf{C}$ is used in place of the vector $\mathbf{a}$ and the vector

$$\mathbf{t} = c\widehat{\mathbf{X}}_b\mathbf{C},$$

where b is the index of the selected feature, is used in place of the constant d. The computational time required for updating $\mathbf{U}$ and $\mathbf{C}$ is $O(mn)$, that is, updating the caches after the selection step is not more complex than the selection step itself.

Putting everything together, the overall computational complexity of greedy RankRLS is $O(kmn)$, where k is the number of features the algorithm selects until it stops. This is because the algorithm performs k iterations during which it adds one new feature to the set of selected features and each iteration requires $O(mn)$ time as shown above. The space complexity of the algorithm is $O(mn)$ which is dominated by keeping the matrices $\mathbf{C}$, $\mathbf{U}$, and $\widehat{\mathbf{X}}$ in memory.

4. EXPERIMENTS

We perform experiments on the publicly available LETOR benchmark data set (version 4.0) for learning to rank for information retrieval [1] [21]. We run experiments on two data sets, MQ2007 and MQ2008. MQ2007 consists of 69623 examples divided into 1700 queries, and MQ2800 contains 15211 examples divided into 800 queries. In both data sets the examples have the same 46 high-level features.

We follow the experimental setup proposed by the authors of LETOR. All results are averages from 5-fold cross-validation, where on each round 3 folds are used for training, 1 for parameter selection and 1 for testing. We use the exact splits provided in the data sets. Mean Average Precision (MAP) is used when selecting parameters. In addition to average precision, we measure Normalized Discountive Cumulative Gain (NDCG) when calculating test performance. In the results we present MAP, P@10, mean NDCG, and NDCG@10 values.

We compare greedy RankRLS to RankRLS and RankSVM, which are trained on all the features. For greedy RankRLS, we choose via grid search both the number of selected features and value of regularization parameter, such that lead to best MAP performance on the validation fold. For normal RankRLS, which is trained on all the features, only the value of the regularization parameter needs to be tuned. RankRLS and greedy RankRLS are implemented as part of the RLScore open source machine learning framework [2]. The RankSVM results are taken directly from the baselines section of the LETOR distribution website. The experimental setup for the RankSVM runs, as described by LETOR authors, is the same as outlined here, the used implementation was the SVM^{rank} of Joachims[3] [9]. We also plot performance curves as a function of the number of selected features on the validation sets, and examine the feature sets selected on different folds.

Tables 1 and 2 contains the selected features on the MQ2007 and MQ2008 data sets, respectively. Where more than 10 features was selected, we present only the first 10. On two of the folds of MQ2007, the optimal number of features are 11 and 12, on three of the folds almost all of the features are chosen. On MQ2008 relatively few features were chosen on all of the folds, on two of the folds the best validation performance was reached with only one feature. There are differences in the feature sets selected in the different rounds of cross-validation, but one thing remains constant. On both data sets, and on each cross-validation round, the feature selected first is feature number 39, "LMIR.DIR of whole document". The feature is a language model based feature which corresponds to a posteriori estimate of the likelihood of the query given the whole document, where a Dirichlet prior over the documents is used [26]. Based on our results this feature seems to be very useful for ranking, since as it turns out models using only it can in some cases be competitive with models trained on all the features.

In Figures 1, 2, 3, and 4 are the average MAP and mean NDCG performances over the validation folds, plotted for different regularization parameter values. We note that the results are quite unstable, suggesting that reliable selection of the regularization parameter and number of selected features remains a challenging problem. On MQ2007 the performance increases with the number of selected features. This is why in three of the folds the selection strategy used in our study lead to selecting almost all of them. However, on MQ2008 best validation performances are reached with relatively few features, after which the performance decreases. On MQ2007 close to optimal validation results can be reached already with around 15 features. This suggests that perhaps a multi-objective criterion should be used in parameter selection, which in addition to favoring high validation performance would also penalize models that use too many features.

In Tables 3, 4, 5, and 6, are the test results for MQ2007, and in Tables 7, 8, 9, and 10 are the test results for MQ2008. Overall, the results for greedy RankRLS, RankRLS and RankSVM are very close to each other, even on fold-by-fold basis. The results further verify the earlier results in [16,17], which suggest that RankRLS and RankSVM optimization often lead to very similar results. Further, the results show that at least on this data, sparse models learned using greedy forward selection are competitive with models learned using all the features.

5. DISCUSSION AND FUTURE WORK

The greedy RankRLS implementation presented in this paper is computationally feasible when dealing with data sets, such as LETOR, in which the overall number of available features is not very large. However, the situation is different if the data points are represented, for example, as

[1] http://research.microsoft.com/en-us/um/beijing/projects/letor/

[2] http://www.tucs.fi/rlscore

[3] http://www.cs.cornell.edu/People/tj/svm_light/svm_rank.html

Table 1: Selected features on MQ2007.

Model	fold1	fold2	fold3	fold4	fold5
λ	2^8	2^6	2^9	2^8	2^7
k	11	40	46	44	12
selected 1	39	39	39	39	39
selected 2	19	32	27	28	25
selected 3	25	19	23	45	19
selected 4	23	26	19	23	43
selected 5	32	23	13	43	23
selected 6	16	16	18	33	29
selected 7	43	5	42	13	22
selected 8	22	33	33	18	18
selected 9	5	18	16	22	5
selected 10	33	3	5	15	16

Table 2: Selected features on MQ2008

Model	fold1	fold2	fold3	fold4	fold5
λ	2^0	2^{10}	2^3	2^6	2^0
k	1	4	7	4	1
selected 1	39	39	39	39	39
selected 2		23	29	29	
selected 3		37	25	25	
selected 4		32	23	23	
selected 5			46		
selected 6			37		
selected 7			19		

raw text documents, and the words or their composites occurring in the documents form the set of available features. In this case, there is the drawback that $m \times n$-dimensional dense matrices has to be maintained in memory, while the data are stored in a sparse matrix of the same size having only a few nonzero entries. This is, because each document has nonzero values only for a small subset of features. In addition to the feasibility problems with memory, the $O(mn)$ time required per iteration may be too expensive in practise. Fortunately, it is possible to design such variations of greedy RankRLS that are better suited for this type of data.

First, we can avoid storing the dense $m \times n$-matrices by spending more computational resources. This is possible with a variation whose time complexity is $O(k^2mn)$. As an additional modification, we can reduce the time spent in each iteration by selecting the new feature from a random subset of the available features, resulting to a time complexity $O(k^2m\kappa)$, where κ is the size of random subsets. This type of idea is used, for example, for selecting the basis vectors for Gaussian process regressors by [22]. Finally, we can take advantage of the sparsity of the data matrix in reducing the time complexity down to $O(k^2\overline{m}\kappa)$, where $\overline{m}$ is the average number of training examples for which the features have nonzero values, if we use so-called back-fitting variation of our algorithm instead of performing pre-fitting as our current implementation does. For descriptions of the terms back-fitting and pre-fitting, we refer to [23]. The detailed

Table 3: Map results on MQ2007

Fold	GRankRLS	RankRLS	RankSVM
1	0.4859	0.4912	0.4894
2	0.4571	0.4573	0.4573
3	0.4655	0.4655	0.4676
4	0.4423	0.4425	0.4401
5	0.4709	0.4687	0.4680
avg	0.4643	0.4650	0.4645

Table 4: P@10 results on MQ2007

Fold	GRankRLS	RankRLS	RankSVM
1	0.3958	0.3997	0.4036
2	0.3858	0.3855	0.3932
3	0.3684	0.3684	0.3699
4	0.3670	0.3673	0.3652
5	0.3808	0.3811	0.3847
avg	0.3796	0.3804	0.3833

Table 5: MeanNDCG results on MQ2007

Fold	GRankRLS	RankRLS	RankSVM
1	0.5228	0.5281	0.5278
2	0.4840	0.4841	0.4810
3	0.5056	0.5056	0.5042
4	0.4757	0.4754	0.4699
5	0.5033	0.5003	0.5003
avg	0.4983	0.4987	0.4966

Table 6: NDCG@10 results on MQ2007

Fold	GRankRLS	RankRLS	RankSVM
1	0.4735	0.4784	0.4818
2	0.4247	0.4246	0.4266
3	0.4466	0.4466	0.4461
4	0.4221	0.4221	0.4163
5	0.4487	0.4460	0.4485
avg	0.4431	0.4435	0.4439

Table 7: Map results on MQ2008

Fold	GRankRLS	RankRLS	RankSVM
1	0.4311	0.4524	0.4502
2	0.4239	0.4300	0.4213
3	0.4582	0.4542	0.4529
4	0.5283	0.5225	0.5284
5	0.5183	0.5006	0.4950
avg	0.4720	0.4719	0.4696

Table 8: P@10 results on MQ2008

Fold	GRankRLS	RankRLS	RankSVM
1	0.2333	0.2391	0.2423
2	0.2178	0.2217	0.2229
3	0.2363	0.2325	0.2357
4	0.2975	0.2949	0.2981
5	0.2484	0.2503	0.2465
avg	0.2467	0.2477	0.2491

Table 9: MeanNDCG results on MQ2008

Fold	GRankRLS	RankRLS	RankSVM
1	0.4454	0.4633	0.4577
2	0.4186	0.4269	0.4296
3	0.4787	0.4741	0.4686
4	0.5403	0.5407	0.5442
5	0.5369	0.5138	0.5159
avg	0.4840	0.4838	0.4832

Table 10: NDCG@10 results on MQ2008

Fold	GRankRLS	RankRLS	RankSVM
1	0.1920	0.2145	0.2117
2	0.1585	0.1669	0.1738
3	0.2558	0.2489	0.2494
4	0.2940	0.2874	0.2892
5	0.2254	0.2165	0.2155
avg	0.2251	0.2268	0.2279

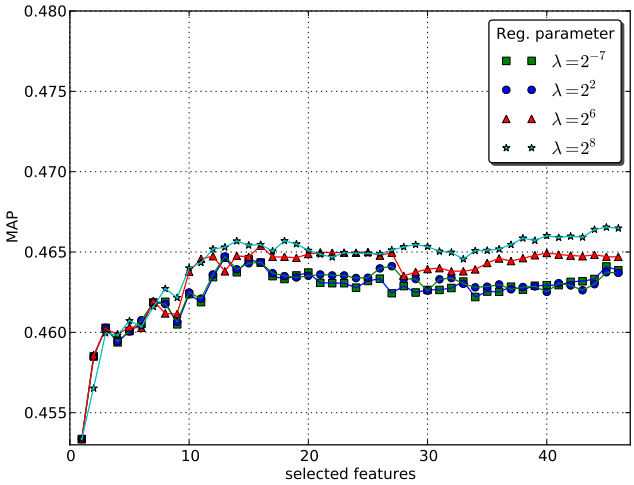

Figure 1: Average MAP on validation sets for MQ2007.

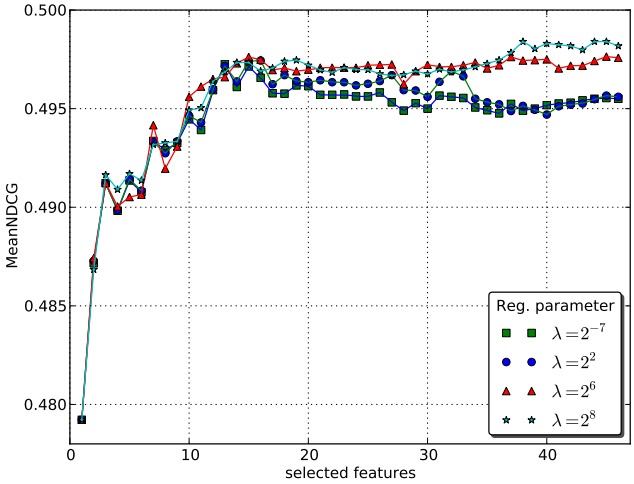

Figure 2: Average Mean NDCG on validation sets for MQ2007.

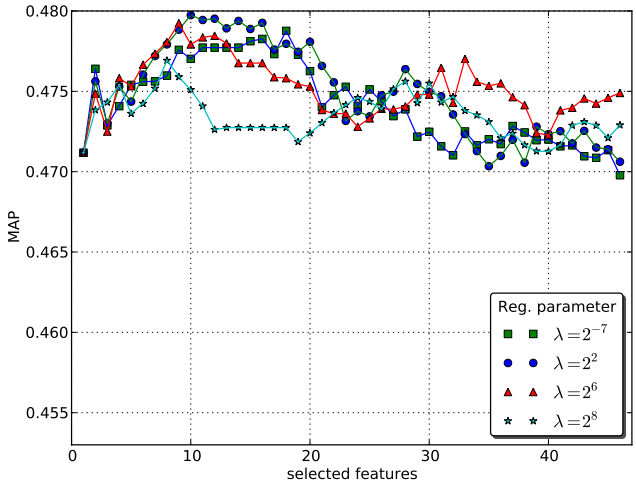

Figure 3: Average MAP on validation sets for MQ2008.

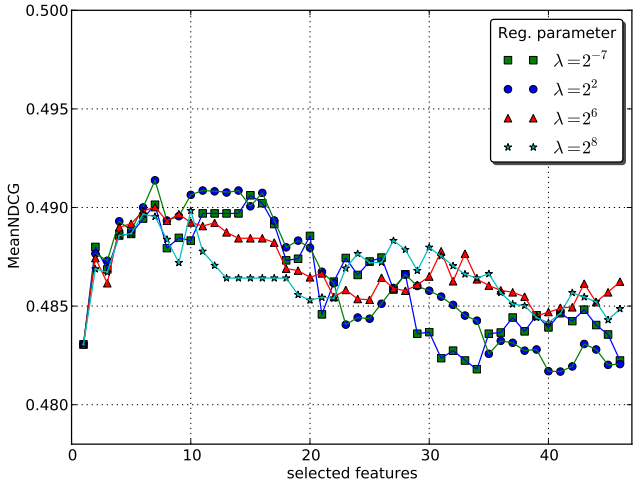

Figure 4: Average Mean NDCG on validation sets for MQ2008.

descriptions of these variations are left for future work.

The greedy forward selection approach can sometimes suffer from the so-called nesting effect, meaning that the best subset of size k, for example, may not necessarily cover the features included in the best subset of size $k-1$. Floating search methods (see e.g. [13, 20, 27]), which are able to discard features selected in previous iterations, have been proposed as a means to deal with this issue. Replacing the greedy search strategy with a floating search would be a fairly straightforward extension to the presented algorithm.

6. CONCLUSION

To conclude, we propose a computationally efficient method for learning sparse predictors for ranking tasks. The method uses on greedy forward selection as a search strategy and leave-query-out cross-validation as a selection criterion. The computational complexity of the method is linear in the number of training examples, in the overall number of features, and in the number of features to be selected. Thus, the method is computationally highly efficient despite the fact that the method optimizes a pairwise ranking loss function and uses a complex cross-validation criterion. Empirical evaluation with the LETOR benchmark data set demonstrates the soundness of the proposed approach.

Acknowledgments

We would like to thank the anonymous reviewers for their insightful comments. This work has been supported by the Academy of Finland.

7. REFERENCES

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

[2] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In L. D. Raedt and S. Wrobel, editors, *Proceedings of the 22nd*

international conference on Machine learning (ICML 2005), pages 89–96. ACM, 2005.

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

[4] O. Chapelle, Q. Le, and A. Smola. Large margin optimization of ranking measures. In *NIPS Workshop: Machine Learning for Web Search*, 2007.

[5] X. Geng, T.-Y. Liu, T. Qin, and H. Li. Feature selection for ranking. In C. L. Clarke, N. Fuhr, N. Kando, W. Kraaij, and A. P. de Vries, editors, *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR 2007)*, pages 407–414. ACM, 2007.

[6] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

[7] H. V. Henderson and S. R. Searle. On deriving the inverse of a sum of matrices. *SIAM Review*, 23(1):53–60, 1981.

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

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

[10] G. H. John, R. Kohavi, and K. Pfleger. Irrelevant features and the subset selection problem. In W. W. Cohen and H. Hirsch, editors, *Proceedings of the Eleventh International Conference on Machine Learning (ICML 1994)*, pages 121–129, San Fransisco, CA, 1994. Morgan Kaufmann Publishers.

[11] R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[12] D. A. Metzler. Automatic feature selection in the markov random field model for information retrieval. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management (CIKM'07)*, pages 253–262. ACM, 2007.

[13] T. Pahikkala, A. Airola, and T. Salakoski. Feature selection for regularized least-squares: New computational short-cuts and fast algorithmic implementations. In *IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, 2010. To appear.

[14] T. Pahikkala, A. Airola, and T. Salakoski. Linear time feature selection for regularized least-squares, 2010. Preprint http://arxiv.org/abs/1003.3570.

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

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

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

[18] T. Pahikkala, W. Waegeman, A. Airola, T. Salakoski, and B. De Baets. Conditional ranking on relational data. In *ECML PKDD '10: Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases*, 2010. To appear.

[19] F. Pan, T. Converse, D. Ahn, F. Salvetti, and G. Donato. Feature selection for ranking using boosted trees. In D. W.-L. Cheung, I.-Y. Song, W. W. Chu, X. Hu, and J. J. Lin, editors, *Proceeding of the 18th ACM conference on Information and knowledge management (CIKM'09)*, pages 2025–2028. ACM, 2009.

[20] P. Pudil, J. Novovičová, and J. Kittler. Floating search methods in feature selection. *Pattern Recognition Letters*, 15(11):1119–1125, 1994.

[21] T. Qin, T.-Y. Liu, J. Xu, and H. Li. LETOR: A benchmark collection for research on learning to rank for information retrieval. *Information Retrieval*, 2010.

[22] A. J. Smola and P. Bartlett. Sparse greedy gaussian process regression. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 619–625. MIT Press, 2001.

[23] P. Vincent and Y. Bengio. Kernel matching pursuit. *Machine Learning*, 48(1–3):165–187, 2002.

[24] H. Yu, J. Oh, and W.-S. Han. Efficient feature weighting methods for ranking. In D. W.-L. Cheung, I.-Y. Song, W. W. Chu, X. Hu, and J. J. Lin, editors, *Proceeding of the 18th ACM conference on Information and knowledge management (CIKM'09)*, pages 1157–1166. ACM, 2009.

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

[26] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In W. B. Croft, D. J. Harper, D. H. Kraft, and J. Zobel, editors, *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR 2001)*, pages 334–342. ACM, 2001.

[27] T. Zhang. Adaptive forward-backward greedy algorithm for sparse learning with linear models. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 1921–1928. MIT Press, 2009.

Publication II

Learning Multi-Label Predictors under Sparsity Budget

Pekka Naula, Tapio Pahikkala, Antti Airola, Tapio Salakoski. In: Anders Kofod-petersen, Fredrik Heintz, Helge Langseth (Eds.), Proceedings of the Eleventh Scandinavian Conference on Artificial Intelligence (SCAI 2011), Frontiers in Artificial Intelligence and Applications 227, 30-39, IOS Press, 2011.

Publication III

Greedy Regularized Least-Squares for Multi-Task Learning

Pekka Naula, Tapio Pahikkala, Antti Airola, Tapio Salakoski. In: Myra Spiliopoulou, Haixun Wang, Diane Cook, Jian Pei, Wei Wang, Osmar Zaane, Xindong Wu (Eds.), 11th IEEE International Conference on Data Mining Workshops (ICDMW'11), 527–533, IEEE Computer Society, 2011.

2011 11th IEEE International Conference on Data Mining Workshops

Greedy Regularized Least-Squares for Multi-Task Learning

Pekka Naula, Tapio Pahikkala, Antti Airola, Tapio Salakoski
University of Turku and Turku Centre for Computer Science
Turku, Finland
Email: forname.surname@utu.fi

***Abstract*—Multi-task feature selection refers to the problem of selecting a common predictive set of features over multiple related learning tasks. The problem is encountered for example in applications, where one can afford only a limited set of feature extractors for solving several tasks. In this work, we present a regularized least-squares (RLS) based algorithm for multi-task greedy forward feature selection. The method selects features jointly for all the tasks by using leave-one-out cross-validation error averaged over the tasks as the selection criterion. While a straightforward implementation of the approach by combining a wrapper algorithm with a black-box RLS training method would have impractical computational costs, we achieve linear time complexity for the training algorithm through the use of matrix algebra based computational shortcuts. In our experiments on insurance and speech classification data sets the proposed method shows a better prediction performance than baseline methods that select the same number of features independently.**

***Keywords*-feature subset selection; multi-task learning; budgeted learning; regularized least-squares**

I. Introduction

Multi-task feature selection concerns the task of selecting jointly a common set of representative features for a group of interrelated learning tasks. Perhaps the main benefit of feature selection [1] is that it allows one to reduce the costs associated with acquiring the feature values. For example, in a multi-sensory environment, where different features correspond to inputs from different sensors, limiting the number of used features may allow one to discard unnecessary sensors. This can allow reductions for example in the manufacturing costs of a product. In such multi-task learning problems [2] where several learned models are applied in the same environment, it may be beneficial to use learning algorithm that can take this into account, rather than to treat all tasks as separate. More specifically, in multi-task feature selection, one may search for such subset of features that can simultaneously support all the considered tasks well.

A typical example where one may require the ability to solve several tasks simultaneously would be a smart handheld device. Given that the device contains a set of sensors like accelerometer, barometer, GPS, temperature sensor, heart rate sensor and so on, it could be used to simultaneously predict certain health related properties of the user, such as calorie consumption, current stress level, whether the user is fit to drive et cetera. These tasks are similar in nature in the sense that some of the sensors could be used in solving several of the tasks. However, the sensors and other sources of information come at a cost, be it money or space in a circuit board, and we may have to make a choice which of them will be included in the final product.

In this work, we study the problem of multi-task learning under a sparsity budget. We assume that a number of different learning tasks need to be solved in the same environment, and that all the tasks share the same feature representations. Acquiring the feature values comes at a cost, therefore it is necessary to limit the number of needed features, while still maintaining as high predictive accuracy as the budget allows. Here, we assume that the training data corresponding to the different tasks may have been gathered from different populations, environments, or points in time. Thus, essentially we have several training sets corresponding to different tasks, and the goal is to be able to find accurate models for each task, while enforcing that the union of the feature sets selected does not grow larger than the allocated budget.

There has been recently an interest in designing methods for feature selection in the multi-task setting. Proposed methods include L1 regularization based techniques, such as the group lasso method [3, 4] and the method of Obozinski et al. [5], the maximum entropy discrimination based method of Jebara [6], and the work of Xiong et al. [7]. Argyriou et al. [8] proposed a method that learns common sparse representations over a pool of related tasks via joint regularization. Zhou et al. [9] presented a group regularization method, exclusive lasso, which was applied to multi-task feature selection setting. The method assumes a negative correlation among the tasks and it was shown to generate sparse solutions due to competitions among variables within the same group. Zhang et al. [10] introduces a probabilistic framework for multi-task feature selection using $l_{1,q}$ norm, where optimal value for q was determined from data automatically.

The wrapper approach [11] is one of the mainstream approaches to feature selection. Here, one performs a search over the power set of features, at each search step training a learning method for the tested feature set, and evaluating the quality of the features via some performance estimate such as cross-validation error. However, the wrapper approach to multi-task feature selection has not been much explored.

978-0-7695-4409-0/11 $26.00 © 2011 IEEE
DOI 10.1109/ICDMW.2011.91

A probable reason for this, as mentioned by Obozinski et al. [5], is the perceived computational cost of wrapper based feature selection, as typical implementations of the approach result in combinatorial growth in the number of times a learner needs to be trained. Even with greedy search strategies, having to re-train a learner for each task, each search step and each round of cross-validation is simply too much for the approach to be practical.

Recently, Pahikkala et al. [12] have proposed a wrapper selection method for the regularized least-squares (RLS) learner. The method, which they call greedy RLS, performs greedy forward selection over the powerset of features, using leave-one-out cross-validation performance as the selection criterion. Linear time complexity is achieved by using matrix algebra based computational shortcuts for operations such as cross-validation computations and updating a predictive model in order to incorporate new features. Naula et al. [13] extended this work to the multi-label learning setting.

In this work, we propose an efficient approach to performing wrapper selection in a general multi-task setting, where features may be selected from multiple datasets sharing same features, but corresponding to different populations and learning tasks. The method generalizes the computational shortcuts proposed for the greedy RLS method, preserving the linear time complexity. The method is applicable both for classification and regression problems. In addition to settings where the individual learning tasks correspond to simple binary classification or single output regression problems, the method can also be used in settings where some or all of the tasks require learning to predict multiple outputs. Thus the method allows also efficient feature selection jointly over multiple multi-class classification, multi-label classification or multi-output regression problems.

In the experiments, we demonstrate that the approach in practice scales to large real-world datasets. Further, given a fixed feature budget the proposed approach leads to higher predictive performance than the baseline methods that do not take into account the related tasks when performing selection.

II. Methods

Let $\mathbf{X}^j \in \mathbb{R}^{m_j \times n}$ for $j = 1, \ldots, t$ be matrices containing the feature representation of the examples in the training set for the jth task, where n is the number of features, m_j is the number of training examples in the jth task and t is the number of tasks. The h, ith entry of $\mathbf{X}^j$ contains the value of the ith feature in the hth training example. Moreover, let $\mathbf{Y}^j \in \mathbb{R}^{m_j \times l_j}$ for $j = 1, \ldots, t$, where l_j is the number of labels per training example in the jth task, be matrices consisting of the labels of the training examples for the t tasks. The number of labels per training example can be larger than one if the task in question is, for example, a multi-class classification task. In multi-class settings, the labels can be restricted to be either -1 or 1 depending whether the data points belong to the class, while they can be any real numbers in multi-label regression tasks (see e.g. Hsu et al. [14] and references therein).

For each of the t tasks, we construct as many linear prediction functions as is required by the tasks. The functions for the jth task can be expressed as

$$f^j(\mathbf{x}) = (\mathbf{x}_{\mathcal{S}})^{\mathrm{T}} \mathbf{W}^j,$$

where $\mathbf{W}^j \in \mathbb{R}^{|\mathcal{S}| \times l_j}$ is a matrix containing the coefficients of the l_j learned predictors for the jth task, $\mathbf{x}$ is a data point for which the prediction of is to be made, and $\mathbf{x}_{\mathcal{S}}$ a vector representation of $\mathbf{x}$ that only contains the features indexed by the set $\mathcal{S}$. For example, if the jth task is a one-versus-all classification problem with l_j different classes, the matrix $\mathbf{W}^j$ contains l_j linear predictors, one per column. Note also that the different tasks may have different amount of labels per data point, that is, it is possible that l_j are not equal for all j.

We assume that the set $\mathcal{S}$ contains feature indices selected by a training algorithm that performs feature selection while constructing the predictors. Moreover, we assume that the number of features we can extract from a data point at prediction time is constrained by a given sparsity budget, that is, we have $|\mathcal{S}| \leq k$ for some $k \in \mathbb{N}$. Note that since the sparsity budget constraint is given for the number of features, it is assumed that all features have an equal extraction cost at the prediction time. This can be easily generalized to a setting in which each feature would have a different extraction cost. However, this is not considered further in this paper, because of limited space and the lack of available data for practical experiments.

The most straightforward approach for constructing the predictors under a common budget constraint is to simply train them separately and select, say k/t features separately for each task. After this, it is possible to perform a subsequent training phase in which the predictors for each task are retrained with the union of the separately selected feature sets. This can be achieved via numerous off-the-shelf feature selection methods available for solving single-task problems. It may of course happen that the size of the union of the separately selected feature sets is smaller than k, if the sets are not mutually disjoint. Then, the selection process can be continued until the sparsity budget is completely fulfilled. A natural step forward is to select the features jointly for all tasks, favoring such features that increase the performance averaged over all tasks.

Before considering the actual training algorithms, we present some of the building blocks. As a base learner, we use regularized least-squares (RLS) [15] for multiple outputs (also known as ridge regression [16]), a state-of-the art machine learning method suitable for several types of machine learning tasks. With multiple outputs, we refer to the fact that the tasks we consider may be instances of multi-label prediction problems, such as the standard

one-versus-all approach for multi-class classification for example. Training of a multi-output RLS can be expressed for a single task with training data $\mathbf{X} \in \mathbb{R}^{m \times n}$, $\mathbf{Y} \in \mathbb{R}^{m \times l}$ as finding a solution to the following problem:

$$\underset{\mathbf{W} \in \mathbb{R}^n}{\operatorname{argmin}} \left\{ \|\mathbf{X}\mathbf{W} - \mathbf{Y}\|_F^2 + \lambda \|\mathbf{W}\|_F^2 \right\}, \qquad (1)$$

where $\lambda > 0$ is a regularization parameter and $\|\mathbf{W}\|_F$ is the Frobenius matrix norm. The first term in (1), called the empirical risk, measures how well the prediction function fits to the training data. The second term is called the regularizer and it controls the tradeoff between the empirical error on the training set and the complexity of the prediction function. The regularization parameter λ can be set separately for each of the tasks, if some tasks require more regularization than the others.

As a feature selection criterion, we use the leave-one-out (LOO) cross-validation performance averaged over all tasks. It is well-known that, for RLS-based learning algorithms, LOO performance can be calculated as efficiently as training set error via computational short-cuts based on matrix algebra (see e.g. Elisseeff and Pontil [17] and references therein). For a more in-depth description of LOO performance, we refer to Lachenbruch [18], Elisseeff and Pontil [17].

Pahikkala et al. [12] proposed a feature selection method for single-task problems with a single label per training example, named greedy RLS, whose computational complexity is $O(kmn)$, that is, the method is linear with respect to the number of data points in the training set m, to the overall number of features among which the selection is made n and to the number of features selected k. This method was later extended for tasks having multiple labels per data point by Naula et al. [13]. The modification of greedy RLS for multi-task learning is quite straightforward, since only the selection criterion has to be modified.

As a training algorithm for learning predictors with a restricted sparsity budget, we present a greedy forward feature selection method for RLS with LOO criterion. By greedy, we indicate that the algorithm starts from an empty set of features and adds one feature at a time to the set but never removes any features from the set. A high level pseudo code of the multi-task greedy RLS algorithm is given in Algorithm 1. The technical details consisting of the efficient matrix algebra -based computational short-cuts are given in Algorithms 2, 3 and 4. The short-cuts are analogous to those presented by Pahikkala et al. [12] for a single-task single-label greedy RLS and their correctness and computational complexities can be shown in similar way. Therefore, these considerations are not repeated in this paper.

In the pseudo code, the outermost loop adds one feature at a time into the set of selected features $\mathcal{S}$ until the size of the set has reached the sparsity budget k. The middle loop goes through every feature that has not yet been added into the set of selected features. For the ith feature available for addition, the inner loop computes the average LOO performance over the t tasks for all RLS predictors trained using the features $\mathcal{S} \cup \{i\}$. Instead of the average LOO, also other approaches such as weighted average could be used to measure the overall performance. After going through all feature candidates, the algorithm then adds the feature with the best average LOO performance into the set of selected features.

Algorithm 1 MTGRLS($\mathbf{X}^1, \ldots, \mathbf{X}^t, \mathbf{Y}^1, \ldots, \mathbf{Y}^t, k$)

```
S ← ∅  ▷ The current set of selected features common for all tasks.
Initialize(X^1, ..., X^t, Y^1, ..., Y^t, λ)
while |S| < k do                          ▷ Select k common features.
    e ← ∞
    b ← 0
    for i ∈ {1, ..., n} \ S do
        e_avg ← 0
        for j ∈ {1, ..., t} do
            e_{i,j} ←Looperf(i, j)
            e_avg ← e_avg + e_{i,j}/t
        if e_avg < e then
            e ← e_avg
            b ← i
    S ← S ∪ {b}  ▷ Update the set of selected features.
    for j ∈ {1, ..., t} do
        Update(b, j)  ▷ Update predictors for each task.
return W^1, ..., W^t, S
```

Algorithm 2 Initialize($\mathbf{X}^1, \ldots, \mathbf{X}^t, \mathbf{Y}^1, \ldots, \mathbf{Y}^t, \lambda$)

```
for j ∈ {1, ..., t} do
    A^j ← λ^{-1} Y^j        ▷ Initialize m_j × l_j-matrix A^j.
    d^j ← λ^{-1} 1          ▷ Initialize m_j-vector d^j.
    C^j ← λ^{-1} X^{jT}     ▷ Initialize m_j × n-matrix C^j.
```

Algorithm 2 called in the beginning of the main program allocates space and initializes all cache matrices and vectors required in the computational short-cuts. For each task, the algorithm must initialize two matrices and one vector that are used to store the intermediate results of the feature selection process.

Algorithm 3, called as Looperf(i, j) during each iteration of the inner loop of the main program, returns a real value corresponding to the LOO performance of a RLS predictor trained with a feature set $\mathcal{S} \cup \{i\}$ for the jth task. The computational complexity of this algorithm is $O(m_j)$, that is, it depends only on the number of training examples in the jth task. This is because it takes advantage of the previously computed and cached results stored in $\mathbf{A}^j$, $\mathbf{d}^j$ and $\mathbf{C}^j$.

Algorithm 4, called as Update(b, j) for each task in the end of each iteration of the outer loop of the main program,

Algorithm 3 Looperf(i, j)

$\mathbf{v} \leftarrow (\mathbf{X}_i^j)^{\mathrm{T}}$
$\mathbf{u} \leftarrow \mathbf{C}_{:,i}^j(1 + \mathbf{v}^{\mathrm{T}}\mathbf{C}_{:,i}^j)^{-1}$
$\tilde{\mathbf{A}} \leftarrow \mathbf{A}^j - \mathbf{u}(\mathbf{v}^{\mathrm{T}}\mathbf{A}^j)$
for $h \in \{1, \ldots, m_j\}$ **do** ▷ Compute LOO predictions for training examples
    $\tilde{\mathbf{d}}_h \leftarrow \mathbf{d}_h - \mathbf{u}_h\mathbf{C}_{h,i}^j$
    $\mathbf{P}_h \leftarrow \mathbf{Y}_h^j - (\tilde{\mathbf{d}}_h)^{-1}\tilde{\mathbf{A}}_h$
$e \leftarrow L(\mathbf{P}, \mathbf{Y}^j)$ ▷ LOO error measured with loss L
return e

Algorithm 4 Update(b, j)

$\mathbf{v} \leftarrow (\mathbf{X}_b^j)^{\mathrm{T}}$
$\mathbf{u} \leftarrow \mathbf{C}_{:,b}^j(1 + \mathbf{v}^{\mathrm{T}}\mathbf{C}_{:,b}^j)^{-1}$
$\mathbf{A}^j \leftarrow \mathbf{A}^j - \mathbf{u}(\mathbf{v}^{\mathrm{T}}\mathbf{A}^j)$
for $h \in \{1, \ldots, m_j\}$ **do**
    $\mathbf{d}_h^j \leftarrow \mathbf{d}_h^j - \mathbf{u}_h\mathbf{C}_{h,b}^j$
$\mathbf{C}^j \leftarrow \mathbf{C}^j - \mathbf{u}(\mathbf{v}^{\mathrm{T}}\mathbf{C}^j)$

updates the cached results required by Algorithm 3 so that they correspond to the currently selected set of features. This is done always after a new feature has been added into $\mathcal{S}$. The computational complexity of this algorithm, dominated by the size of the matrices $\mathbf{A}$ and $\mathbf{C}$, is $O(m_j l_j + m_j n)$.

Putting everything together, the computational complexity of the main algorithm scales as $O(knc)$, where

$$c = \sum_{j=1}^{t} m_j l_j$$

is the overall number of all labels in the overall multi-task multi-label training set. This complexity is linear with all three variables, and hence it is analogous to that of greedy RLS presented by Pahikkala et al. [12].

III. Experiments

A. Datasets and Setup

In the experiments we compare two baseline methods (Method 1 and Method 2) and our proposed method (Method 3) for budgeted multi-task learning. We assume that all the features have equal cost and we are given a budget constraint k on the number of features, and the number of different tasks is t. Method 1 performs feature selection for all the t tasks separately, using the (multi-class variation of) greedy RLS algorithm, so that the individual training processes do not share any knowledge between each other. The budget is divided evenly between the tasks, meaning that each individual predictor uses k/t features. Method 1 can be considered wasteful in the sense that none of the tasks benefit from the features selected for the other tasks. Method 2 aims to fix this problem as follows. All the features are again selected as in Method 1, but after the selection is finished, the predictors for each task are re-trained using the union of all the selected feature sets. Finally, Method 3 (Algorithm 1) aims to improve on Method 2 by performing the selection process itself jointly over all the tasks, in each step selecting such features that give the best average performance over all the tasks. The methods are compared over a varying range of budgets. In the comparisons, the budget size is always computed as the size of the final set of selected features. Thus, if some of the features selected by Method 1 are shared by multiple tasks, each such feature incurs only unit cost in the budget. This means that the comparison between the three methods are always made so that they all use exactly the same number of features.

We carry out our experiments on two real world datasets, Isolet spoken alphabet recognition[1] and CoIL 2000[2]. The Isolet dataset consists of 7797 examples[3] of spoken English alphabet characters, where 150 speakers have spoken each letter twice. The goal of the task is to find out which alphabet has been spoken by the speaker based on 617 acoustic features such as spectral coefficients, sonorant- and post-sonorant features etc (more about the features, see Fanty and Cole [19]). The dataset is divided into 5 disjoint subsets, isolet1-5, where each subset consists of 30 similar speakers. CoIL dataset consists of 5822 training and 4000 test examples of customers of an insurance company. The goal of the task is to predict who would be interested in buying a caravan insurance policy based on 86 features of product usage and socio-demographic data.

We modify the datasets in order to make them usable for the purpose of multi-task learning. We follow the experimental setup on the Isolet data presented by Parameswaran and Weinberger [20] by treating each subsets, isolet1-5, as its own classification task (5 tasks). We simulate a setting where the different tasks would correspond to different languages as follows. We select a reduced number of disjoint labels for each of the tasks (Labels A-E, F-J, K-O, P-T and U-Z for the tasks 1,2,3,4 and 5, respectively) and remove all the examples that do not belong under these labels in the datasets. Moreover, we generated the training and test sets by random splits of the data, so that 2/3 of the examples from each task belong to the training set and 1/3 to the test set. CoIL dataset is transformed into multiple tasks by using categorical features 1,4,5,6,44 and 86 as output variables and leaving the remaining 80 features as the joint data set. Each task has an identical input dataset but a different number of output labels (Labels 1-40[4], 1-6, 1-10, 0-9, 0-3 and 0-1 for the tasks 1,2,3,4,5 and 6, respectively).

The regularization parameter λ is set to 1 in this study. We carried out experiments also with other regularization

[1] available at UCI Machine Learning Repository
[2] available at http://kdd.ics.uci.edu/databases/tic/tic.html
[3] Three examples are historically missing.
[4] class number 14 does not include any samples

parameter values, the results were very similar to those obtained with λ value 1. All the individual tasks in both the data sets correspond to multi-class classification problems, average accuracy over all the tasks on the test sets is used to evaluate the overall behavior of both of the algorithms.

B. Results

We plot the performance curves for the Isolet data and CoIL data in Figures 1 and 2, respectively. Both figures contain the performance curves for the proposed method (Method 3) and the two baseline approaches. The average multi-class accuracy is plotted as a function of the available budget.

We can see in Figures 1 and 2 that multi-task learning (Method 3) outperforms both single-task methods (Method 1 and Method 2) on both Isolet and CoIL datasets. Particularly with the low available budgets, the difference in performances is clear. Thus, the experimental results support the notion that selecting features jointly over the tasks is beneficial when limited by a common feature budget.

Figures 3 and 4 shows the cost curves on Isolet and CoIL datasets, respectively. The cost curves indicate the overlap between the feature sets selected for each task separately in a single-task learning by Method 1. *Realized cost* represents the size of the union of feature sets over all the tasks as a function of the given budget constraint. *Max cost* represents the upper bound on the realized cost for a given budget constraint. This situation can be happened only, if the feature sets selected for all the tasks are mutually exclusive. *Min cost* represents the lower bound on the realized cost and it can occur in a case where the feature sets selected for all the tasks are exactly the same. Max cost and realized cost curves are quite close in Figure 3 on Isolet data, which indicates that the feature sets selected for the separate tasks are very different. This situation favors Methods 2 and 3 over Method 1, as more features can be allocated for the individual predictors. The clearly higher accuracy of Method 3 compared to Method 2 on Isolet (see Fig. 1) further indicates that the joint selection strategy leads to better performance than the combination of the independently selected feature sets. On the contrary, the realized cost curve is quite similar to the min cost curve on CoIL dataset, indicating the situation where the feature sets selected for different tasks overlap much more than for Isolet. This is reflected in the performance differences that are in this case smaller (see Fig. 2).

Tables I and II provide the classification accuracies for Methods 1, 2 and 3, for each of the tasks on Isolet data and CoIL data over some of the selected budgets, respectively. The numbers in the brackets for Method 1 indicate the size of the feature sets per each individual task on selected budgets. It should be noted that the sum of the sizes of the feature sets over all the tasks on different budgets is often bigger than selected budget due to the redundant features.

Table I
PERFORMANCE FOR TASKS ON ISOLET DATA WITH METHOD 1 (TOP), METHOD 2 (MIDDLE), AND METHOD 3 (BOTTOM).

Budget	avg	task 1	task 2	task 3	task 4	task 5
4 (1)	0.381	0.400	0.390	0.400	0.400	0.317
8 (2)	0.580	0.600	0.610	0.600	0.600	0.492
16 (4)	0.868	0.800	0.950	0.820	0.880	0.891
26 (6)	0.901	0.830	0.950	0.890	0.920	0.917

Budget	avg	task 1	task 2	task 3	task 4	task 5
4	0.527	0.360	0.670	0.400	0.630	0.575
8	0.805	0.700	0.890	0.760	0.890	0.783
16	0.882	0.790	0.980	0.840	0.910	0.892
26	0.937	0.840	0.990	0.940	0.950	0.967

Budget	avg	task 1	task 2	task 3	task 4	task 5
4	0.740	0.530	0.930	0.640	0.860	0.742
8	0.864	0.820	0.970	0.810	0.870	0.850
16	0.894	0.860	0.980	0.840	0.890	0.900
26	0.950	0.940	1.000	0.920	0.950	0.942

Table II
PERFORMANCE FOR TASKS ON COIL DATA WITH METHOD 1 (TOP), METHOD 2 (MIDDLE), AND METHOD 3 (BOTTOM).

Budget	avg	task 1	task 2	task 3	task 4	task 5	task 6
5 (1)	0.588	0.120	0.548	0.358	0.518	0.964	0.941
19 (5)	0.631	0.252	0.598	0.450	0.580	0.964	0.940
35 (10)	0.657	0.298	0.616	0.501	0.623	0.964	0.940
48 (15)	0.658	0.303	0.606	0.502	0.634	0.964	0.940

Budget	avg	task 1	task 2	task 3	task 4	task 5	task 6
5	0.603	0.233	0.553	0.393	0.534	0.964	0.940
19	0.647	0.284	0.605	0.486	0.605	0.964	0.940
35	0.665	0.314	0.620	0.519	0.634	0.964	0.940
48	0.674	0.339	0.629	0.523	0.650	0.964	0.940

Budget	avg	task 1	task 2	task 3	task 4	task 5	task 6
5	0.619	0.246	0.592	0.440	0.533	0.964	0.941
19	0.659	0.296	0.615	0.513	0.625	0.964	0.941
35	0.677	0.339	0.633	0.533	0.655	0.964	0.940
48	0.680	0.352	0.627	0.538	0.656	0.964	0.940

IV. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a computationally efficient approach to learning multiple tasks under a sparsity budget using greedy forward selection. The proposed method combines the ideas of multi-task learning and feature subset selection by selecting a minimal set of common features simultaneously for several tasks. We evaluated the performance of the method on two real world public datasets, Isolet spoken alphabet recognition and CoIL2000. The results show that our proposed multi-task method outperforms approaches that do not share information between the tasks during the selection process, especially with a small available budget. The results are fascinating for example from the industry point of view due to the fact that many small-sized home appliances or portable devices are manufactured under hard budget constraints resulting in a restricted space for a numerous of expensive and large-sized hardware components.

So far we have been investigating settings where all the features are assumed to be equally costly but it would be worth while to extend the method to settings where features are associated with variable costs. Finally, especially in the

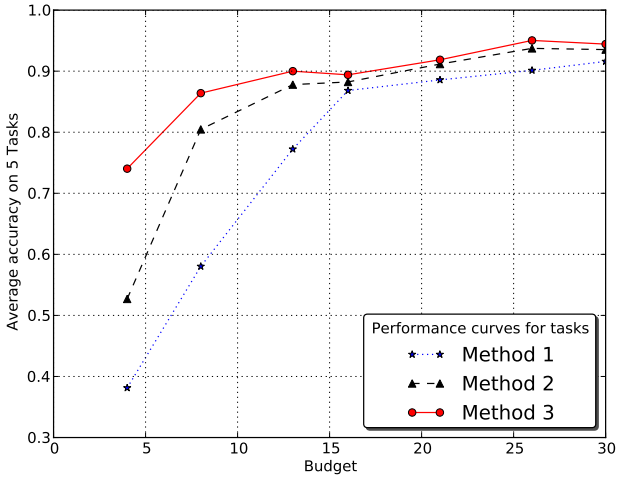

Figure 1. Performance curves on Isolet data.

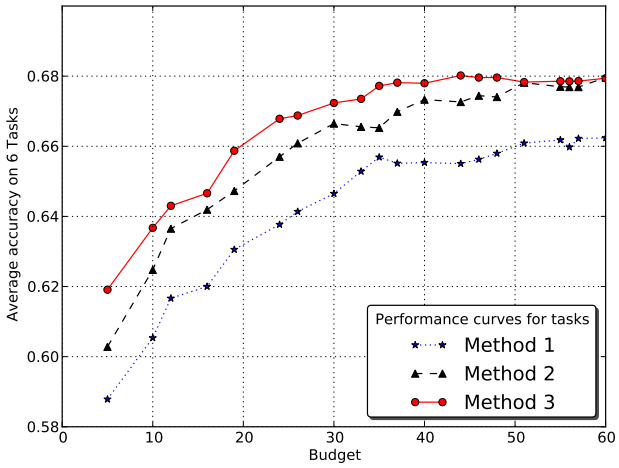

Figure 2. Performance curves on CoIL data.

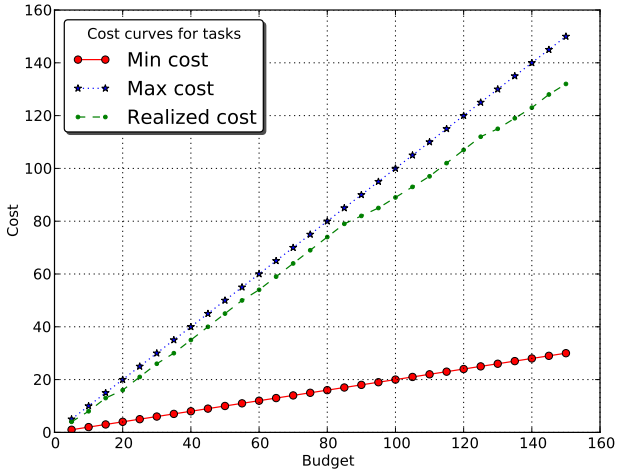

Figure 3. Cost curves for Isolet data.

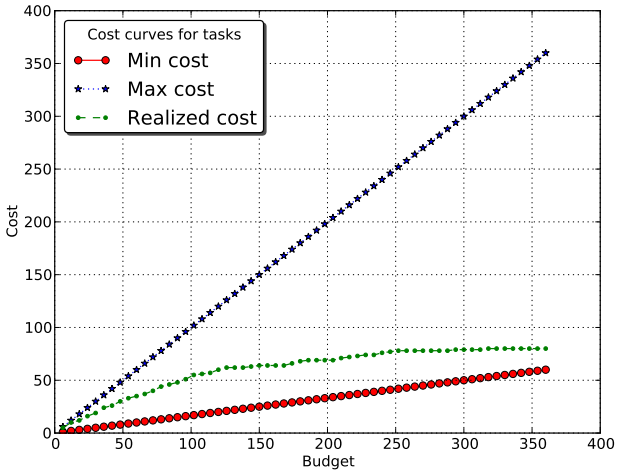

Figure 4. Cost curves for CoIL data.

context of variable feature costs, it might be beneficial to use also other search strategies than the greedy forward selection such as genetic algorithms [21] due to their effectiveness in multi-criteria optimization.

ACKNOWLEDGMENT

This work has been supported by the Academy of Finland (grants 134020 and 128061).

REFERENCES

[1] A. L. Blum and P. Langley, "Selection of relevant features and examples in machine learning," *Artificial Intelligence*, vol. 97, pp. 245–271, 1997.

[2] R. Caruana, "Multitask learning," *Machine Learning*, vol. 28, pp. 41–75, 1997.

[3] B. A. Turlach, W. N. Venables, and S. J. Wright, "Simultaneous variable selection," *Technometrics*, vol. 47, no. 3, pp. 349–363, 2005.

[4] M. Yuan and Y. Lin, "Model selection and estimation in regression with grouped variables," *Journal of the Royal Statistical Society, Series B*, vol. 68, pp. 49–67, 2006.

[5] G. Obozinski, B. Taskar, and M. I. Jordan, "Joint covariate selection and joint subspace selection for multiple classification problems," *Statistics and Computing*, vol. 20, pp. 231–252, April 2010.

[6] T. Jebara, "Multitask sparsity via maximum entropy discrimination," *Journal of Machine Learning Research*, vol. 12, pp. 75–110, 2011.

[7] T. Xiong, J. Bi, R. B. Rao, and V. Cherkassky, "Probabilistic joint feature selection for multi-task learning," in *Proceedings of the Seventh SIAM International Conference on Data Mining (SDM 2007)*. Society for Industrial and Applied Mathematics, 2007, pp. 332–342.

[8] A. Argyriou, T. Evgeniou, and M. Pontil, "Convex multi-task feature learning," *Machine Learning*, vol. 73, no. 3, pp. 243–272, 2008.

[9] Y. Zhou, R. Jin, and S. Hoi, "Exclusive lasso for multi-task feature selection," in *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, ser. JMLR Workshop and Conference Proceedings, Y. W. Teh and M. Titterington, Eds., vol. 9, 2010, pp. 988–995.

[10] Y. Zhang, D.-Y. Yeung, and Q. Xu, "Probabilistic multi-task feature selection," in *Advances in Neural Information Processing Systems 23*, J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, Eds. MIT Press, 2010, pp. 2559–2567.

[11] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial Intelligence*, vol. 97, pp. 273–324, 1997.

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

[13] P. Naula, T. Pahikkala, A. Airola, and T. Salakoski, "Learning multi-label predictors under sparsity budget," in *Eleventh Scandinavian Conference on Artificial Intelligence, SCAI 2011*, ser. Frontiers in Artificial Intelligence and Applications, A. Kofod-Petersen, F. Heintz, and H. Langseth, Eds., vol. 227. IOS Press, May 2011, pp. 30–39.

[14] D. Hsu, S. Kakade, J. Langford, and T. Zhang, "Multi-label prediction via compressed sensing," in *Advances in Neural Information Processing Systems 22*, Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, Eds. MIT Press, 2009, pp. 772–780.

[15] R. Rifkin, "Everything old is new again: A fresh look at historical approaches in machine learning," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, 2002.

[16] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, pp. 55–67, 1970.

[17] A. Elisseeff and M. Pontil, "Leave-one-out error and stability of learning algorithms with applications," in *Advances in Learning Theory: Methods, Models and Applications*, ser. NATO Science Series III: Computer and Systems Sciences, J. Suykens, G. Horvath, S. Basu, C. Micchelli, and J. Vandewalle, Eds. Amsterdam, Netherlands: IOS Press, 2003, vol. 190, ch. 6, pp. 111–130.

[18] P. A. Lachenbruch, "An almost unbiased method of obtaining confidence intervals for the probability of misclassification in discriminant analysis," *Biometrics*, vol. 23, no. 4, pp. 639–645, 1967.

[19] M. A. Fanty and R. Cole, "Spoken letter recognition," in *Advances in Neural Information Processing Systems 3*, M. J. Lippman, R. P. and D. S. Touretzky, Eds. Morgan Kaufmann, 1990, pp. 220–226.

[20] S. Parameswaran and K. Weinberger, "Large margin multi-task metric learning," in *Advances in Neural Information Processing Systems 23*, J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, Eds., 2010, pp. 1867–1875.

[21] J. Yang and V. Honavar, "Feature subset selection using a genetic algorithm," *IEEE Intelligent Systems and their Applications*, vol. 13, no. 2, pp. 44–49, 1998.

Publication IV

Multi-Label Learning Under Feature Extraction Budgets

Pekka Naula, Antti Airola, Tapio Salakoski, Tapio Pahikkala. Pattern Recognition Letters 40, 56-65, 2014.

Publication V

Learning low cost multi-target models by enforcing sparsity

Pekka Naula, Antti Airola, Tapio Salakoski, and Tapio Pahikkala. In Moonis Ali, Young Sig Kwon, Chang-Hwan Lee, Juntae Kim, and Yongdai Kim, editors, The 28th International Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems (IEA/AIE 2015), Lecture Notes in Computer Science, pages 252–261. Springer, 2015.

Learning Low Cost Multi-target Models by Enforcing Sparsity

Pekka Naula( ), Antti Airola, Tapio Salakoski, and Tapio Pahikkala

University of Turku, Turku, Finland
{pekka.naula,antti.airola,tapio.salakoski,tapio.pahikkala}@utu.fi

Abstract. We consider how one can lower the costs of making predictions for multi-target learning problems by enforcing sparsity on the matrix containing the coefficients of the linear models. Four types of sparsity patterns are formalized, as well as a greedy forward selection framework for enforcing these patterns in the coefficients of learned models. We discuss how these patterns relate to costs in different types of application scenarios, introducing the concepts of extractor and extraction costs of features. We experimentally demonstrate on two real-world data sets that in order to achieve as low prediction costs as possible while also maintaining acceptable predictive accuracy for the models, it is crucial to correctly match the type of sparsity constraints enforced to the use scenario where the model is to be applied.

Keywords: Multi-target learning problems · Sparsity patterns · Feature selection · Extractor cost · Extraction cost

1 Introduction

Linear models are often preferred in real world prediction tasks due to their simplicity, understandability and ease of evaluation. A linear model

$$Y = w_0 + w_1 X_1 + w_2 X_2 + \ldots + w_d X_d + \epsilon,$$

describes a relationship between the output variable Y and input variables $\{X_i\}_{i=1}^{d}$. The unknown regression coefficients $\{w_i\}_{i=0}^{d}$ have to be estimated before the model can be used in prediction tasks such as classification or regression. Typically, the regression coefficients are determined by learning from data that are acquired, depending on the application in question, for example by making measurements using physical sensors, sending out questionnaires, performing medical experiments, following user behaviour online, or buying from a data provider. Usually the data does not come for free [9], rather there are costs associated with the features needed during model construction and when making predictions. Measuring the data takes time, hardware components needed for measurements are expensive, data providers may charge money for each variable recorded etc. Thus often it is beneficial if one can produce accurate predictions using as few input variables as possible, that is, sparsity of the model is preferred.

M. Ali et al. (Eds.): IEA/AIE 2015, LNAI 9101, pp. 252–261, 2015.
DOI: 10.1007/978-3-319-19066-2_25

In single-target regression or classification tasks, where the model is represented as a vector of coefficients, the degree of sparsity for the model can be defined as the number of non-zero components in the coefficient vector. Formally, $\|w\|_0 = |\{w_i \neq 0, i = 0, \ldots, d\}|$ where $\|\,\cdot\,\|_0$ denotes l_0-norm. Sparse models contain less variables than dense ones and are therefore easier to interpret. Sometimes too complex models are prone to overfitting the data, enforcing sparsity may prevent this as a form of model regularization. However, the advantage considered in this work is the cost reduction [7] gained through removal of redundant or irrelevant features.

Many sparse modeling methods have been developed in order to remove unnecessary features from a set of candidate features. One of the most popular method is the lasso [8] which is often used to encourage sparsity by regularizing with l_1-norm. Sometimes prior knowledge about expected sparsity patterns can be used in feature selection, which motivates for example the use of the group lasso [10] that encourages sparsity in group level. The other popular family of techniques for inducing sparseness in coefficients of the model, is to implement a search over the power set of available features [2]. Considering all the possible feature subsets is computationally infeasible, rather suboptimal search heuristics are applied. In practice, the most widely used search approach is the greedy forward selection method (see e.g. [11]). We note that many standard dimensionality reduction techniques, such as principal component analysis, are not suitable for learning sparse models. While they project the data into a low-dimensional space, they still require all the original features to perform the projection.

Typically, rather than requiring only a single prediction, many real-world problems constitute rather multi-target prediction tasks. Here one may based on same variables make predictions on multiple related outcomes, using several linear models. Examples of such settings include multi-class classification with one-versus-all encoding [6], multi-label classification problems [3] and multivariate regression problems. More generally we may consider multi-task prediction problems where the tasks still share a feature representation, but unlike in the previous cases the training data for different tasks may be gathered from different sources, and predictions are not necessary needed for all possible targets during prediction time. For multi-target prediction problems, the coefficients of the models can be represented in a matrix form, where rows correspond to different targets and columns correspond to features. A matrix is considered sparse, if it consists primarily of zero elements. However, compared to the single-target case the concept of matrix sparsity allows for different sparsity patterns, depending whether the non-zero elements are distributed freely, or along rows or columns in the matrix. Depending on the application, two coefficient matrices with exactly the same number of non-zero elements may lead to drastically different evaluation costs.

In this paper, we study how one can lower the costs of making predictions for multi-target learning problems by enforcing sparsity on the matrix containing the coefficients of the linear models. We consider different types of costs that lead to favoring different types of sparsity patterns in the predictive models, and

show how such sparsity can be obtained. The goal of this work is not to propose completely new algorithms, since the optimization problems needed for learning sparse predictors can already be solved sufficiently well using existing greedy or lasso type of methods. Rather, the goal of this work is to explore what are the types of sparsity patterns best suited for different types of application settings. For our experiments we implement greedy regularized least-squares methods, as they have been recently shown in a comprehensive experimental comparison to have state-of-the art performance when learning linear multi-target models with sparsity constraints [3].

2 Sparsity Patterns

In this section, we consider a setting where several targets have to be predicted under a strict budget. We assume that features or groups of features have unit costs (a natural extension is to allow varying costs, but for the simplicity of presentation and lack of suitable data this case is not considered in this work). By *feature extractor* we refer to a source that generates features. Such an extractor can for example be a sensor embedded into a device, a medical test, a question in a questionnaire etc. We consider two settings, one where there is a one-to-one mapping between features and extractors, and one with one-to-many mapping, where each extractor produces a group of features. Correspondingly, by *feature extraction* we refer to the procedure, where the extractors are used to obtain the features. We assume that the available extractor or extraction budgets are essentially unlimited when gathering the training data and building the model, but the final models should be as cheap to use as possible. This is a fairly reasonable assumption for example in product development in industry, where one may accept large initial R&D costs when building model prototypes, but the per unit cost of the final products should be as low as possible.

Thus, we may distinguish between two different costs, the extractor or the extraction costs. The extractor cost refers to the price of the feature extractors needed for computing the predictions. The cost could be for example the amount of money needed to manufacture the sensors embedded in a device or the number of medical devices needed in a hospital ward. The extractor cost can be considered as an initial investment made before the predictor can be used. In contrast, extraction cost is paid every time a prediction is made. This can be, for example, the monetary costs of performing blood tests for a patient, the time spent filling a questionnaire or the time spent for computing the predictions. In the single-target case there is no real need to distinguish between these two settings, as minimizing the number of non-zero coefficients in the model (possibly subject to variable costs and group structure) will lead to good solutions for both goals. However, for multi-output prediction problems the settings clearly differ.

The coefficients of the linear models of the predictors are denoted in matrix form, with rows and columns corresponding to targets and features, respectively. By *Sparsity patterns* we refer to the different ways non-zero coefficients can be distributed in the matrix. In this study we consider four types of sparsity patterns (see Fig. 1, where blue cells denote non-zero coefficients). Type I

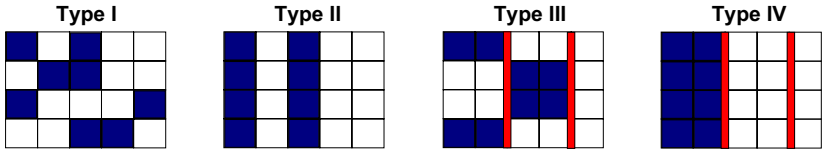

Fig. 1. Sparsity patterns. Rows correspond to targets and columns to features. The columns are ordered into three groups: $\{1, 2\}$, $\{3, 4\}$ and $\{5\}$.

presents a pattern where non-zero coefficients are distributed freely over the rows and columns. Type II denotes the pattern where non-zero coefficients are set columnwise, reflecting a setup where all the targets require the same features for prediction. Type III and IV extend these two basic cases with the additional assumption that the features are ordered in groups and they are selected a whole group at a time. Next, we discuss how these sparsity patterns relate to different types of extractor or extraction costs that may be encountered in practical settings.

2.1 Predicting Multiple Outputs Simultaneously

In the standard setting for multi-class or multi-label classification, with binary encoding of the classes or labels, or in multi-output regression, one needs to provide predictions for all possible outputs simultaneously. For example, one may analyze a picture aiming to detect multiple properties of it simultaneously (outside or inside, contains a face, nature or city), or assign a news document to one or several possible categories (sports, weather, business...). Since the models are evaluated at the same time, clearly any feature that has a non-zero coefficient on any of the rows needs to be extracted.

First, let us consider the feature extractor costs for the case where each feature is generated by its own extractor (see Fig. 1). Here, the cost of the Type I is 5, since we need to be able to generate every single feature for computing the predictions. In contrast, the cost for Type II and IV is only 2, since only two features is needed for prediction. If we further assume the previously defined group structure, then Type I needs all 3 extractors, Types II and III two of them and Type IV only one of them. In the latter setting the feature extraction costs coincide with the extractor costs, since every feature that is present in the model for even one target will be used during prediction.

2.2 Predicting Outputs Independently

Prediction may also be done independently for the outputs. For example, mobile phone applications may use the same set of available feature extractors (sensors and basic software on phone), but the predictions are made independently at different time points (one app for health monitoring while running, other for managing diabetes while at home).

Here (see Fig. 1 again), the feature extractor costs are the same as in the previous case, since the number of required extractors does not depend on whether

the predictions are made at the same time or not. However, if the group structure is not assumed the extraction costs are the same for all the models. For all the outputs, on average two features need to be extracted. Here, instead of enforcing sparsity on column-level (Type II or IV sparsity), it is often advantageous to select features independently for each linear model (Type I or III sparsity), since allowing this freedom results in no extra costs. As a summary, if the extractor costs are the main concern, then this setting corresponds to the one considered in the previous subsection. However, in case one needs to consider only the extraction costs, then the models corresponding to different targets may be independently learned using standard sparsity-enforcing methods such as (group) lasso or greedy search methods.

3 Framework for Sparse Multi-target Problems

In the following considerations we reserve bold lowercase and uppercase letters for vectors and matrices, respectively. Moreover, we denote by $A_{\mathcal{I}}$ the submatrix of A containing only the columns indexed by $\mathcal{I}$.

3.1 Optimization Framework

In this section we present a framework for learning linear multi-target predictors under budget constraints. Let T be the number of targets and k and be the budget. Let $\mathbf{X}^t \in \mathbb{R}^{n_t \times d}$ be a design matrix for tth problem, where the rows correspond to training instances and the columns to feature values, and n_t denotes the number of samples for the target t and d is the number of features, and let $\mathbf{y}^t \in \mathbb{R}^{n_t}$ be the output vector for the tth target. We also divide the features into disjoint groups $G_i \subset \{1, \ldots, d\}$, $i = 1, \ldots, J$ and $G_i \cap G_j = \emptyset, i \neq j$.

The aim is to create a linear model for each of the T targets:

$$\mathbf{y}^t = \mathbf{X}^t \overline{\mathbf{w}}^t + \epsilon^t, t = 1, \ldots, T,$$

where $\overline{\mathbf{w}}^t \in \mathbb{R}^d$ are true regression coefficients that are going to be estimated and ϵ^t are error terms. Let $\mathbf{w}^t$ be estimated regression coefficients for target t and let $\mathbf{W}$ be a matrix that contains these vectors of coefficients in rows. Applying the square loss for least squares regression, we can write the multi-target optimization problem in the following form:

$$\underset{\mathbf{W} \in \mathbb{R}^{T \times d}}{\operatorname{argmin}} \sum_{t=1}^{T} \|\mathbf{y}^t - \mathbf{X}^t \mathbf{w}^t\|_2^2 + \lambda_1 \Omega_1(\mathbf{W}) + \lambda_2 \Omega_2(\mathbf{W}), \tag{1}$$

where $\Omega_1, \Omega_2 : \mathbb{R}^d \to \mathbb{R}$ are penalties, and λ_1, λ_2 are regularization parameters.

Constraint function $\Omega_1(\mathbf{W})$ is not mandatory in (1) but to shrink the regression coefficients to achieve some regularization effect we use quadratic constraint

$$\Omega_1(\mathbf{W}) = \sum_{t=1}^{T} \|\mathbf{w}^t\|_2^2$$

Table 1. Sparsity inducing constraints

$\Omega_2(\mathbf{W})$	separate	joint
no group	$\sum_{t=1}^{T} \|\mathbf{w}^t\|_0$	$\|\{j \mid \exists i, \mathbf{W}_{i,j} \neq 0\}\|$
group	$\sum_{t=1}^{T} \left\|\{\mathbf{w}^t_{G_i} \neq \mathbf{0}, i = 1, \ldots, J\}\right\|$	$\left\|\{G_i \mid \exists t, \mathbf{w}^t_{G_i} \neq \mathbf{0}\}\right\|$

over all the targets. The choice of constraint function $\Omega_2(\mathbf{W})$ depends on the type of sparsity we wish to enforce, the different alternatives are presented in Table 1. If features are not generated in groups Type I sparsity (1st row, 1st column) or Type II sparsity may be enforced (1st row, 2nd column). If the group structure is known in advance, we may either enforce Type III sparsity (2nd row, 1st column), or Type IV sparsity (2nd row, 2nd column).

3.2 Algorithms

The optimization problem (1) cannot be optimally solved in polynomial time due to the presence of the sparsity constraint Ω_2. However, it has been previously shown that in such settings approximative greedy forward selection procedure can still produce quite good solutions (see e.g. the recent results in [3] on enforcing Type II sparsity). We consider a greedy search heuristic that starts from an empty set of features and on each iteration chooses one additional feature such that provides the lowest mean squared error via cross-validation, when added to the model. The new chosen feature is then added into the current selected feature set and same procedure is continued until budget limit k is reached.

Algorithm 1. greedy separate feature selection

1: **for** $t \in \{1, \ldots, T\}$ **do** ▷ go through all the targets
2: $S \leftarrow \emptyset$ ▷ The set of selected features.
3: **while** $|S| < k$ **do**
4: $b := \text{argmin}_{r \in \{1,\ldots,d\} \setminus \mathcal{S}} \left\{\mathcal{L}(\mathbf{X}^t_{\mathcal{S}\cup\{r\}}, \mathbf{y}^t)\right\}$ ▷ Find the best new feature.
5: $\mathcal{S} \leftarrow \mathcal{S} \cup \{b\}$
6: $\mathbf{w}^t \leftarrow \mathcal{A}(\mathbf{X}^t_{\mathcal{S}}, \mathbf{y}^t)$ ▷ Update coefficients.

Algorithm 1 presents a pseudocode for enforcing Type I sparsity. Inside the **while**-loop, we go through all the remaining features $r \in \{1, \ldots, d\} \setminus \mathcal{S}$ and calculate the mean squared cross-validation error in the procedure denoted by $\mathcal{L}(\cdot)$ that trains predictor for target t using features included in the set $\mathcal{S} \cup \{r\}$. The feature b that returns the lowest error will be selected and added into feature set $\mathcal{S}$. After budget limit k is reached, we train the final model using only the features in the set $\mathcal{S}$ in the procedure denoted by $\mathcal{A}(\cdot)$.

Algorithm 2 selects jointly a set of common features for all the targets, enforcing Type II sparsity. Otherwise it behaves similarly as the Algorithm 1 except

Algorithm 2. greedy joint feature selection

$S \leftarrow \emptyset$ ▷ The set of selected features.
while $|S| < k$ **do**
  $b := \text{argmin}_{r \in \{1,\ldots,d\} \setminus \mathcal{S}} \left\{ \sum_t \mathcal{L}(\mathbf{X}^t_{\mathcal{S} \cup \{r\}}, \mathbf{y}^t) \right\}$ ▷ Find the best new feature.
  $\mathcal{S} \leftarrow \mathcal{S} \cup \{b\}$
$\mathbf{w}^t \leftarrow \mathcal{A}(\mathbf{X}^t_{\mathcal{S}}, \mathbf{y}^t), t = 1, \ldots, T$ ▷ Update coefficients.

Algorithm 3. greedy group separate feature selection

$G_i \subset \{1, \ldots, d\}, i = 1, \ldots, J$ ▷ Init the disjoint feature groups
for $t \in \{1, \ldots, T\}$ **do** ▷ go through all the targets
  $F \leftarrow \emptyset$ ▷ The set of selected feature groups.
  $S \leftarrow \emptyset$ ▷ The set of selected features.
  while $|F| < k$ **do**
    $b := \text{argmin}_{r \in \{1,\ldots,J\} \setminus F} \left\{ \mathcal{L}(\mathbf{X}^t_{\mathcal{S} \cup G_r}, \mathbf{y}^t) \right\}$ ▷ Find the best feature group.
    $F \leftarrow F \cup \{b\}$
    $\mathcal{S} \leftarrow \mathcal{S} \cup G_b$
  $\mathbf{w}^t \leftarrow \mathcal{A}(\mathbf{X}^t_{\mathcal{S}}, \mathbf{y}^t)$ ▷ Update coefficients.

Algorithm 4. greedy group joint feature selection

$G_i \subset \{1, \ldots, d\}, i = 1, \ldots, J$ ▷ Init the disjoint feature groups
$F \leftarrow \emptyset$ ▷ The set of selected feature groups.
$S \leftarrow \emptyset$ ▷ The set of selected features.
while $|F| < k$ **do**
  $b := \text{argmin}_{r \in \{1,\ldots,J\} \setminus F} \left\{ \sum_t \mathcal{L}(\mathbf{X}^t_{\mathcal{S} \cup G_r}, \mathbf{y}^t) \right\}$ ▷ Best new feature group.
  $F \leftarrow F \cup \{b\}$
  $\mathcal{S} \leftarrow \mathcal{S} \cup G_b$
$\mathbf{w}^t \leftarrow \mathcal{A}(\mathbf{X}^t_{\mathcal{S}}, \mathbf{y}^t), t = 1, \ldots, T$ ▷ Update coefficients.

that the mean squared error is calculated over all the targets. Algorithm 3 and Algorithm 4 works similarly than Algorithm 1 and Algorithm 2, respectively, but instead of selecting features they select feature groups.

A straightforward implementation of the Algorithms 1-4 based on black-box solver that would compute the regularized-least squares cross-validation estimate for each tested feature at a time is computationally demanding, as the training needs to be done for each tested feature, target and round of cross-validation. It has recently been shown that both Algorithm 1 [4,5] and Algorithm 2 [3] can be performed in linear time, using linear algebra shortcuts. An interesting direction of future research, that falls outside the scope of this work, would be to extend these speed-ups also for Algorithms 3 and 4.

4 Experiments

In the experiments we study how enforcing different sparsity patterns reduces the prediction costs on two real-world problems.

4.1 Datasets and Setup

We perform our experiments on two real world datasets, the Flags (43 features, 19 feature groups, 7 labels) and the Kddcup10 (118 features, 41 feature groups, 9 labels), available at the Mulan library[1] and at the UCI KDD archive[2], respectively. We modify both data sets to make them suitable for our test settings. First, the Kddcup10 and the Flags data sets are transformed into multi-target data sets by considering each label as its own binary classification task. In order to create feature groups for the setting where we compare group separate feature learning and group joint feature learning schemes, we code each categorical variable as a group of binary features, one for each possible value. Because the Kddcup10 is a quite unbalanced dataset that includes only a few samples for some rare labels, we consider only the 9 most common labels in our experiments.

The experiments are based on nested 5-fold cross-validation [1]. Parameter and feature selection is performed in an inner cross-validation loop, and the final model trained on four folds is always tested on the independent test fold that was not used for feature or parameter selection. Both data sets are tested over regularization parameter values in grid $[2^{-4}, 2^{-2}, 2^{0}, 2^{2}, 2^{4}]$ and the best value for each budget is chosen based on the lowest 4-fold internal cross-validation error. We report the average AUC (area under the ROC curve) over all the targets.

4.2 Results

Fig. 2 contains the average AUC performance curves with respect to costs over all the targets on the Flags and the Kddcup10 datasets. The figures on the first and the second row are based on the assumption that there are no feature groups, whereas the last two rows show the case where the features are divided into groups. Moreover, on the left column we plot the average costs for each target, while on the right column are the joint costs over all the targets.

First, we consider the case where the predictions are done independently for the targets (left column) and extraction costs dominate. This corresponds to the application setting where the models may be jointly trained but the predictions are not done at the same time. In this case better classification performance can be gained with smaller extraction costs by favoring Type I sparsity (or in group case Type III) on the Kddcup10 dataset whereas benefits on the Flags dataset are dependent on the budget size, maybe due to the small sample size of the dataset. Thus, the results support the notion that if the costs are not shared between the targets, then sharing common features might not be beneficial, though apparently it can sometimes still be helpful.

Second, we consider the case where the costs are shared between the targets (right column), either because the extractor costs dominate, or because the predictions are always done jointly, so that the extraction costs are shared. Here, the methods that perform feature selection jointly, enforcing Type II or IV sparsity,

[1] http://mulan.sourceforge.net
[2] http://kdd.ics.uci.edu

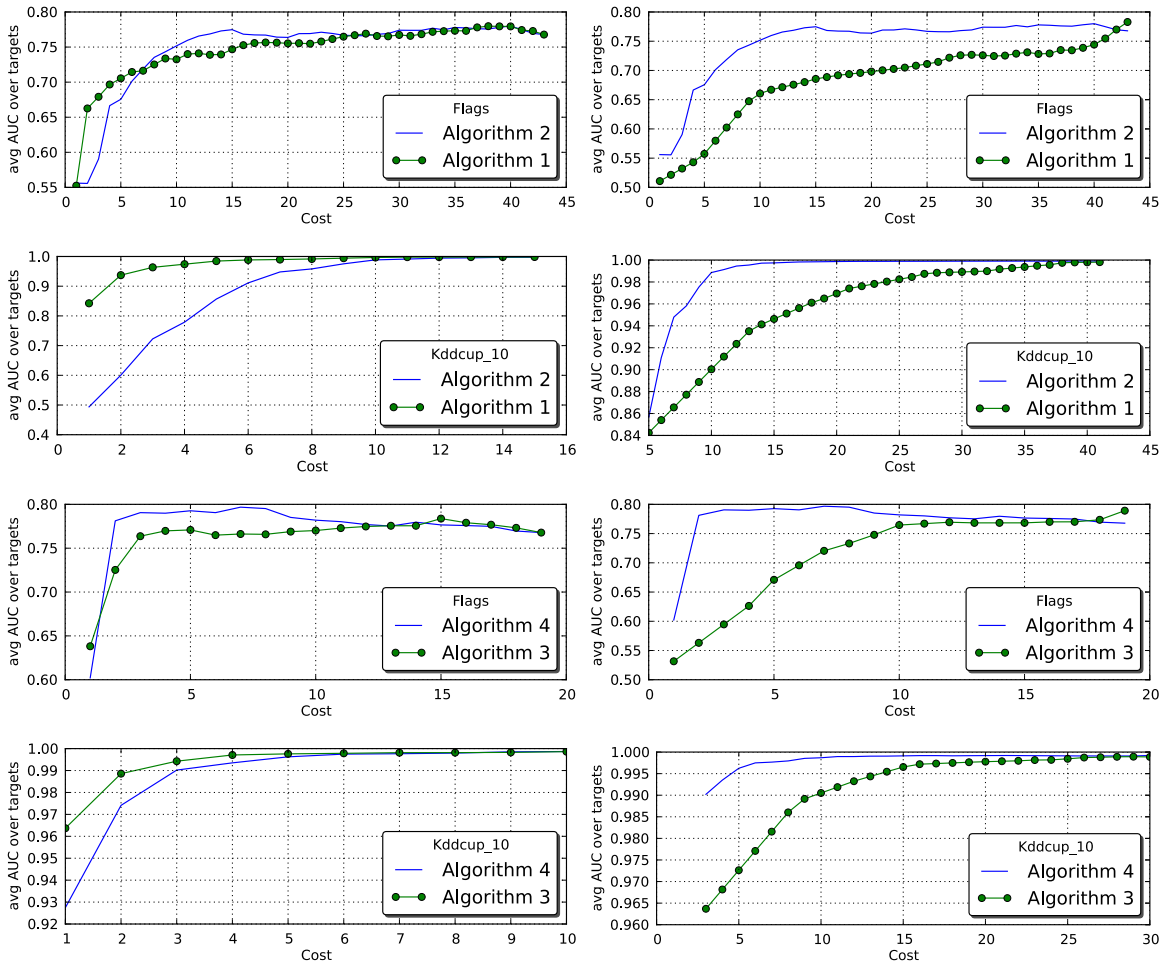

Fig. 2. Performance curves

are clearly beneficial as they allow much higher predictive accuracies especially with small budget sizes. Thus, the results match the intuition behind what sparsity patterns are best suited for which type of setting, though for the Flags data, enforcing joint sparsity proves to be beneficial even if from budget perspective there is not a specific reason why the sharing should be needed.

5 Conclusions and Future Work

In this study, we considered the costs of making predictions for multi-target problems defining four types of sparsity patterns. The results also demonstrate how these patterns relate to extractor and extraction costs resulting in lower prediction costs in different application domains. We presented also greedy forward selection algorithms that enforce such sparsity in the coefficients of the models resulting in lowered prediction costs.

References

1. Ambroise, C., McLachlan, G.J.: Selection bias in gene extraction on the basis of microarray gene-expression data. Proceedings of the National Academy of Sciences of the United States of America **99**(10), 6562–6566 (2002)
2. Kohavi, R., John, G.H.: Wrappers for feature subset selection. Artificial Intelligence **97**, 273–324 (1997)
3. Naula, P., Airola, A., Salakoski, T., Pahikkala, T.: Multi-label learning under feature extraction budgets. Pattern Recognition Letters **40**, 56–65 (2014)
4. Pahikkala, T., Airola, A., Salakoski, T.: Speeding up greedy forward selection for regularized least-squares. In: Draghici, S., Khoshgoftaar, T.M., Palade, V., Pedrycz, W., Wani, M.A., Zhu, X. (eds.) Proceedings of the Ninth International Conference on Machine Learning and Applications (ICMLA 2010), pp. 325–330. IEEE (2010)
5. Pahikkala, T., Okser, S., Airola, A., Salakoski, T., Aittokallio, T.: Wrapper-based selection of genetic features in genome-wide association studies through fast matrix operations. Algorithms for Molecular Biology **7**(1), 11 (2012)
6. Rifkin, R., Klautau, A.: In defense of one-vs-all classification. Journal of Machine Learning Research **5**, 101–141 (2004)
7. Shalev-Shwartz, S., Srebro, N., Zhang, T.: Trading accuracy for sparsity in optimization problems with sparsity constraints. SIAM Journal on Optimization **20**(6), 2807–2832 (2010)
8. Tibshirani, R.: Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society, Series B **58**, 267–288 (1994)
9. Turney, P.D.: Types of cost in inductive concept learning. In: Dietterich, T., Margineantu, D., Provost, F., Turney, P.D. (eds.) Proceedings of the ICML 2000 Workshop on Cost-Sensitive Learning (2000)
10. Yuan, M., Lin, Y.: Model selection and estimation in regression with grouped variables. Journal of the Royal Statistical Society, Series B **68**, 49–67 (2006)
11. Zhang, T.: On the consistency of feature selection using greedy least squares regression. Journal of Machine Learning Research **10**, 555–568 (2009)

Publication VI

Assessment of metal ion concentration in water with structured feature selection

Pekka Naula, Antti Airola, Sari Pihlasalo, Ileana Montoya Perez, Tapio Salakoski, Tapio Pahikkala. Chemosphere, 185:1063–1071, 2017.

Turku Centre for Computer Science
TUCS Dissertations

1. **Marjo Lipponen**, On Primitive Solutions of the Post Correspondence Problem
2. **Timo Käkölä**, Dual Information Systems in Hyperknowledge Organizations
3. **Ville Leppänen**, Studies on the Realization of PRAM
4. **Cunsheng Ding**, Cryptographic Counter Generators
5. **Sami Viitanen**, Some New Global Optimization Algorithms
6. **Tapio Salakoski**, Representative Classification of Protein Structures
7. **Thomas Långbacka**, An Interactive Environment Supporting the Development of Formally Correct Programs
8. **Thomas Finne**, A Decision Support System for Improving Information Security
9. **Valeria Mihalache**, Cooperation, Communication, Control. Investigations on Grammar Systems.
10. **Marina Waldén**, Formal Reasoning About Distributed Algorithms
11. **Tero Laihonen**, Estimates on the Covering Radius When the Dual Distance is Known
12. **Lucian Ilie**, Decision Problems on Orders of Words
13. **Jukkapekka Hekanaho**, An Evolutionary Approach to Concept Learning
14. **Jouni Järvinen**, Knowledge Representation and Rough Sets
15. **Tomi Pasanen**, In-Place Algorithms for Sorting Problems
16. **Mika Johnsson**, Operational and Tactical Level Optimization in Printed Circuit Board Assembly
17. **Mats Aspnäs**, Multiprocessor Architecture and Programming: The Hathi-2 System
18. **Anna Mikhajlova**, Ensuring Correctness of Object and Component Systems
19. **Vesa Torvinen**, Construction and Evaluation of the Labour Game Method
20. **Jorma Boberg**, Cluster Analysis. A Mathematical Approach with Applications to Protein Structures
21. **Leonid Mikhajlov**, Software Reuse Mechanisms and Techniques: Safety Versus Flexibility
22. **Timo Kaukoranta**, Iterative and Hierarchical Methods for Codebook Generation in Vector Quantization
23. **Gábor Magyar**, On Solution Approaches for Some Industrially Motivated Combinatorial Optimization Problems
24. **Linas Laibinis**, Mechanised Formal Reasoning About Modular Programs
25. **Shuhua Liu**, Improving Executive Support in Strategic Scanning with Software Agent Systems
26. **Jaakko Järvi**, New Techniques in Generic Programming – C++ is more Intentional than Intended
27. **Jan-Christian Lehtinen**, Reproducing Kernel Splines in the Analysis of Medical Data
28. **Martin Büchi**, Safe Language Mechanisms for Modularization and Concurrency
29. **Elena Troubitsyna**, Stepwise Development of Dependable Systems
30. **Janne Näppi**, Computer-Assisted Diagnosis of Breast Calcifications
31. **Jianming Liang**, Dynamic Chest Images Analysis
32. **Tiberiu Seceleanu**, Systematic Design of Synchronous Digital Circuits
33. **Tero Aittokallio**, Characterization and Modelling of the Cardiorespiratory System in Sleep-Disordered Breathing
34. **Ivan Porres**, Modeling and Analyzing Software Behavior in UML
35. **Mauno Rönkkö**, Stepwise Development of Hybrid Systems
36. **Jouni Smed**, Production Planning in Printed Circuit Board Assembly
37. **Vesa Halava**, The Post Correspondence Problem for Market Morphisms
38. **Ion Petre**, Commutation Problems on Sets of Words and Formal Power Series
39. **Vladimir Kvassov**, Information Technology and the Productivity of Managerial Work
40. **Frank Tétard**, Managers, Fragmentation of Working Time, and Information Systems

41. **Jan Manuch**, Defect Theorems and Infinite Words
42. **Kalle Ranto**, Z_4-Goethals Codes, Decoding and Designs
43. **Arto Lepistö**, On Relations Between Local and Global Periodicity
44. **Mika Hirvensalo**, Studies on Boolean Functions Related to Quantum Computing
45. **Pentti Virtanen**, Measuring and Improving Component-Based Software Development
46. **Adekunle Okunoye**, Knowledge Management and Global Diversity – A Framework to Support Organisations in Developing Countries
47. **Antonina Kloptchenko**, Text Mining Based on the Prototype Matching Method
48. **Juha Kivijärvi**, Optimization Methods for Clustering
49. **Rimvydas Rukšėnas**, Formal Development of Concurrent Components
50. **Dirk Nowotka**, Periodicity and Unbordered Factors of Words
51. **Attila Gyenesei**, Discovering Frequent Fuzzy Patterns in Relations of Quantitative Attributes
52. **Petteri Kaitovaara**, Packaging of IT Services – Conceptual and Empirical Studies
53. **Petri Rosendahl**, Niho Type Cross-Correlation Functions and Related Equations
54. **Péter Majlender**, A Normative Approach to Possibility Theory and Soft Decision Support
55. **Seppo Virtanen**, A Framework for Rapid Design and Evaluation of Protocol Processors
56. **Tomas Eklund**, The Self-Organizing Map in Financial Benchmarking
57. **Mikael Collan**, Giga-Investments: Modelling the Valuation of Very Large Industrial Real Investments
58. **Dag Björklund**, A Kernel Language for Unified Code Synthesis
59. **Shengnan Han**, Understanding User Adoption of Mobile Technology: Focusing on Physicians in Finland
60. **Irina Georgescu**, Rational Choice and Revealed Preference: A Fuzzy Approach
61. **Ping Yan**, Limit Cycles for Generalized Liénard-Type and Lotka-Volterra Systems
62. **Joonas Lehtinen**, Coding of Wavelet-Transformed Images
63. **Tommi Meskanen,** On the NTRU Cryptosystem
64. **Saeed Salehi**, Varieties of Tree Languages
65. **Jukka Arvo**, Efficient Algorithms for Hardware-Accelerated Shadow Computation
66. **Mika Hirvikorpi**, On the Tactical Level Production Planning in Flexible Manufacturing Systems
67. **Adrian Costea**, Computational Intelligence Methods for Quantitative Data Mining
68. **Cristina Seceleanu**, A Methodology for Constructing Correct Reactive Systems
69. **Luigia Petre**, Modeling with Action Systems
70. **Lu Yan**, Systematic Design of Ubiquitous Systems
71. **Mehran Gomari**, On the Generalization Ability of Bayesian Neural Networks
72. **Ville Harkke**, Knowledge Freedom for Medical Professionals – An Evaluation Study of a Mobile Information System for Physicians in Finland
73. **Marius Cosmin Codrea**, Pattern Analysis of Chlorophyll Fluorescence Signals
74. **Aiying Rong**, Cogeneration Planning Under the Deregulated Power Market and Emissions Trading Scheme
75. **Chihab BenMoussa**, Supporting the Sales Force through Mobile Information and Communication Technologies: Focusing on the Pharmaceutical Sales Force
76. **Jussi Salmi**, Improving Data Analysis in Proteomics
77. **Orieta Celiku**, Mechanized Reasoning for Dually-Nondeterministic and Probabilistic Programs
78. **Kaj-Mikael Björk**, Supply Chain Efficiency with Some Forest Industry Improvements
79. **Viorel Preoteasa**, Program Variables – The Core of Mechanical Reasoning about Imperative Programs
80. **Jonne Poikonen**, Absolute Value Extraction and Order Statistic Filtering for a Mixed-Mode Array Image Processor
81. **Luka Milovanov**, Agile Software Development in an Academic Environment
82. **Francisco Augusto Alcaraz Garcia**, Real Options, Default Risk and Soft Applications
83. **Kai K. Kimppa**, Problems with the Justification of Intellectual Property Rights in Relation to Software and Other Digitally Distributable Media
84. **Dragoş Truşcan**, Model Driven Development of Programmable Architectures
85. **Eugen Czeizler**, The Inverse Neighborhood Problem and Applications of Welch Sets in Automata Theory

86. **Sanna Ranto**, Identifying and Locating-Dominating Codes in Binary Hamming Spaces
87. **Tuomas Hakkarainen**, On the Computation of the Class Numbers of Real Abelian Fields
88. **Elena Czeizler**, Intricacies of Word Equations
89. **Marcus Alanen**, A Metamodeling Framework for Software Engineering
90. **Filip Ginter**, Towards Information Extraction in the Biomedical Domain: Methods and Resources
91. **Jarkko Paavola**, Signature Ensembles and Receiver Structures for Oversaturated Synchronous DS-CDMA Systems
92. **Arho Virkki**, The Human Respiratory System: Modelling, Analysis and Control
93. **Olli Luoma**, Efficient Methods for Storing and Querying XML Data with Relational Databases
94. **Dubravka Ilić**, Formal Reasoning about Dependability in Model-Driven Development
95. **Kim Solin**, Abstract Algebra of Program Refinement
96. **Tomi Westerlund**, Time Aware Modelling and Analysis of Systems-on-Chip
97. **Kalle Saari**, On the Frequency and Periodicity of Infinite Words
98. **Tomi Kärki**, Similarity Relations on Words: Relational Codes and Periods
99. **Markus M. Mäkelä**, Essays on Software Product Development: A Strategic Management Viewpoint
100. **Roope Vehkalahti**, Class Field Theoretic Methods in the Design of Lattice Signal Constellations
101. **Anne-Maria Ernvall-Hytönen**, On Short Exponential Sums Involving Fourier Coefficients of Holomorphic Cusp Forms
102. **Chang Li**, Parallelism and Complexity in Gene Assembly
103. **Tapio Pahikkala**, New Kernel Functions and Learning Methods for Text and Data Mining
104. **Denis Shestakov**, Search Interfaces on the Web: Querying and Characterizing
105. **Sampo Pyysalo**, A Dependency Parsing Approach to Biomedical Text Mining
106. **Anna Sell**, Mobile Digital Calendars in Knowledge Work
107. **Dorina Marghescu**, Evaluating Multidimensional Visualization Techniques in Data Mining Tasks
108. **Tero Säntti**, A Co-Processor Approach for Efficient Java Execution in Embedded Systems
109. **Kari Salonen**, Setup Optimization in High-Mix Surface Mount PCB Assembly
110. **Pontus Boström**, Formal Design and Verification of Systems Using Domain-Specific Languages
111. **Camilla J. Hollanti**, Order-Theoretic Mehtods 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 Commutation 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
145. **Aleksi Saarela**, Word Equations and Related Topics: Independence, Decidability and Characterizations
146. **Lasse Bergroth**, Kahden merkkijonon pisimmän yhteisen alijonon ongelma ja sen ratkaiseminen
147. **Thomas Canhao Xu**, Hardware/Software Co-Design for Multicore Architectures
148. **Tuomas Mäkilä**, Software Development Process Modeling – Developers Perspective to Contemporary Modeling Techniques
149. **Shahrokh Nikou**, Opening the Black-Box of IT Artifacts: Looking into Mobile Service Characteristics and Individual Perception
150. **Alessandro Buoni**, Fraud Detection in the Banking Sector: A Multi-Agent Approach
151. **Mats Neovius**, Trustworthy Context Dependency in Ubiquitous Systems
152. **Fredrik Degerlund**, Scheduling of Guarded Command Based Models
153. **Amir-Mohammad Rahmani-Sane**, Exploration and Design of Power-Efficient Networked Many-Core Systems
154. **Ville Rantala**, On Dynamic Monitoring Methods for Networks-on-Chip
155. **Mikko Pelto**, On Identifying and Locating-Dominating Codes in the Infinite King Grid
156. **Anton Tarasyuk**, Formal Development and Quantitative Verification of Dependable Systems
157. **Muhammad Mohsin Saleemi**, Towards Combining Interactive Mobile TV and Smart Spaces: Architectures, Tools and Application Development
158. **Tommi J. M. Lehtinen**, Numbers and Languages
159. **Peter Sarlin**, Mapping Financial Stability
160. **Alexander Wei Yin**, On Energy Efficient Computing Platforms
161. **Mikołaj Olszewski**, Scaling Up Stepwise Feature Introduction to Construction of Large Software Systems
162. **Maryam Kamali**, Reusable Formal Architectures for Networked Systems
163. **Zhiyuan Yao**, Visual Customer Segmentation and Behavior Analysis – A SOM-Based Approach
164. **Timo Jolivet**, Combinatorics of Pisot Substitutions
165. **Rajeev Kumar Kanth**, Analysis and Life Cycle Assessment of Printed Antennas for Sustainable Wireless Systems
166. **Khalid Latif**, Design Space Exploration for MPSoC Architectures

167. **Bo Yang**, Towards Optimal Application Mapping for Energy-Efficient Many-Core Platforms
168. **Ali Hanzala Khan**, Consistency of UML Based Designs Using Ontology Reasoners
169. **Sonja Leskinen**, m-Equine: IS Support for the Horse Industry
170. **Fareed Ahmed Jokhio**, Video Transcoding in a Distributed Cloud Computing Environment
171. **Moazzam Fareed Niazi**, A Model-Based Development and Verification Framework for Distributed System-on-Chip Architecture
172. **Mari Huova**, Combinatorics on Words: New Aspects on Avoidability, Defect Effect, Equations and Palindromes
173. **Ville Timonen**, Scalable Algorithms for Height Field Illumination
174. **Henri Korvela**, Virtual Communities – A Virtual Treasure Trove for End-User Developers
175. **Kameswar Rao Vaddina**, Thermal-Aware Networked Many-Core Systems
176. **Janne Lahtiranta**, New and Emerging Challenges of the ICT-Mediated Health and Well-Being Services
177. **Irum Rauf**, Design and Validation of Stateful Composite RESTful Web Services
178. **Jari Björne**, Biomedical Event Extraction with Machine Learning
179. **Katri Haverinen**, Natural Language Processing Resources for Finnish: Corpus Development in the General and Clinical Domains
180. **Ville Salo**, Subshifts with Simple Cellular Automata
181. **Johan Ersfolk**, Scheduling Dynamic Dataflow Graphs
182. **Hongyan Liu**, On Advancing Business Intelligence in the Electricity Retail Market
183. **Adnan Ashraf**, Cost-Efficient Virtual Machine Management: Provisioning, Admission Control, and Consolidation
184. **Muhammad Nazrul Islam**, Design and Evaluation of Web Interface Signs to Improve Web Usability: A Semiotic Framework
185. **Johannes Tuikkala**, Algorithmic Techniques in Gene Expression Processing: From Imputation to Visualization
186. **Natalia Díaz Rodríguez**, Semantic and Fuzzy Modelling for Human Behaviour Recognition in Smart Spaces. A Case Study on Ambient Assisted Living
187. **Mikko Pänkäälä**, Potential and Challenges of Analog Reconfigurable Computation in Modern and Future CMOS
188. **Sami Hyrynsalmi**, Letters from the War of Ecosystems – An Analysis of Independent Software Vendors in Mobile Application Marketplaces
189. **Seppo Pulkkinen**, Efficient Optimization Algorithms for Nonlinear Data Analysis
190. **Sami Pyöttiälä**, Optimization and Measuring Techniques for Collect-and-Place Machines in Printed Circuit Board Industry
191. **Syed Mohammad Asad Hassan Jafri**, Virtual Runtime Application Partitions for Resource Management in Massively Parallel Architectures
192. **Toni Ernvall**, On Distributed Storage Codes
193. **Yuliya Prokhorova**, Rigorous Development of Safety-Critical Systems
194. **Olli Lahdenoja**, Local Binary Patterns in Focal-Plane Processing – Analysis and Applications
195. **Annika H. Holmbom**, Visual Analytics for Behavioral and Niche Market Segmentation
196. **Sergey Ostroumov**, Agent-Based Management System for Many-Core Platforms: Rigorous Design and Efficient Implementation
197. **Espen Suenson**, How Computer Programmers Work – Understanding Software Development in Practise
198. **Tuomas Poikela**, Readout Architectures for Hybrid Pixel Detector Readout Chips
199. **Bogdan Iancu**, Quantitative Refinement of Reaction-Based Biomodels
200. **Ilkka Törmä**, Structural and Computational Existence Results for Multidimensional Subshifts
201. **Sebastian Okser**, Scalable Feature Selection Applications for Genome-Wide Association Studies of Complex Diseases
202. **Fredrik Abbors**, Model-Based Testing of Software Systems: Functionality and Performance
203. **Inna Pereverzeva**, Formal Development of Resilient Distributed Systems
204. **Mikhail Barash**, Defining Contexts in Context-Free Grammars
205. **Sepinoud Azimi**, Computational Models for and from Biology: Simple Gene Assembly and Reaction Systems
206. **Petter Sandvik**, Formal Modelling for Digital Media Distribution

207. **Jongyun Moon**, Hydrogen Sensor Application of Anodic Titanium Oxide Nanostructures
208. **Simon Holmbacka**, Energy Aware Software for Many-Core Systems
209. **Charalampos Zinoviadis**, Hierarchy and Expansiveness in Two-Dimensional Subshifts of Finite Type
210. **Mika Murtojärvi**, Efficient Algorithms for Coastal Geographic Problems
211. **Sami Mäkelä**, Cohesion Metrics for Improving Software Quality
212. **Eyal Eshet**, Examining Human-Centered Design Practice in the Mobile Apps Era
213. **Jetro Vesti**, Rich Words and Balanced Words
214. **Jarkko Peltomäki**, Privileged Words and Sturmian Words
215. **Fahimeh Farahnakian**, Energy and Performance Management of Virtual Machines: Provisioning, Placement and Consolidation
216. **Diana-Elena Gratie**, Refinement of Biomodels Using Petri Nets
217. **Harri Merisaari**, Algorithmic Analysis Techniques for Molecular Imaging
218. **Stefan Grönroos**, Efficient and Low-Cost Software Defined Radio on Commodity Hardware
219. **Noora Nieminen**, Garbling Schemes and Applications
220. **Ville Taajamaa**, O-CDIO: Engineering Education Framework with Embedded Design Thinking Methods
221. **Johannes Holvitie**, Technical Debt in Software Development – Examining Premises and Overcoming Implementation for Efficient Management
222. **Tewodros Deneke**, Proactive Management of Video Transcoding Services
223. **Kashif Javed**, Model-Driven Development and Verification of Fault Tolerant Systems
224. **Pekka Naula**, Sparse Predictive Modeling – A Cost-Effective Perspective

Turku
Centre *for*
Computer
Science

http://www. tucs.fi

tucs@abo.fi

University of Turku
Faculty of Mathematics and Natural Sciences
- Department of Information Technology
- Department of Mathematics and Statistics

Turku School of Economics
- Institute of Information Systems Science

Åbo Akademi University
Faculty of Science and Engineering
- Computer Engineering
- Computer Science

Faculty of Social Sciences, Business and Economics
- Information Systems

ISBN 978-952-12-3590-0
ISSN 1239-1883

Pekka Naula

Sparse Predictive Modeling – A Cost-Effective Perspective

Pekka Naula

Sparse Predictive Modeling – A Cost-Effective Perspective