KATHOLIEKE UNIVERSITEIT
LEUVEN

Arenberg Doctoral School of Science, Engineering & Technology
Faculty of Engineering
Department of Computer Science

Algorithms for Multi-target Learning

Beau PICCART

Dissertation presented in partial
fulfillment of the requirements for
the degree of Doctor
in Engineering

June 2012

Algorithms for Multi-target Learning

Beau PICCART

Jury:
Prof. dr. Herman Neuckermans, chair
Prof. dr. Hendrik Blockeel, promotor
Prof. dr. Maurice Bruynooghe
Prof. dr. Marie-Francine Moens, secretary
Prof. dr. Lieven Eeckhout
(Universiteit Gent)
Prof. dr. Stefan Kramer
(Universität Mainz)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Engineering

June 2012

D/2012/7515/72
ISBN 978-94-6018-536-6

Preface

This work contains the results of research conducted at the Department of Computer Science at the University of Leuven. Funding was provided by the FWO, without which this research would not have been possible. I would like to thank the many inspiring people who contributed in the realization of this work. In particular: Hendrik for being an excellent supervisor (and one of the nicest persons I know) throughout the years and the constructive feedback in relation to this and other work. Jan for giving me an excellent kick-start in the PhD-process. Maurice Bruynooghe, Sien Moens and Stefan Kramer for the excellent feedback on my dissertation text. Lieven and the rest of the UGent-group for the fruitful cooperation we had. My colleagues for creating a fun and intellectually stimulating environment. Katrien for the language corrections she made. My mom and dad, without whom I couldn't even have started my studies and for stimulating my curiosity since I was a kid. My sista/homie for the support while writing my text and just for being a great sister. And last but not least, all of my friends!

Abstract

In this work we will investigate multi-target Learning, also called multi-task learning, a branch of Machine Learning. A multi-target learner builds a model which, given some input, returns/predicts multiple variables simultaneously. This stand in contrast with the usual Single-Target model which predicts only one variable. We will see that MT-models have numerous advantages and that many real world applications can benefit from using a multi-target approach.

We observe that when multiple targets are predicted simultaneously by a MT-model, the predictive accuracy might increase in comparison to the single-target model. This is a result of inductive transfer which we discuss in depth in chapter 3. An algorithm is proposed which tries to maximally exploit this effect in order to increase the predictive accuracy of the model for one specific target (the main target).

Next, we focus our attention to a different problem: Collaborative Filtering, a type of recommender system. Such a system tries to predict user preferences, for items such as books or movies, based on already known preferences of the users. We demonstrate how this problem fits the multi-target setting and propose a graph based algorithm to alleviate two common problems inherent to the Collaborative Filtering setting: the Cold Start and the Sparsity problem.

Finally, we propose a new learning setting called Two-Way learning. A specific type of multi-target learning where we try to predict an attribute of a relation between two types of objects. We propose data transposition as an effective method to solve a Two-Way learning problem. This method is applied and studied in depth on the problem of processor performance prediction.

Beknopte samenvatting

Dit werk handelt over multi-target Learning, een tak van Machine Learning. Een multi-target algoritme bouwt een model dat, gegeven een zekere invoer, meerdere variabelen gelijktijdig voorspelt. Dit staat in sterk contrast met het gebruikelijke single-target model dat slechts één variabele voorspelt. We bespreken de voordelen van deze multi-target modellen en zullen zien dat een multi-target aanpak zinvol is voor talrijke toepassingen.

We observeren dat het simultaan voorspellen van meerdere variabelen een positief effect kan hebben op de nauwkeurigheid van de voorspellingen. Dit effect, Inductive Transfer, bespreken we in detail in Hoofdstuk 3. We stellen een algoritme voor dat Inductive Transfer gebruikt om de nauwkeurigheid van ons voorspellingsmodel significant te verbeteren.

Vervolgens bespreken we Collaborative Filtering, een type van aanbevelingssystemen. Zo een systeem tracht de voorkeur van gebruikers te voorspellen voor bijvoorbeeld boeken of films. Dit gebeurt aan de hand van reeds gekende gebruikersvoorkeuren. We tonen aan dat dit een multi-target probleem vormt en ontwikkelen een algoritme dat twee problemen inherent aan aanbevelingssystemen tracht te minimaliseren: het Cold Start en het Sparsity probleem.

Ten slotte stellen we een nieuwe setting voor: Two-Way learning. Dit is een specifiek type multi-target probleem waarbij we een attribuut van een relatie tussen twee types van objecten trachten te voorspellen. We tonen aan hoe deze problemen effectief opgelost kunnen worden door middel van datatranspositie en we passen de voorgestelde methode toe om processorperformantie te voorspellen.

List of abbreviations

AUC	Area Under Curve
CF	Collaborative Filtering
DT	Decision Tree
EAST	Empirical Assymetric Selective Transfer
EMD	Expected Manhattan Distance
GD	Global Distribution
ILP	Inductive Logic Programming
kNN	k-Nearest Neighbor
LR	Linear Regression
MAE	Mean Average Error
MAP	Mean Average Precision
MD	Manhattan Distance
MF	Matrix Factorization
MLP	Multi-Layer Perceptron
MT	Multi-Target
MTL	Multi-Target Learner
MTRT	Multi-Target Regression Tree
NN	Neural Network
PC	Pearson Correlation
PCP	Physico-Chemical Parameters
PGBCF	Probabilistic Graph Based Collaborative Filter
SMO	Sequential Minimal Optimization
SRC	Spearman Rank Correlation
ST	Single-Target
STL	Single-Target Learner
STRT	Single-Target Regression Tree
SVM	Support Vector Machine
Uni	Uniform

Contents

Chapter 1

Introduction

In this work we will investigate Multi-Target learning (MTL), also called multi-task learning, a branch of Machine Learning. We start this introduction by describing classic single-target learning and extend this towards multi-target learning.

A (single-target) machine learning algorithm is concerned with the creation of a model that predicts missing or future data. Take for example a linear regression between two variables A and B. This can be used as a predictive model that maps one variable to the other. The regression is based on known data points, sample instances of variable A for which the corresponding value of variable B is known. In the predictive phase, only the value of A is known and the value of B is predicted by the regression which models the relation between A and B.

In general, the model consists of a function that maps (multiple) input variables to the desired output. The learner builds this function by taking advantage of examples (training data) which allows it to determine the unknown underlying probability distribution of the data. As such, the data is used to illustrate relations between observed variables. These examples cannot cover all possible inputs for the model. Hence, the machine learning algorithm generalizes over the examples in order to predict new instances. This results in the following definition of a machine learning algorithm:

Assume we have a data set S containing pairs $(\mathbf{x}, y)$ with $\mathbf{x} \in X$ the input vector and $y \in Y$ the corresponding target value.

A learner A_{ST} learns from a data set $S = \{(\mathbf{x}, y)\}$ a function $f_i : X \to Y$ such that $\sum_{(\mathbf{x},y)\in S} L_i(f_i(\mathbf{x}), y)$ is minimized, with L_i some loss function over Y.

Table 1.1: Training data for the Play Tennis problem.

ID	Outlook	Temperature	Humidity	Wind	Tennis
1	Sunny	High	Low	Low	YES
2	Overcast	High	Low	Low	YES
3	Sunny	Low	High	High	NO
4	Overcast	Low	Low	Low	YES
5	Overcast	High	Low	High	NO

Numerous such learning algorithms have been developed. Each with its own strengths and weaknesses. No single algorithm is able to outperform all others (Radcliffe and Surry 1995).

Multi-target learning: an example

We demonstrate these concepts with a decision tree learner which builds a model for the play-tennis problem. The goal is to build a model, a decision tree with this learner, which predicts whether a person will play tennis or not given a number of conditions like the current temperature, humidity, etcetera.

For this problem, the training data consists of the data in Table 1.1. Each row describes one training example. The variables *Outlook*, *Temperature*, *Humidity* and *Wind* are the input-variables. *Tennis* is the output or target-variable. This is the variable which the model has to predict for an unseen example which contains only the input-variables.

Figure 1.1 gives a possible decision tree (DT) model for the play tennis model. To evaluate the tree on a new example, one answers each question until an end-node is reached. For example: the model predicts a person would not play tennis on an overcast and humid day, regardless of the temperature or wind conditions.

This model predicts only one variable: the *Tennis* target variable. Therefore we call this a single-target model. Suppose we want to predict if a person will read a book, besides predicting whether he will play tennis or not. The training data might look like the data in table 1.2

The usual approach to this problem is to built two separate models: one which predicts *Tennis* and another one predicting *Book*. The alternative to this approach is to build one *multi-target* model which predicts both variables

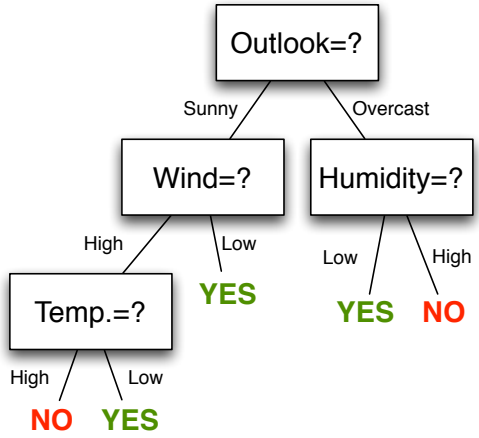


Figure 1.1: A single-target decision tree model for the play tennis problem.

Table 1.2: Training data for the *Play Tennis / Read Book* problem.

ID	Outlook	Temperature	Humidity	Wind	Tennis	Book
1	Sunny	High	Low	Low	YES	NO
2	Overcast	High	Low	Low	YES	NO
3	Sunny	Low	High	High	NO	YES
4	Overcast	Low	Low	Low	YES	YES
5	Overcast	High	Low	High	NO	YES

simultaneously. Figure 1.2 shows such a model, a multi-target decision tree in this case.

We formally define multi-target learning in Section 2.1.1. These multi-target models have a number of benefits over single-target models. They can lead to more accurate models due to their ability to exploit potential dependencies among the target variables, they can be faster to learn, lead to faster prediction times and have the ability to show relations between the different target variables [Caruana 1993].

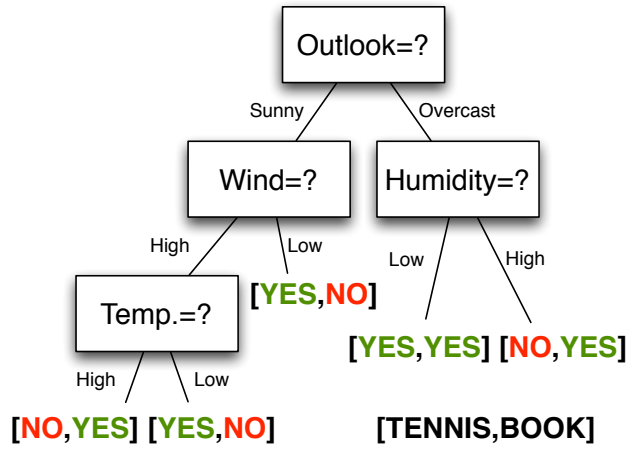


Figure 1.2: A multi-target decision tree model for the *play tennis / read book* problem.

Applications

Multi-target prediction is encountered for instance in ecological modelling where the domain expert is interested in (simultaneously) predicting the frequencies of different organisms in river water (Blockeel, Džeroski, and Grbović 1999) or agricultural soil (Demšar et al. 2006). It can also be applied in multi-label classification tasks (Clare and King 2001), where a set of labels is to be predicted for each instance instead of a single label; in prediction tasks with a structured output space (Tsochantaridis et al. 2005), such as hierarchical multi-label classification (Blockeel et al. 2006), where the output is structured as a taxonomy of labels (e.g., newsgroups); and in multi-task or transfer learning, where knowledge gained from learning one task is reused to better learn related tasks (Caruana 1997a). Chapter 4 discusses how recommender systems can be build using multi-target learners. The following chapter introduces 2Way learning, a multi-target settings which fits a number of problems, including processor performance prediction which we will discuss in detail in Chapter 6.

Contributions and structure of this text

Chapter 2 formally defines multi-target learning and gives an overview of existing multi-target algorithms.

Subsequent chapters discuss a number of different problems related to Multi-Target Learning to which we propose effective solutions.

Chapter 3 discusses inductive transfer in multi-target algorithms. We show that a multi-target model is able to outperform a set of single-target models in which each model predicts a different target. This is a consequence of inductive transfer. The different targets are related, resulting in additional information available to the learner. This brings us to the question of how to maximally exploit this additional information. We propose an algorithm called EAST (Piccart, Struyf, and Blockeel 2008a; Piccart, Struyf, and Blockeel 2008b) which tries to maximally exploit this additional information.

In Chapter 4 we demonstrate how Recommender Systems(Adomavicius and Tuzhilin 2005b) can be build using a multi-target learner for Collaborative Filtering (CF). Such a system recommends users items of a certain type, for example books or movies. The dataset consists of the item selection history (often purchases) of all users. The predictive model takes as input variables the previously selected items by the user for which we're making a recommendation. The output variables correspond to al the possible items. The variables are binary: a positive prediction means we include the item in the recommendation, a negative outcome means the item is not recommended for that user. Multi-target algorithms are very suitable for this problem given the often very large number of items/output variables. Straight application of multi-target algorithms to the recommender system setting does not result in great performance. The setting has some properties which makes effective generalization difficult; notably the sparsity of the dataset. We develop a number of methods (Piccart, Blockeel, and Struyf 2010; Piccart et al. 2010) which alleviate the problem of data sparsity in Collaborative Filtering in Chapter 4.

Chapter 5 introduces a new setting: Two-Way learning. This class of problems consists of learning the relation between two types of objects. Many problems fit this setting, including the previously discussed recommender system. The Two-Way setting can be reduced to a normal Multi-Target setting but this would disregard the special structure of the problem. The following question arises: is it possible to exploit the special structure of a Two-Way problem in order to build a more accurate predictive model. We propose a method (Piccart et al. 2012) which is able to do so and can greatly improve the predictive accuracy of multi-target algorithms in this setting, yet is very simple to implement. The method is applied to two diverse applications and its behaviour is analysed in depth.

Finally, in Chapter 6, we apply the previously proposed methods to a practical application. The application consists of predicting the run-time speed of a certain application of interest on a large number of different machine architectures.

Hence, each machine for which we want to predict the run-time of our application corresponds to a target. We use the measured performance on a small set of (predictive) machines to predict the performance of our application on each machine in the target set. This leads to the problem selecting which machines to include in the set of predictive machines. We propose a solution (Piccart et al. 2011) which improves significantly over the current state-of-the-art. The method selects a given number of machines which are as predictive as possible. Furthermore, we demonstrate that a small number of predictive machines is sufficient to accurately predict the performance of the application of interest on the target machines.

Chapter 2

Multi-target models

2.1 Multi-target models

The majority of single-target learning algorithms has been extended towards multi-target algorithms. In this section we will give a formal definition of multi-target models followed by an overview of existing multi-target approaches.

2.1.1 Definition

Assume we have a data set S containing couples $(\mathbf{x}, \mathbf{y})$ with $\mathbf{x} \in X$ the input vector and $\mathbf{y} \in Y = Y_1 \times \cdots \times Y_n$ the target vector. Denote with $y_i \in Y_i$ the i'th component of $\mathbf{y}$.

A single-target learner A_{ST} learns from a data set $S = \{(\mathbf{x}, y_i)\}$, with $y_i \in Y_i$ a scalar variable, a function $f_i : X \to Y_i$ such that $\sum_{(\mathbf{x},y_i)\in S} L_i(f_i(\mathbf{x}), y_i)$ is minimized, with L_i some loss function over Y_i.

A multi-target learner A_{MT} learns from a data set $S = \{(\mathbf{x}, \mathbf{y})\}$, with $\mathbf{y} \in Y$ an n-dimensional vector, a function $F : X \to Y$ such that $\sum_{(\mathbf{x},\mathbf{y})\in S} L(F(\mathbf{x}), \mathbf{y})$ is minimized, with L a loss function over Y. In the following, we will use $L(\mathbf{y}, \mathbf{y}') = \sum_i L_i(y_i, y_i')$.

We say that the multi-target approach gives better predictive performance than the single-target approach if for any $(\mathbf{x}, \mathbf{y})$, drawn randomly from the population, on average, $L(F(\mathbf{x}), \mathbf{y}) < \sum_i L_i(f_i(\mathbf{x}), y_i)$.

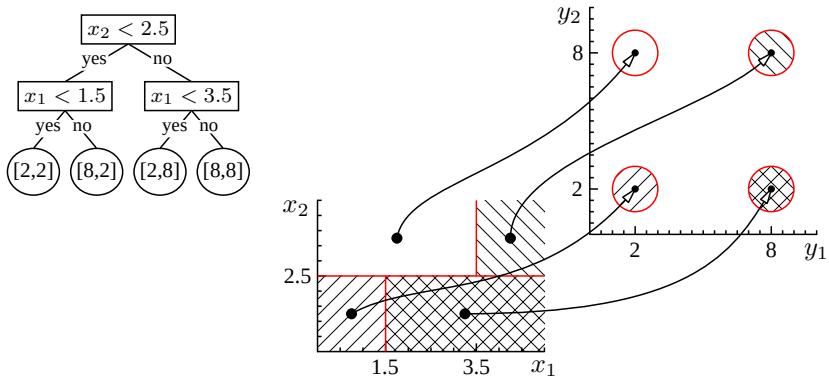


Figure 2.1: A multi-target regression tree together with its mapping from the input to the target space.

2.1.2 Multi-Target Decision Trees

Most descriptions of decision tree learning assume a single target, which may be nominal (classification) or numerical (regression). Blockeel, De Raedt, and Ramon (1998) argued that decision tree learning can easily be extended towards the case of multi-target prediction, by extending the notion of class entropy or variance towards the multi-dimensional case. They define the variance of a set as the mean squared distance between any element of a set and a centroid of the set. Depending on the definition of distance, which could be Euclidean distance in a multidimensional target space, a decision tree will be built that gives accurate predictions for multiple target variables.

Fig. 2.1 shows an example of a decision tree with two numeric target variables. It also illustrates that multi-target regression trees work well if the training data maps hyper-rectangles in the input space to compact clusters in the target space.

Similar to other multi-target models, it has been shown that multi-target trees can be more accurate than single-target trees. This holds for both multi-label classification trees (Blockeel et al. 2006) and for multi-objective regression trees (Struyf and Džeroski 2006).

Suzuki, Gotoh, and Choki (2001) introduced bloomy decision trees. Flower nodes are added to the tree, giving predictions for the different targets. The nodes appear not only at the fringe of a tree but also inside a tree. The method has been implemented as D3-B (Decision tree in Bloom), and experimentally demonstrated to outperform C4.5.

Multi-variate regression trees were introduced by Sain and Carmack (2001) . In such a tree, the distribution of the target variables is assumed to be a mixture distribution. No comparison to other methods was made.

2.1.3 Neural Networks

Richard Caruana et al. introduced a number of multi-target (multi-task) neural network approaches. Caruana (1993) presented the first MT neural network. They argue that generalization is improved by leveraging the domain-specific information in training-signals of related tasks and coined the term multitask inductive transfer. From the point of view of the main task, the other tasks may serve as bias. An experimental validation demonstrated the multi-target model outperforming the single-target model 8 out of 10 times. We study inductive transfer in detail in Chapter 3 and propose a method to maximally exploit this inductive transfer.

The proposed neural network uses a shared hidden layer for all targets but target-specific weights at the output layer. The hidden layer can be interpreted as a form of feature selection or domain representation for all targets. The hidden layer thus learns the features common to all tasks. This may lead to improved performance as the learner can use the information available for all targets, enabling targets with limited available data to benefit from the data available for other related targets. This approach creates a representation bias towards the intersection of what would be learned by the individual tasks. Caruana et al. argue that we may never find an answer to the question "what are related tasks?" and that the only solution to find out which tasks/targets are beneficial to each other is empirical validation. This is the approach we take in Chapter 3.

Silver and Mercer (1996) introduces a method for the parallel transfer of task knowledge using dynamic learning rates. This method improves upon the work of Caruana et al. The ηMTL algorithm is introduced which uses a higher learning speed for targets which are more related. Relatedness between the targets is based on the "weight space distance", a dynamic measure. The method performs as well as standard MTL when the targets are related and is able to outperform MTL when the set of targets contains one or more unrelated targets, in relation to the main target.

In Silver and Mercer (2001), a number of distance measures are evaluated using the ηMTL algorithm. These include the dynamic "weight space distance", static measures like linear correlation and mutual information, and hybrid combinations of these relatedness measures. The hybrid measures performs the best, followed by the static measures. The dynamic measure performs the worst,

presumably because it is not a well-behaved proper distance metric. Correlation outperforms mutual information, most likely because the outcome lies between -1 and 1. The static measures tend to perform worse when the output is uncorrelated while the input is correlated. Note that all these measures are symmetric: hence the inductive transfer is assumed to be symmetric as well. That is, when target A is beneficial towards target B, i.e. they are related, it is assumed that target B is equally beneficial towards A. We will see in Chapter 3 that this rarely holds. This observation led us to developed an asymmetric empirical measure of relatedness which forms the basis of our algorithm proposed in Section 3.3.

Ghosn and Bengio (2003) proposes bias learning, a knowledge sharing method. The approach is based on Affine Manifold Learning, a generalization of the Nearest Neighbor method. The experimental results demonstrate that the approach performs similar to the one proposed by Caruana. They observe that the performance difference between a MTL and an STL method decreases with an increase of the dataset size.

2.1.4 Kernel Methods

A number of kernel methods has been proposed, mostly adaptations of Support Vector Machines (SVM).

Evgeniou and Pontil (2004) proposed Regularized MTL. The adaptation to MTL is straightforward. The method is equal or better to the STL model up to 30 targets but performance degrades significantly above 100 targets. No comparison to other MTL methods was made.

Trafalis and Oladunni (2005) proposes an alternative MTL-SVM algorithm. The rationale behind its performance increase, with relation to STL, is based on the assumption that the error terms, i.e. noise, for each target are correlated and that learning these targets together tends to cancel out noise.

2.1.5 Nearest Neighbor

Nearest Neighbor methods can be adapted to MTL by using a weighted distance measure with weights chosen to minimize the error for all targets in the training set. Thrun and O'Sullivan (1996) performed a number of experiments around clustering tasks which were then predicted with such a Nearest Neighbor method for each cluster. Relatedness,inductive transfer, was measured empirically by constructing a matrix of all target pairs, similar to our approach in Section 3.2.1. Subsequently, the targets are clustered according to the data in this matrix.

This approach assumes the transfer to be symmetric, even though the data presented by Thrun and O'Sullivan (1996) shows the matrix to be asymmetric.

2.1.6 Multi-Label classification

A multi-label classifier tries to assign to each instance a subset of a set of possible labels. Such a classifier can be constructed using a multi-target with a binary output target for each possible label. Hence, the field of multi-label classification is closely related to the field of multi-target learning. The multi-label learner tries to exploit relations among the labels in order to improve the predictive performance of the model (Wicker, Pfahringer, and Kramer 2012), similar to how a multi-target model exploits relations among the different targets. A multitude of multi-label classifiers has been proposed. These range from applications in text mining (Read et al. 2009; Godbole and Sarawagi 2004; Fürnkranz et al. 2008), to multi-media classification (Boutell et al. 2004; Dimou et al. 2009) and bioinformatics (Elisseeff and Weston 2005; Vens et al. 2008) .

Chapter 3

Inductive transfer in multi-target models

We observe that the predictive accuracy of a multi-target model (a) can outperform the accuracy of a single-target model (b). That is, for a given target, either (a) or (b) can yield the most accurate model. The multi-target model is able to exploit potential dependencies among the targets. This is a form of inductive transfer between the different targets which may be beneficial as well as detrimental to accuracy.

In this chapter we will discuss this transfer in depth and propose an algorithm to maximize the inductive transfer towards a specific target.

3.1 The accuracy of multi-target models

It has been shown that multi-target models, which predict multiple target variables simultaneously, can be more accurate than predicting each target individually with a separate single-target model (Caruana 1997a). This is a consequence of the fact that when the targets are related, they can carry information about each other; the single-target approach is unable to exploit that information, while multi-target models naturally exploit it. This effect is known as inductive transfer: the information a target carries about the other targets is transferred to those other targets. Note the connection with collective classification (Jensen, Neville, and Gallagher 2004): the latter exploits

dependencies among targets of different instances, while multi-target models exploit dependencies among the multiple targets of the same instance.

Multi-target models do, however, not always lead to more accurate prediction. As we will show, for a given target variable, the variable's single-target model may be more accurate than the multi-target model. That is, inductive transfer from other variables can be beneficial for one target, but it may also be detrimental to the accuracy of another target. Let us focus on one particular target and call this the main target. The subset of targets that, when combined with the main target in a multi-target model, results in the most accurate model for the main target, may be non-trivial, i.e., different from the empty set and from the set of all targets. We call this set the optimal support set for the main target. In this chapter we investigate how to best approximate this set. Note that the two natural extremes of this approach are the single-target model (the support set is empty) and the full multi-target model (the support set includes all targets).

3.2 Problem Setting

Under the monotonicity assumption given in 2.1, obtaining better predictive performance on average implies that there must be individual targets for which the predictive performance, as measured on this single target, must improve. That is, there must be at least one i for which $L_i(F_i(\mathbf{x}), y_i) < L_i(f_i(\mathbf{x}), y_i)$, with $F_i(\mathbf{x})$ the i'th component of $F(\mathbf{x})$.

This observation leads to the question whether single-target models could be improved by following the multi-target approach. That is: even when there is only one single target that we want to predict, we may be able to build a better model for predicting that target if we can exploit the information present in other, related, variables.

Thus, the problem setting becomes as follows. We are given a training set $S = \{(\mathbf{x}, \mathbf{y})\}$, and are interested in predicting y_n from $\mathbf{x}$. The variables $y_i, i \neq n$ need not be predicted, and will not be available at prediction time, but we can use them during the learning phase. We call this setting the single-target setting with support targets: one single target y_n (the main target) needs to be predicted, but a number of additional support targets $y_i, i \neq n$ are available at induction time, and can be used to improve the model for y_n.

One approach to tackle the single-target with support targets setting is to build a multi-target model F for $y_1, \ldots, y_n$ and project that model on the main target, that is, return as single-target model for y_n the model $F_n(\mathbf{x})$. We here propose

a more refined approach that, as we will see, tries to identify the "best" subset of support targets.

Note the following important point: while the support targets are used during learning, they are not assumed to be available during the prediction phase. If they were, then an alternative to the method proposed here would be to learn a model that also uses the support targets as inputs (e.g., in the case of decision tree learning, to learn a tree that is allowed to test these attributes). It is quite likely that that would lead to better prediction. The model that we will try to learn here, is one that will make predictions without having that information; we use the support targets to learn a model that maps $\mathbf{x}$ onto a target attribute more accurately, even though the model itself has no access to the support targets. This is a form of inductive transfer, or transfer learning. We apply knowledge gained while solving one problem (the support targets) and apply it to a different but related problem in order to increase the predictive accuracy.

3.2.1 The Transfer Matrix

To gain insight in the effect of applying multi-target models to single-target prediction, we construct a transfer matrix (Thrun and O'Sullivan 1996) $C = (c_{i,j})$, where $c_{i,j}$ is the expected gain in predictive performance for target y_i that the two-target model with targets y_i and y_j yields over the single-target model for y_i. In other words, $c_{i,j}$ indicates if inductive transfer from y_j to y_i is useful. In the following, we call $c_{i,j}$ the transfer from y_j to y_i.

Table 3.1 shows the transfer matrix for one of the data sets that we will use in the experimental evaluation. The multi- and single-target models on which the matrix is based are regression trees. We use Pearson Correlation as our performance measure. It's often used in regression problems and defined as:

$$pc(\mathbf{p}, \mathbf{q}) = \frac{\sum_i (p_i - \overline{p})(q_i - \overline{q})}{\sqrt{\sum_i (p_i - \overline{p})^2 \sum_i (q_i - \overline{q})^2}} \quad ,$$

in which $\overline{u}$ denotes the average of u's components.

The matrix elements are the relative differences in 10-fold cross-validated Pearson correlation between the multi- and single-target model for y_i, averaged over 5 runs with different random folds; each row corresponds to a different support target y_j in the multi-target model.

From the transfer matrix, we see that transfer is an asymmetric quantity: it is possible that y_i can be predicted more accurately if y_j is included as support

Table 3.1: Transfer matrix for the Soil Quality 1, Collembola groups data set (a subset of S_2, see experimental setup).

$j \setminus i$	1	2	3	4	5
1	0	0.01	0.04	0.13	0
2	0	0	0.06	0.13	0
3	0.07	0.01	0	*0.13*	0
4	0.03	0	*-0.02*	0	*-0.01*
5	0.1	0.03	0.31	*0.18*	0

Table 3.2: Correlation matrix for Soil Quality 1, Collembola groups.

$j \setminus i$	1	2	3	4	5
1	1	0.03	0.12	0.05	0.07
2	0.03	1	0.08	0.27	0.45
3	0.12	0.08	1	0.31	0.14
4	0.05	0.27	0.31	1	0.19
5	0.07	0.45	0.14	0.19	1

target ($c_{i,j} > 0$), whereas the prediction for y_j actually deteriorates when y_i is included ($c_{j,i} < 0$).

While empirically measuring transfer as defined above is the most direct way of deciding which support target to use for predicting a given target, two important approximations to this approach have been used in previous work: (1) converting transfer into a symmetric quantity (Thrun and O'Sullivan 1996), and (2) using other measures, such as correlation, to somehow approximate transfer (Silver and Mercer 2001). The advantage of such approximations is that they reduce the computational cost of the approach. For example, by combining (1) and (2), one could use the pairwise linear correlation between the targets to cluster the targets and then build a multi-target model for each cluster in the partition.

The disadvantage of (1) is that, because transfer is really asymmetric, replacing it by a symmetric approximation will result in suboptimal models for certain targets. Consider again Table 3.1. Clustering y_3 and y_4 together will result in a suboptimal model for y_3, while putting y_3 and y_4 in different multi-target models may result in a suboptimal model for y_4.

The disadvantage of (2) is that correlation is often not a good approximation for transfer or relatedness. Table 3.2 shows the targets' correlation matrix for our

example. y_2 and y_5 have the highest correlation. Nevertheless, the transfer from y_2 to y_5 is zero and that from y_5 and y_2 is small. One reason is that transfer does not only depend on the values of the target variables. It also depends on other factors, such as the mapping that the data represents between input and output. Measures that only depend on the targets, may therefore approximate transfer poorly. Fig. 2.1 illustrates this: the correlation between the targets is zero, yet the multi-target regression tree predicts both targets accurately.

These two disadvantages are alleviated by our approach, which we discuss next.

3.3 Selective Inductive Transfer

Since addition of extra support targets may increase the predictive accuracy for the main target, but is not guaranteed to do so, and moreover some target variables may help while others are detrimental, we can consider the following procedure: add extra target variables one by one, always selecting that target variable that appears to give the largest improvement; and continue doing this until it does not result in any further improvement.

There is the question of how we can select the "best" target to add to our current set of support targets. As explained before, any measure which is symmetric or takes only relations among the target values into account will be suboptimal. Therefore, we directly measure the increase in predictive performance that a candidate support target yields using (internal) cross-validation. This takes into account all possible effects of including a certain support target.

Our "Empirical Asymmetric Selective Transfer (EAST)" procedure is described in Algorithm 1. It essentially implements the approach outlined above. The internal loop finds the next best candidate support target that can be added to the multi-target model. To do so, it calls the procedure cross-validate(T, L_n, S), which computes the 10-fold cross-validated loss L_n with regard to the main target y_n of a multi-target model with targets T constructed from S. The outer loop repeats this process until no candidate support target further improves predictive performance. The method is depicted in figure 3.1.

The computational cost of EAST compares as follows to building a single-target regression tree. If T_{ST} is the execution time required for building a single-target tree, then iteration i of EAST's outer loop costs $9 \cdot T_{\mathrm{ST}} \cdot (i+1) \cdot (n-i)$, because it tries $(n-i)$ candidate support targets and cross-validates for each candidate a $(i+1)$-target tree. Building one single $(i+1)$-target tree costs $\approx T_{\mathrm{ST}} \cdot (i+1)$; cross-validating it costs 9 times more (10 folds, each with a training set of $0.9 \cdot |S|$). m iterations of EAST therefore cost $9 \cdot T_{\mathrm{ST}} \sum_{i=1}^{m} (i+1)(n-i)$,

Algorithm 1 Empirical Asymmetric Selective Transfer (EAST).

input: data set S, main target $t = y_n$,
support targets $T_s := \{y_i \mid i \neq n\}$.
$T := \{t\}$
$L^* :=$ cross-validate(T, L_n, S)
repeat
  found := *false*
  for each $t_s \in (T_s - T)$ **do**
    $L :=$ cross-validate($T \cup \{t_s\}$, L_n, S)
    if $L < L^*$ **then**
      $L^* := L$; *found* := *true*; $t_s^* := t_s$
    end if
  end for
  if *found* **then**
    $T := T \cup \{t_s^*\}$
  end if
until not *found*
return induce(T, S)

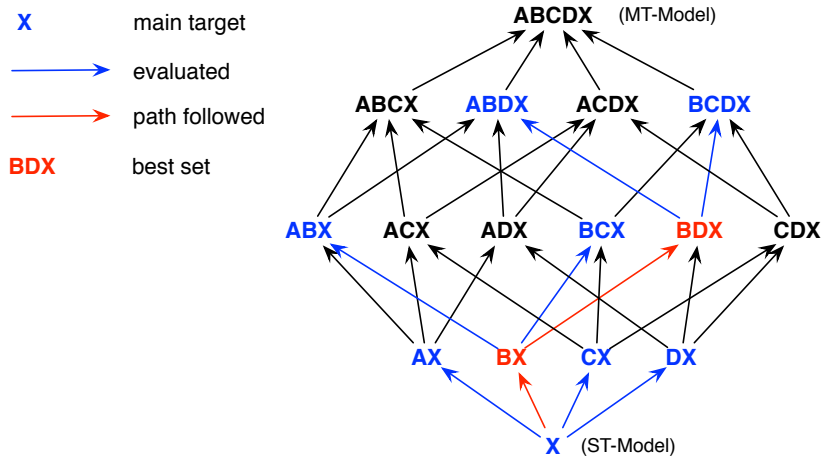


Figure 3.1: The EAST procedure.

i.e., EAST is a factor $O(nm^2/2 - m^3/3)$ slower than building a single-target regression tree. For example, for a data set with 10 candidate support targets, of which EAST selects 5, EAST will be roughly 150 times slower.

In our experiments, EAST's runtime proved to be acceptable because T_{ST} was relatively small (e.g., less than one minute). For large data sets, an alternative is to replace the cross-validation in EAST's internal loop by a single train/test split. This would make it about a factor 10 faster.

3.4 Experimental Evaluation

The aim of our experiments is to test to which extent EAST, for a given main target, succeeds in finding a good set of support targets.

Comparing the accuracy of EAST to the accuracy of the multi-target model with the best subset of support targets is difficult because computing the best subset would require us to enumerate all possible subsets (which is intractable for large n) and to compute the predictive accuracy of the multi-target model corresponding to a each subset exactly. Therefore, we compare EAST to two common baseline models instead: a single-target model for the main target (ST), and a multi-target model that includes all targets (MT).

Note that EAST is identical to ST if the set of selected support targets is empty, and identical to MT if the set includes all available targets. From this, it follows that if EAST successfully finds the best set of support targets, the following must hold:

1. EAST must be at least as accurate as ST. If EAST is more accurate than ST, then it selects a set of support targets that is beneficial to the main target's accuracy. Conversely, if EAST is less accurate than ST, then it incorrectly selects a set of support targets that is detrimental to the main target's accuracy.
2. EAST must be at least as accurate as MT. If EAST is more accurate than MT, then it successfully excludes a set of targets that is detrimental to the main target's accuracy. On the other hand, if EAST is less accurate than MT, then EAST selects too few support targets or a suboptimal subset of the targets.

3.4.1 Data Sets

The data sets that we use are listed, together with their properties, in Table 3.3. Most data sets are of ecological nature. We omit the description of each data set; the interested reader can find details in the following publications: Demšar et al. 2005 for S_1, Demšar et al. 2006 for S_2 and S_3, Kampichler, Džeroski, and

Table 3.3: Data set properties. N is the number of examples, $|\mathbf{x}|$ the number of input variables, and $|\mathbf{y}|$ is the number of target variables.

	Domain/Task	N	$\|\mathbf{x}\|$	$\|\mathbf{y}\|$
S_1	Sigmea Real	817	4	2
	Soil Quality 1	1944	139	
S_2	Acari/Coll. groups	”	”	9
S_3	Coll. species	”	”	39
S_4	Soil Quality 2	393	48	3
	Water quality	1060		
S_5	Plants/Animals	”	16	14
S_6	Chemical	”	836	16

Wieland 2000 for S_4, and Blockeel, Džeroski, and Grbović 1999 for S_5 and S_6. Each data set represents a multi-target regression problem and the number of target variables varies from 2 to 39.

3.4.2 Experimental Procedure

EAST has been implemented in decision tree induction system *Clus*, which is available as open source software from `http://dtai.cs.kuleuven.be/clus/`. *Clus* also implements single- and multi-target regression trees, so all results that we present next are obtained with the same system and parameter settings. All parameters are set to their default values. To avoid overfitting we prune the trees using CART validation set based pruning, i.e., we use 70% training data for tree building and 30% for pruning (as suggested by Torgo 1998). We normalize each target variable to zero mean and unit variance.

We compare for each data set and target variable, the predictive performance of a traditional single-target regression tree (STRT), a tree constructed by EAST with all other targets as candidate support targets, and a multi-target tree including all targets (MTRT). We estimate predictive performance as the 10-fold cross-validated Pearson correlation, averaged over five runs. We report average correlation together with the standard deviation of the individual estimates. To compare the algorithms on a given target or data set, we use the corrected resampled t-test statistic with significance level 0.01 (as implemented in Weka).

Table 3.4: Cross-validated Pearson correlation averaged over all targets. •,○ denote a statistically significant improvement or degradation of EAST or MTRT over STRT. ■,□ denote a statistically significant improvement or degradation of EAST over MTRT.

Data set	STRT	EAST	MTRT
S_1	0.63±0.40	0.64±0.40	0.64±0.40
S_2	0.60±0.18	0.63±0.13•	0.64±0.13•
S_3	0.34±0.40	0.41±0.35•□	0.48±0.29•
S_4	0.19±0.23	0.24±0.23•	0.26±0.22•
S_5	0.26±0.17	0.29±0.15•■	0.27±0.15
S_6	0.37±0.27	0.41±0.25•■	0.39±0.23

3.4.3 Results & Discussion

Table 3.4 shows for each method, cross-validated Pearson correlation, averaged over all targets. EAST performs significantly better than STRT for 5 out of 6 data sets and never performs significantly worse. MTRT, on the other hand, only significantly outperforms STRT in 3 out of 6 data sets. On the two data sets where only EAST is significantly better than STRT, it is also significantly better than MTRT. For one data set (S_3), EAST still significantly outperforms STRT, yet it is also significantly worse than MTRT. We discuss S_3 in more detail below.

The above results support that EAST, for a given target, is able to select a good subset of the other targets as support targets. For 5/6 data sets it finds on average subsets that are better than selecting the empty subset, i.e., STRT. Also looking at individual targets (Table 3.5) shows that EAST selects helpful support targets: for 18 targets it finds a subset that is significantly better than the empty set; for two it selects a subset that is worse (one that actually decreases predictive performance).

For S_5 and S_6 EAST finds on average a subset that works better than using all targets, i.e., MTRT. This can also be seen at the level of individual targets (Table 3.5): EAST performs significantly better than MTRT on 8 and significantly worse on 3 of the targets of S_5 and S_6.

Table 3.6 shows detailed results for S_5. These results are consistent with our earlier observations. EAST performs always at least as good as STRT and is significantly better on four targets. The same does not hold for MTRT, which does significantly worse than STRT on four targets. So, there are clearly targets where using all other targets as support targets is suboptimal. For these targets,

Table 3.5: Pairwise comparison of methods. For each pair A/B, the number of targets are given that represent significant wins or losses for A when compared to B.

Data set	EAST/STRT		EAST/MTRT		MTRT/STRT	
	#win	#loss	#win	#loss	#win	#loss
S_1	1	0	0	0	1	0
S_2	1	0	0	0	2	0
S_3	7	1	0	11	23	1
S_4	1	0	0	0	1	0
S_5	4	0	4	2	4	4
S_6	4	1	4	1	4	4

Table 3.6: Detailed results for data set S_5. •,○ denote a statistically significant improvement or degradation of EAST or MTRT over STRT. ■,□ denote a statistically significant improvement or degradation of EAST over MTRT.

Target	STRT	EAST	MTRT
1	0.23±0.11	0.23±0.09 ■	0.19±0.10○
2	0.04±0.09	0.11±0.10•	0.10±0.09•
3	0.25±0.11	0.26±0.09 ■	0.17±0.10○
4	0.24±0.17	0.32±0.13•□	0.36±0.13•
5	0.26±0.15	0.27±0.12	0.27±0.11
6	0.47±0.08	0.45±0.09	0.46±0.08
7	0.06±0.14	0.14±0.15•□	0.23±0.13•
8	0.29±0.11	0.31±0.10	0.28±0.10
9	0.46±0.09	0.46±0.11 ■	0.40±0.10○
10	0.32±0.11	0.30±0.10 ■	0.22±0.09○
11	0.23±0.09	0.24±0.10	0.23±0.09
12	0.29±0.10	0.32±0.10•	0.33±0.08•
13	0.12±0.13	0.15±0.12	0.12±0.08
14	0.44±0.11	0.46±0.11	0.45±0.10

EAST significantly outperforms MTRT (and performs comparable to STRT). While EAST avoids the losses of MTRT over STRT, it is also significantly worse than MTRT for two targets. For these it either selected too few support targets or a suboptimal subset of support targets.

As can be seen from Table 3.4, the only data set where EAST performs significantly worse than MTRT is S_3. Here, EAST on average is able to find a set of support targets that performs better than always selecting the

Table 3.7: Number of support targets chosen by EAST. #ST shows the average and standard deviation of the number of selected support targets. $|\mathbf{y}| - 1$ is the maximum number of support targets (the total number of targets minus one).

Data set	#ST	$\lvert\mathbf{y}\rvert - 1$
S_1	0.64±0.48	1
S_2	1.95±1.23	8
S_3	4.15±2.31	38
S_4	1.04±1.45	2
S_5	2.15±1.37	13
S_6	3.05±0.72	15

empty set, but this increase in accuracy is clearly suboptimal as EAST still performs less good than MTRT. This can also be seen from its losses for 11 targets over MTRT (Table 3.5). For such targets, EAST either selected too few or a suboptimal subset of the targets. We conjecture that the former is the case, i.e., that EAST selects too few targets: on average it only selects 4.15 ± 2.31 out of the 38 targets available in S_3 (Table 3.7). The most likely cause is the large number of targets in this data set. EAST implements a greedy hill-climbing search, which ends if the estimated loss of the next series of candidate subsets is higher than the estimate for the current set. So, if the loss of the current subset is under-estimated, EAST might stop too early. The risk that this happens increases with the number of iterations, i.e., with the optimal number of support targets. A second possible cause is that EAST may end up in a local optimum. Note that this never happens for the data sets with fewer than 10 targets (S_1, S_2 and S_4) and rarely for those with a medium number of targets (S_5, S_6), as can be seen in Table 3.5.

3.5 Conclusions

This chapter addresses the single-target with support targets prediction task, where the goal is to build a model for the main target (or one model for each target in case of multiple targets), and where a number of candidate support targets are available (only) at model induction time, which may carry information about the main target. The chief contribution is Selective Inductive Transfer (EAST), an algorithm that searches for the subset of support targets that, when predicted together with the main target in a multi-target model, maximally improves predictive performance of the main target. Experiments show that EAST, on top of a multi-target regression tree learner, performs

significantly better than single-target regression trees in 5/6 data sets and comparable in 1/6 data set. With regard to multi-target regression trees, it performs significantly better in 2/6, comparable in 3/6, and worse in 1/6 data sets.

EAST implements a greedy hill-climbing search algorithm to find the best set of support targets. As we discussed earlier, in data sets with many candidate support targets, this has two potential drawbacks: (1) it may stop adding targets too early because it relies on an empirical loss estimate (internal cross-validation), which may under-estimate the loss of the current subset, and (2) it may end up in a local optimum. Future research includes alternative strategies such as back-wards elimination of support targets and genetic search strategies.

Chapter 4

Collaborative Filtering

4.1 Introduction

Recommender systems (Adomavicius and Tuzhilin 2005b) are software systems designed to propose, given a large set of available items (books, movies, music pieces, news articles, ...), the items that are most likely of interest to the user. Two common classes of algorithms used in recommender systems are content-based algorithms and collaborative filtering (CF) algorithms. Content-based algorithms base their recommendations on the content of the items (e.g., the text of a news article), while collaborative filtering algorithms base their recommendations on information recorded by the system about the preferences of other users.

Collaborative filtering systems may rely on ratings that users have explicitly assigned to (a subset of) the items, or on binary data, which records for each user the items that he or she has previously bought (e.g., in the context of an e-commerce system). We consider the former setting. Here, the input data can be represented in terms of a ratings matrix, which has one row for each user and one column for each item and in which the elements indicate the ratings given by the users to the various items. We assume that a rating is a natural number taken from a fixed range (e.g., $1 \ldots 5$). If there are m users and n items, then the ratings matrix A is an m by n matrix and the element $A_{i,j}$ is either the rating assigned by user i to item j, or it is equal to the missing value indicator if user i has not rated item j.

Given the ratings matrix, a collaborative filtering algorithm can recommend a number of items to a particular user by first predicting ratings for all items

that the user has not rated yet, and by subsequently proposing the items with a high predicted rating. Consequently, we have a multi-target problem with each target corresponding to an unrated item. For example, to predict user i's rating for item j, a k-nearest neighbor based CI algorithm would consider the set of all other users who rated item j, among these search for the k most similar ones (in terms of their other ratings), and subsequently take the most frequently occurring value among the neighbors' ratings for item j as prediction. If predicted ratings are allowed to be real numbers, it may alternatively compute the mean instead of the most frequent value.

One of the most important issues in collaborative filtering is how to deal with data sparsity (Huang, Zeng, and Chen 2007; Sarwar et al. 2000; Huang et al. 2002; Huang, Chen, and Zeng 2004; Papagelis, Plexousakis, and Kutsuras 2005; Nanopoulos 2007; Hofmann 2004). Typically, users only rate a small percentage of the items and the ratings matrix is very sparse (includes many missing values). Data sparsity may be detrimental to accuracy because of the following two reasons. First, reliably estimating the similarity between two users becomes difficult because, due to the many missing values, there may be few items that are rated by both (and hence contain some information regarding their similarity). Second, to predict a rating $A_{i,j}$, one needs sufficient users different from user i who also rated item j. Because of data sparsity there will be a relatively small number of such users resulting in a small input set for nearest neighbor.

We propose two approaches that deal with unknown values by switching to a probabilistic representation of the problem. That is, instead of working with ratings, our algorithms work with probability distributions over the domain of possible ratings. If a user's rating for an item is known, then the corresponding probability distribution has zero variance and places all probability mass on that particular rating. If it is unknown, then the algorithms make use of a uniform, global, or predicted distribution.

On top of this probabilistic representation, we propose an estimated distance measure to compute the similarity between the rating profiles of two users. This measure can be used in nearest neighbor style collaborative filtering algorithms. As we will see, this greatly improves the performance of the nearest neighbor (kNN) approach.

Our second approach to improve upon the kNN approach consists of a graph-based algorithm called probabilistic graph-based collaborative filtering (PGBCF). This algorithm works on a user graph in which the users are nodes and the edges are labeled based on the distance measure. The algorithm seeks to improve uppon the classic kNN method by using information present in indirect neigbors. PGBCF achieves this by propagating probabilistic predictions along the edges

of the user graph.

Finally, we present an approach which uses additional contextual information to increase the predictive performance of the recommendation system. More specifically do we use the social user graph, a graph representing known relations between the users, as context. In contrast to the previous approach, the relations between users are known and the user-graph does not need to be constructed.

The rest of this chapter is organized as follows. We start with a discussion of how the problem relates to the multi-target problem setting. Frequent approaches are presented in Section 4.3. Next, we discuss variations on the Manhattan distance measure and propose the probabilistic representation in Section 4.4. Section 4.5 introduces probabilistic graph-based collaborative filtering. We present experiments comparing these approaches for various datasets and sparsity levels in Section 4.6. Next, we discuss and evaluate a variation on this approach for which the relations between the users is known in Section 4.7. Finally, Section 4.8 states the main conclusions and lists opportunities for further research.

4.2 Collaborative Filtering as a multi-target problem

The standard multi-target setting, as defined in section 2.1.1, makes a clear distinction between input and output (target) variables. It learns a a function $F : X \rightarrow Y$, mapping the input variables X to the target variables Y. This function is learned from a dataset for which the target variables Y are known, the training set, and tested on a separate test dataset. In the Collaborative Filtering setting, these restrictions are dropped: each variable is a potential input as well as output variable and there's no clear distinction between training- and test-set. This is demonstrated in Figure 4.1. We will see that this setting corresponds to a two-way learning problem which we define and discuss in the following chapter.

4.3 Related Work

A number of different general collaborative filtering techniques are described and compared in (Adomavicius and Tuzhilin 2005b), including the nearest neighbor approach (Resnick et al. 1994) that we will use to evaluate our proposed distance measure.

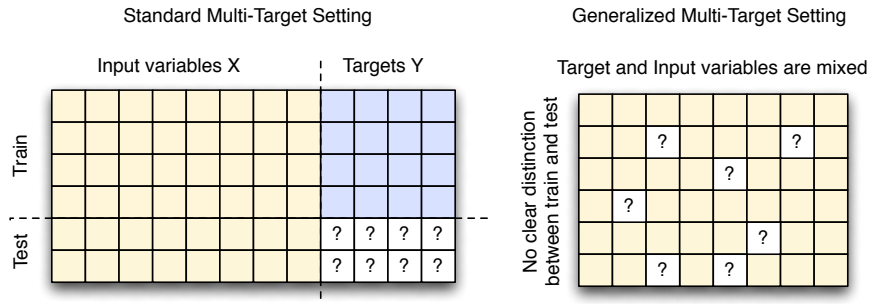


Figure 4.1: The left matrix demonstrates the standard multi-target setting. Input/output and train/test are clearly seperated. These restrictions are dropped for the Collaborative Filtering setting. The general MT setting is visualized by the right matrix.

Methods specifically designed to alleviate the sparsity and cold start problem usually consist of a hybrid method between a collaborative filter and a content-based recommender (Schein et al. 2002a; Nanopoulos 2007; Huang, Chen, and Zeng 2004). A content-based recommender uses additional information about the recommendable items. That is, items similar to those previously liked by the user are recommended. The content-based approach partially avoids the sparsity problem but at the expense of needing additional information about the items, which is not always available. Our approach differs in that it remains a pure collaborative filter.

Our proposed method uses a probabilistic representation. Probabilistic representations have previously been used successfully in Bayesian approaches (Su and Khoshgoftaar 2006; Miyahara and Pazzani 2000).

A number of different graph based approaches have been proposed in the past as well, some of which were specifically designed to alleviate the sparsity problem.

Huang, Chen, and Zeng (2004) construct a bi-partite graph, containing users on one side and items on the other side. An edge between a user and an item is added when the user likes the item. An item is then ranked for a user based on the number of paths between that item and the user. Although the system gives good results, it is limited in use because it can only handle binary data. A user either likes or dislikes an item, edges have no weight.

Papagelis, Plexousakis, and Kutsuras (2005) proposes another graph based algorithm in which the graph contains only users. The system uses only binary data: edges are weighted based on the number of co-ratings between two users.

In Nanopoulos (2007) a method is described which exploits the transitive correlations between items. i.e., some items might be heavily correlated and thus a given rating for one item might tell us something about the rating for other items as well. The algorithm recommends the top-k (transitively) correlated items. This can be seen as a hybrid approach to the sparsity problem.

Huang et al. (2002) presents a graph based recommender system. Unlike our system, the system only recommends a number of items. No predictions are made for the other items. This system uses a two-layer graph, containing user-user, item-item and user-item edges. This graph is then searched for recommendable items.

Note that none of these graph based methods can be directly compared experimentally to our method either because they predict a ranking instead of an absolute value or because they can only handle binary data.

4.4 Distance Measures and Representation

In this section, we first discuss a popular method to deal with missing ratings when computing the similarity between two users' profiles. Next, we define a probabilistic profile representation, and define the notion of expected distance based on this representation, which leads to several natural ways to better handle missing ratings.

4.4.1 Distance Measures

The similarity between two users' profiles (rows in the rating matrix) can be estimated in terms of a similarity measure or a distance measure. This chapter considers the correlation coefficient (a similarity measure) and the Manhattan distance; both are frequently used in collaborative filtering (Candillier, Meyer, and Fessant 2008). To simplify notation, we first assume that no ratings are missing and then define how both measures are extended to handle missing ratings.

Given two user profiles $\mathbf{p}$ and $\mathbf{q}$, i.e., n-dimensional vectors of which the components are ratings taken from the set of possible ratings R (e.g., $R = \{1, 2, 3, 4, 5\}$). For convenience, we repeat the definition of Pearson correlation:

$$pc(\mathbf{p}, \mathbf{q}) = \frac{\sum_i (p_i - \overline{p})(q_i - \overline{q})}{\sqrt{\sum_i (p_i - \overline{p})^2 \sum_i (q_i - \overline{q})^2}} \quad ,$$

in which $\overline{u}$ denotes the average of u's components. The Manhattan distance is defined as

$$d_{MD}(\mathbf{p}, \mathbf{q}) = ||\mathbf{p} - \mathbf{q}||_1 = \sum_{i=1}^{n} |p_i - q_i| \quad ,$$

with $||\cdot||_1$ the 1-norm.

In the presence of missing ratings, correlation is computed only over the components that are known in both $\mathbf{p}$ and $\mathbf{q}$. This approach does not apply to the Manhattan distance since the range of this measure depends on the number of components. Assuming the same distance along each dimension, this would result in a smaller value if more ratings are missing, which is not what we intend. Therefore, we compute the *scaled Manhattan distance* (*MD-Scaled*) instead, which is defined as

$$d_{\textit{MD-Scaled}}(\mathbf{p}, \mathbf{q}) = \frac{n}{|K(\mathbf{p}) \cap K(\mathbf{q})|} \sum_{i \in (K(\mathbf{p}) \cap K(\mathbf{q}))} |p_i - q_i|$$

with $K(\mathbf{u}) = \{i \mid u_i \neq ?\}$ the set of indices for which $\mathbf{u}$'s values are known. E.g., assume that $\mathbf{p} = (3, 1, ?, 0, ?)$ and $\mathbf{q} = (1, 2, ?, ?, ?)$, then $K(\mathbf{p}) = \{1, 2, 4\}$, $K(\mathbf{q}) = \{1, 2\}$ and $d_{\textit{MD-Scaled}}(\mathbf{p}, \mathbf{q}) = 5/2(|3 - 1| + |1 - 2|) = 5$.

As we will see in Section 4.6, the above measure's performance does not scale well with sparsity. If we have a sparsity of 50%, i.e., the probability of a value being known is $p = 0.5$, then the average fraction of overlap $|K(\mathbf{p}) \cap K(\mathbf{q})|/n$ between two vectors $\mathbf{p}$ and $\mathbf{q}$ is only $p^2 = 0.25$. That is, as the proportion of known values decreases linearly, the number of usable, overlapping values decreases quadratically. This in turn causes rapid performance degradation when the number of known values goes down. Both correlation and *MD-Scaled* are undefined when $K(\mathbf{p}) \cap K(\mathbf{q}) = \emptyset$, i.e., when the users do not have co-rated items.

4.4.2 Probabilistic Representation

To address the aforementioned difficult scalability of the similarity and distance measures, we propose a probabilistic representation with probabilistic user profiles, in which crisp ratings are replaced by probability distributions over the domain of possible ratings. That is, we replace each vector component p_i by its marginal probability distribution $Pr(p_i)$ ($\sum_{u \in R} Pr(p_i = u) = 1$).

We can compute the *expected Manhattan distance* (EMD) between two probabilistic user profiles as

$$\begin{aligned} & d_{EMD}(\mathbf{p}, \mathbf{q}) \\ & = E[\, ||\mathbf{p} - \mathbf{q}||_1 \,] \\ & = \sum_{i=1}^{n} \sum_{u_p \in R} \sum_{u_q \in R} |u_p - u_q| Pr(p_i = u_p) Pr(q_i = u_q) \quad . \end{aligned}$$

We assume here that the random variables corresponding to different ratings are independent.

For known ratings $p_i = p \neq ?$, $Pr(p_i)$ is defined as

$$Pr(p_i = u) = \begin{cases} 1 & \text{if } u = p \\ 0 & \text{if } u \neq p \end{cases} \quad .$$

The distribution for vector components that correspond to unknown ratings will be different. We will now consider several ways to define a distribution for this case.

Distribution for Unknown Ratings Which distribution should we use to calculate the EMD if a rating for an item is unknown? We consider the following two possibilities.

- Assume that the item's rating follows the uniform distribution $Pr(p_i = u) = 1/|R|$, with $u \in R$. We call the expected Manhattan distance computed based on this assumption the *expected Manhattan distance - uniform* or *EMD-Uni* for short.
- Suppose that the rating for item i for user j is unknown. Assuming that user j rates similar to all other users, we can use the global distribution of all known ratings for that particular item i instead of the uniform distribution. We will refer to the resulting distance as the *expected Manhattan distance - global distribution* or *EMD-GD* for short.

Consider the first option. If both users did not rate a given item, then the term in the formula for d_{EMD} that corresponds to this item only depends on the rating domain R. It does not depend on the user or on the actual item. For example,

if $R = \{1, \ldots, r\}$ then this value is the constant $c = \sum_{u_p} \sum_{u_q} |u_p - u_q| Pr(p_i = u_p) Pr(q_i = u_q) = \frac{1}{r^2} \sum_{u_p} \sum_{u_q} |u_p - u_q| = r(r^2 - 1)/(3r^2)$. This value can be seen as a penalty that is added to the Manhattan distance for each item not rated by both users. This motivates the following distance measure definition: compute the Manhattan distance over the items that are known for both users and add for each item that is not rated by both users the penalty c. We call this distance measure *Manhattan distance - penalty* or *MD-Penalty*. The main difference between *EMD-Uni* and *EMD-Penalty* lies in the items that are rated by only one of the users: for these items *MD-Penalty* also assigns a penalty of c, while *EMD-Uni* computes the expected distance based on the given distributions. *MD-Penalty* can be computed a factor $|R|$ faster than *EMD-Uni* and *EMD-GD*.

4.5 Probabilistic Graph-based collaborative filtering

In this section we will explain how the proposed distances measure can be used as the basis for graph based collaborative filtering approach..

4.5.1 The User Graph

We represent the data by a so called *user graph* $G = (V, E)$. Each node $\mathbf{v} \in V$ represents a user. If a distance d between two users/nodes can be computed, a weighted edge $e \in E$ between those nodes is added to the graph. The weight w of that edge is inversely proportional to the distance between the nodes it connects:

$$w_{e(\mathbf{v_1}, \mathbf{v_2})} = \frac{1}{d(\mathbf{v_1}, \mathbf{v_2})}$$

The graph will not contain edges between users without co-rated items when using Pearson Correlation or MD-Scaled, When using EMD or MD-Penalty, this procedure will result in a fully connected graph. An example of such a graph can be seen in Figure 4.4.

To use the graph in a collaborative filtering method, we have to complete information in each node, based on the information present in the connected nodes.

We start by translating the classic nearest neighbor approach to a graph based method. This method is then extended to make explicit use of the graph structure.

Algorithm 2 kNN predicts the missing ratings in the ratings in a matrix A. E is the set of edges defining the user graph, w is the edge weight function, and k is the number of neighbors to consider.

procedure kNN(A, E, w, k)
1: **for each** (i, j) **such that** $A_{i,j} = ?$ **do**
2: $N := k$NN-Users(i, j, A, E, w, k)
3: $\mathit{Ratings} := \{A_{x,j} \mid x \in N\}$
4: $A_{i,j} :=$ Mean values in *Ratings*
5: **end for**

procedure kNN-Users(i, j, A, E, w, k)
1: $N := \{(x, w_x) \mid (i, x) \in E \wedge w_x = w(i, x) \wedge A_{x,j} \neq ?\}$
2: **return** Top-k users from N sorted by w_x

4.5.2 Nearest Neighbor Graph Completion

Suppose we have a node $\mathbf{v} = (3, -1, ?, 0, ?)$. Our goal then consists of predicting the missing values v_3 and v_4. We can use the constructed graph to make these predictions. Since a higher edge-weight means a higher similarity, that is a closer neighbor, the missing values are more likely to be equal to the corresponding values of connected nodes for which the edge-weight of the connecting edge is high. A missing value can then be predicted as follows: select the k connected nodes with the highest edge weights, for which the value to be predicted is known, and take the average of those known values. This corresponds to the k nearest neighbors approach(Resnick et al. 1994) given in Algorithm 2.

While this method gives good results in most cases (Adomavicius and Tuzhilin 2005b), it suffers from the so called *Cold Start problem* (Schein et al. 2002b). Users who only rated a few items will be given bad ratings, and new items, only rated by a few users, will rarely be recommended.

4.5.3 PGCF

To alleviate the cold start and the sparsity problem (see Figure 4.2) we propose to use close but indirect neighbors instead of almost unrelated neighbors. This

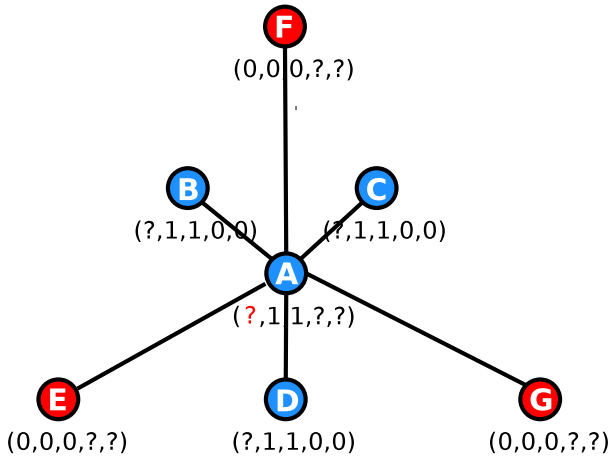


Figure 4.2: The sparsity problem. Red users have a known rating for item 1. Although user A has 3 close neigbors (B,C and D), none of them has a known rating for item 1. To make a prediction for item 1, user A has to rely on distant users E,F and G, resulting in a not so accurate prediction. Edge lengths are proportional to the distance between users.

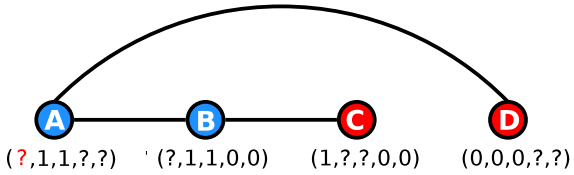


Figure 4.3: Using indirect neigbors. Users C and D gave a rating to item 1. But only D is a (distant) neighbor of user A.

is illustrated in figures 4.4 and 4.3. In Figure 4.3 the goal is to predict a rating for item 1, the red question mark, for user A. The only direct neighbor who rated this item too is user D. The resulting prediction would be 0 since user D gave the first item a rating of zero. If we take a closer look at the common ratings between users A and D, we notice that they gave completely opposite ratings. User A rated items 2 and 3 with a 1 while D gave them both a 0. It is therefore unlikely that the prediction we made was very accurate. If we look at the common ratings between A and B, the common ratings are identical and we can assume that they would rate other items similarly as well. The same is true for B and C. We can thus argue that B would rate item 1 with a 1 since user C did so, and if B would rate item 1 with a 1, user A would do so too. By

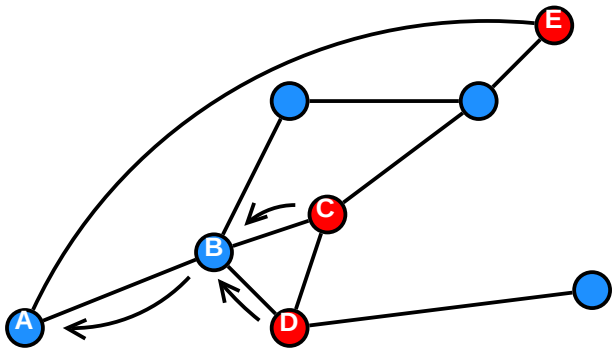


Figure 4.4: A user-graph. Each node represents a user in the dataset. When a distance between two users can be computed, a weighted edge is placed between those users. The arc lengths correspond to distances. Known values (in C,D and E) can be propagated, e.g., from C and D to B and to A subsequently. Without propagation, the predicted value for user A would be based solely on user E, resulting in a bad prediction due to the large distance between A and E.

making the prediction for user B first and subsequently using this prediction to predict item 1 for user A, we can largely avoid the sparsity problem and obtain a more accurate prediction.

This method of propagating predictions will form the basis of our graph based collaborative filtering algorithm and is illustrated in Figure 4.4. One problem with this approach is the following. Making predictions and regarding them as known values is prone to divergence. Errors would be propagated and start to accumulate. To remedy this we have to take two measures:

- make the most certain prediction first
- take the uncertainty of previous predictions into account when using them for other predictions

The Prediction Order Our goal is to fill in first the predictions we can make most confidently. To this end, we need a way to determine the confidence we have in a prediction. Suppose we make a prediction based on k neighbors and all of those neighbors rated the item for which we are predicting a rating exactly the same. This will give us a prediction with a maximum confidence. If, on the

Algorithm 3 PGBCF predicts the missing ratings in the ratings in a matrix A. E is the set of edges defining the user graph, w is the edge weight function, k is the number of neighbors to consider, and R is the rating domain. See Alg. 2 for kNN-Users.

procedure PGBCF(A, E, w, k, R)

1: { *Find prediction order* }
2: $Todo := \emptyset$
3: **for each** (i, j) **such that** $A_{i,j} = ?$ **do**
4: $N := k\text{NN-Users}(i, j, A, E, w, k)$
5: $v :=$ Variance($\{A_{x,j} \mid x \in N\}$)
6: $Todo := Todo \cup \{ (i, j, |N|, v) \}$
7: **end for**
8: Sort *Todo* by decreasing $|N|$ and increasing v
9: { *Initialize distributions for known ratings* }
10: **for each** (i, j) **such that** $A_{i,j} \neq ?$ **do**
11: **for each** $r \in R$ **do**
12: $$Pr(A_{i,j} = r) = \begin{cases} 1 & \text{if } r = A_{i,j} \\ 0 & \text{if } r \neq A_{i,j} \end{cases}$$
13: **end for**
14: **end for**
15: { *Propagate (predicted) distributions* }
16: **for each** $(i, j, \cdot, \cdot) \in Todo$ **do**
17: $N := k\text{NN-Users}(i, j, A, E, w, k)$
18: $Pr(A_{i,j}) := \frac{1}{|N|} \sum_{x \in N} Pr(A_{x,j})$
19: $A_{i,j} :=$ Median $Pr(A_{i,j})$
20: **end for**

other hand, all the neighbors rate the item differently, we will not be able to make a confident prediction.

The variance of the neighbor-ratings used to make the prediction gives us thus a measure of confidence of the prediction. A low variance tells us the prediction is most likely correct. A high variance tells us we have low confidence in the prediction.

We can now use the variance to order the sequence in which we will fill in predictions in the graph.

Propagating Uncertainty Suppose our users can only rate an item positive or negative and we are making a prediction for some item for user A based on the three nearest neighbors. Two out of three neigbors rated the item positively

and one neighbor gave the item a negative rating. In this case, we would predict that user A would rate the item positive but we would not be very confident in this prediction. Only 2/3 neighbors gave a positive rating, i.e. we would assign a confidence level of 2/3 to the prediction.

In our algorithm, predicted values are used for subsequent predictions. For example: when making a prediction for user B, this user might have user A as one of its neigbors. If the neighbors of B (other than A) all rate the item positively, and since our prediction for A was positive, the prediction for B would be positive too. Assigning a confidence value to this prediction is now a little harder. If we regard the predicted value for A as a known value, all of B's neighbors would have rated the item positively and the resulting confidence value would be 1. Doing so disregards the fact that we were not completely confident in our prediction for user A.

As a solution to this shortcoming, we propose to store the distribution of ratings among the neigbors instead of the prediction resulting from this distribution, i.e. instead of storing a positive prediction for user A, we would store $(\oplus 2/3, \ominus 1/3)$. When making the prediction for user B, we use this distribution instead of the prediction. This is accomplished by splitting A's vote in the prediction for B according to the distribution. i.e. if B has two neigbors with a positive vote and the distribution of user A, we would get the following predicted distribution for user B: $(\oplus = \frac{1+1+2/3}{3} = \frac{8}{9}, \ominus = \frac{1/3}{3} = \frac{1}{9})$ or a positive prediction for B with a confidence of 8/9. The fact that we were not completely confident in our prediction for A is now taken into account when making the prediction for B and is reflected in the confidence we have in the prediction for user B. A second benefit from this method is that a low confidence prediction has less influence on a subsequent prediction as its vote is split up more evenly.

As our algorithm works on non-binary data, the method described above is extended to discrete distributions. E.g., suppose we have ratings between 1 and 5 and 5 neighbors rating $3, 4, 3, 5$ and 2 respectively, we would get the following distribution $Pr = (0, \frac{1}{5}, \frac{2}{5}, \frac{1}{5}, \frac{1}{5})$ Since a rating could be the result of a prediction, it is possible that the rating is represented as a distribution. A distribution D, based on the distributions Pr^i of n neighbors is then calculated as follows:

$Pr_j = \sum_{i=1..n} D^i_j / n$

The final prediction is made by taking the median of this distribution. To compare the confidence we have in a prediction we can use the variance of the distribution. The lower the variance, the more confident we are in a prediction. i.e. $D = (0, 1, 0, 0, 0)$ has a variance of 0, all neighbors gave the same rating (2 in this case) and thus the prediction would be accurate.

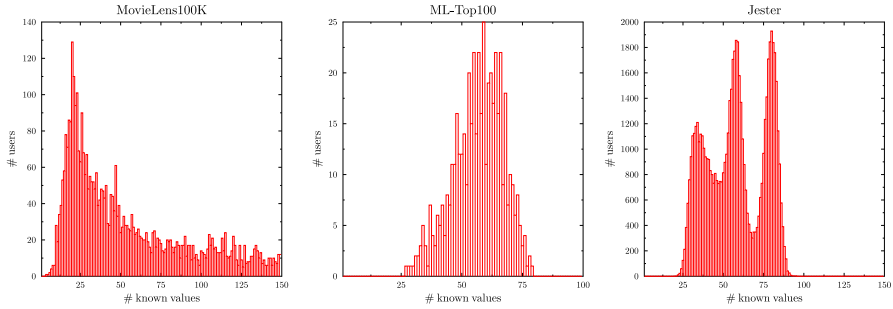


Figure 4.5: Number of users (vertical axis) who rated n items (horizontal axis)

4.6 Experimental results

We have run a number of experiments in order to evaluate the performance of the proposed distance measures and graph based algorithms.

The first dataset used in these experiments is the well known MovieLens (Resnick et al. 1994) dataset which consists of 100000 movie ratings. The second one is the subset of a subset of this dataset consisting of the top 100 most rated movies and the top 100 users who gave the most ratings as used in Srebro, Rennie, and Jaakkola (2005). The third dataset is the Jester joke database, consisting of a number of rated jokes. The details of these datasets can be found in Table 4.1 and Figure 4.5 shows the distribution of the number of rated items by the users.

Table 4.1: Description of the MovieLens100k, the top100 subset and the Jester dataset. Sparsity is defined as the ratio of known ratings to the number of possible ratings: $sparsity = \frac{\#rateditems}{\#users * \#items}$

	ML-100K	ML-Top100	Jester
# users	943	100	73,421
# items	1682	100	100
# rated items	100,000	7086	4.1 Million
sparsity	0.93	0.29	0.44

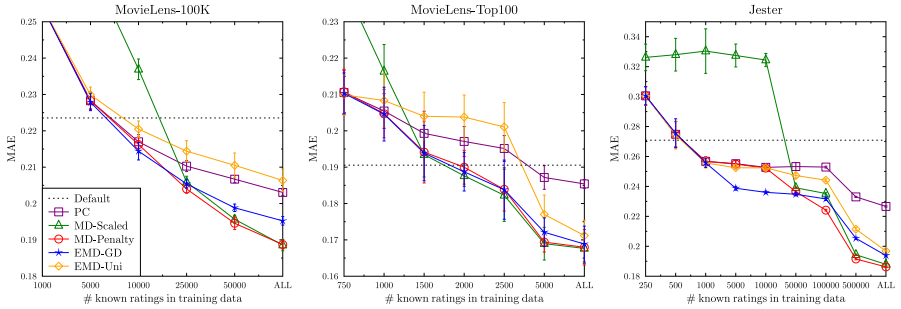


Figure 4.6: Cross-validated MAE versus dataset sparsity (sampled from full dataset) for 10NN in combination with different distance measures.

To evaluate the performance, the dataset is split in 5 disjoint train/test splits, with 80% of the data in each training set. Performance is measured using the *mean average error* (MAE). We will discuss the performance under varying sparsity levels and analyze in depth the performance of the algorithm for a new user, i.e., the effect of the cold start performance.

4.6.1 Distance Measure Evaluation

Figure 4.6 shows the cross-validated mean absolute error as a function of training set sparsity for nearest neighbor (Algorithm 2) in combination with the distance measures discussed in Section 4.4. These include the Pearson correlation (PC), scaled Manhattan distance (MD-Scaled), Manhattan distance with penalty (MD-Penalty), and the expected Manhattan distance based on the global (EMD-GD) and uniform (EMD-Uni) distribution. The figure also indicates the default MAE, which corresponds to always predicting the overall median of the rankings in the dataset. Other methods of alleviating the sparsity problem are not considered in this comparison because their performance has little influence on the usefulness of our proposed distance measure. For example, using a hybrid method (Schein et al. 2002a) to alleviate the sparsity problem could very well be combined with our distance measures. Note that the default MAE is not useful in practice as it cannot be used to rank items.

PC, which is frequently used in collaborative filtering, and EMD-Uni are consistently outperformed by both MD-Penalty and EMD-GD across all datasets and sparsity levels. MD-Scaled performs best or second-best for low dataset sparsity. If sparsity increases, however, its performance significantly deteriorates; MD-Scaled is the worst performing distance measure on sparse datasets. The reason is that this distance measure becomes unstable if the overlap between the users' profiles is too small (Section 4.4.1). MD-Penalty and EMD-GD perform well overall. They have similar performance on MovieLens-Top100. The results on Jester, and to a lesser extent also those on MovieLens-100K, show that MD-Penalty works best for low sparsity levels, while EMD-GD is better for high sparsity levels. Based on these observations, we recommend EMD-GD for sparse datasets and MD-Penalty for less sparse datasets.

Recall that MD-Penalty adds a fixed penalty to the Manhattan distance for each item that is not rated by both users. As a result, MD-Penalty explicitly takes the number of common ratings into account. A lower number of co-rated items yields a higher distance. One might argue that users with a high number of common ratings are more likely to have similar taste, irrespectively of the ratings given. This is reflected in the MD-Penalty measure.

4.6.2 Performance of PGBCF

Figure 4.7 shows the performance of probabilistic graph based collaborative filtering (PGBCF) compared to that of kNN. Both use MD-Scaled as distance measure. Comparing to the graph based algorithms described in section 4.3 is not possible due to them only working on binary datasets or directly predicting a rank.

PGBCF performs comparable to kNN in most of the experiments. We expect PGBCF to be most useful on sparse datasets. If the data is not sufficiently sparse, then propagating predictions is less useful and may decrease accuracy (this is known as the over activation effect (Huang, Chen, and Zeng 2004)). This hypothesis is confirmed by the results: on MovieLens-100K, PGBCF outperforms kNN on sparse datasets. The same is true for Jester. Therefore, PGBCF should be considered for predicting ratings on sparse datasets.

PGBCF's computational cost is about twice that of kNN (ignoring the time required for sorting the unknown ratings). So, the overhead of propagating predictions is relatively small. This is only the case if the goal is to compute item ratings for all users. If we are interested in making predictions only for

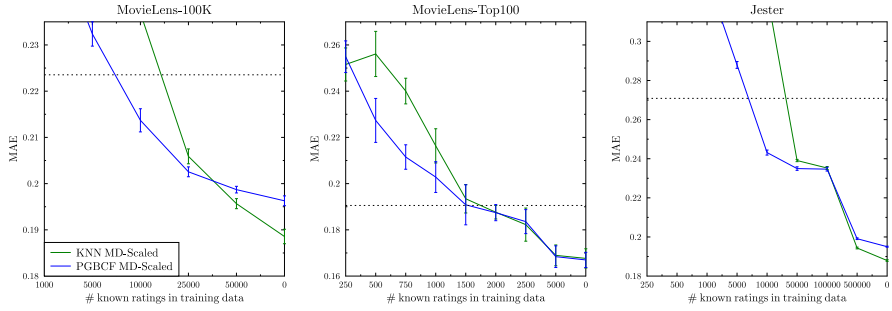


Figure 4.7: Cross-validated MAE of PGBCF and kNN, both using MD-Scaled, across various sparsity levels. We use the original scaled distance measure to study the performance using the graph based method.

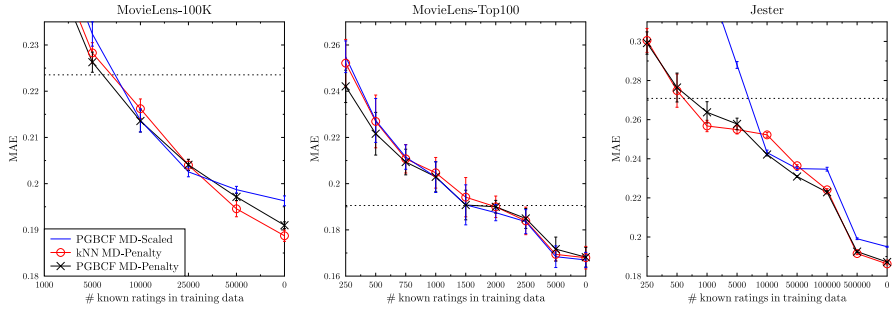


Figure 4.8: Combing our distance measure and our graph based approach.

one user, then this can be easily accomplished with kNN, but not with PGBCF because it makes predictions for all users in a pre-determined order.

Combining the MD-Penalty distance measure with the graph based method gives a small additional improvement on the MovieLens dataset for high sparsity levels (see Figure 4.8) . When the dataset gets less sparse, kNN-MD-Scaled outperforms the combined method. This is a result of PGBCF performing slightly worse than kNN-MD-Penalty on the not so sparse datasets. For the Jester dataset, the combined method performs close to kNN-MD-Penalty, except for when PGBCF outperforms kNN-MD-Penalty, in which case the combined method also outperforms kNN-MD-Penalty.

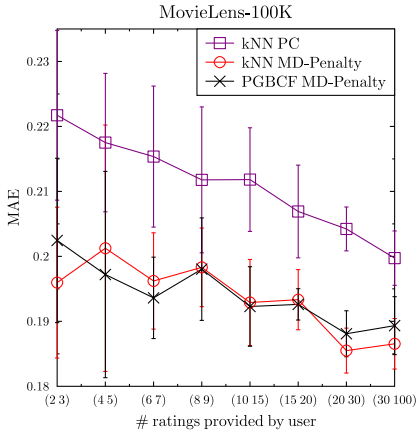


Figure 4.9: 5-fold Cross-validated average error across users who rated a given number of items

4.6.3 Robustness Against Cold Start Problem

To analyze the robustness of the algorithms with regard to the cold start problem, we measure the MAE over users of which the number of rated items lies in a given interval. For example, the MAE of all users who rated only 2-3 items shows how the algorithms perform on new users. Figure 4.9 shows these results for different intervals. We only show results for MovieLens-100K. For the other datasets, there are no users who rated only this few items (See histograms in Figure 4.5).

The results clearly show that the new distance measure outperforms the commonly used Pearson correlation. Although the variance on the results is high, we observe that PGBCF performs better than kNN for users who rated a small number of items. Only for users who rated only 2 or 3 items, PGBCF drops below kNN.

4.7 Context-Aware Recommendation

The approaches discussed so far only use the user ratings to make a prediction. In the context-aware recommendation setting, we are given more information about each user which can be used to further improve the predictions made by

the recommendation system. This section presents a method, unrelated to the previously discussed approach, which uses a user's social neighborhood as the context.

We propose a method which combines two similarity measures which each result in a predicted rating; these are then averaged to obtain a final prediction for each user/movie-pair.

Linear Regression based k-NN

Our first similarity consists of the fit of a linear regression between the ratings given by two users. This is very similar to using correlation as a similarity measure, but is more robust to differences in calibration. Suppose user u_1 rated some movies as $(2, 3, 1, 4, 4, ?)$ (with ? the rating which we want to predict), and u_2 rated those movies as $(3, 4, 2, 5, 5, 3)$. The known ratings of u_1 and u_2 correlate perfectly. Standard 1NN would use u_2's rating of 3 as prediction for the unknown rating of u_1. This does not make much sense, as u_2 consistently rates movies higher than u_1. Performing a linear regression solves this problem: it results in the equation $y = x - 1$, and filling in u_2's rating of 3 would give the more likely result: a predicted rating of 2. The linear regression based prediction is computed using

$$LR - Prediction(u_i, m_j) = \frac{\sum_{u_k \in MSU(u_i, m_j)} LR(u_i, u_k, m_j)}{k} \tag{4.1}$$

where $MSU(u_i, m_j)$ is the set of most similar users, according to the fit of the linear regression, *among those who rated* m_j. $LR(u_i, u_k, m_j)$ is the prediction for pair u_i, m_j based on the linear regression between u_i and u_k in which the rating for m_j given by u_i is filled in.

We will see in Chapter 5 that this approach can be successfully applied to other problems as well.

Social Neighborhood predictions

The second similarity measure is based on the social neighborhood of a user to predict ratings. A prediction is made by averaging the ratings given to the movie of interest by the user's friends:

$$FriendsPred(u_i, m_j) = \frac{\sum_{u_k \in Friends(u_i, m_j)} Ra(u_k, m_j)}{k} \tag{4.2}$$

where $Friends(u_i, m_j)$ are u_i's friends who rated m_j, and $Ra(u_k, m_j)$ those ratings. This approach can be extended to include friends-of-friends. More

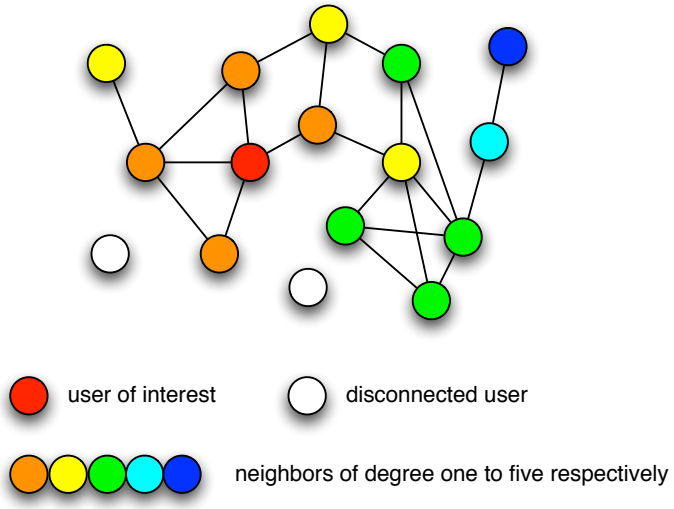


Figure 4.10: The social neighborhood of a user.

generally, we define the i-th-degree social neighborhood of u_k as the set of all people within a friend distance of at most i from u_k. Figure 4.10 depicts such a social neighborhood. This gives the following formula to make predictions based on a user's social neighborhood.

$$SocNeighPred(u_i, m_j, n) = \frac{\sum_{u_k \in SocNeigh(u_i, m_j, n)} Ra(u_k, m_j)}{k} \tag{4.3}$$

with n the degree of the social neighborhood. With a too low degree, we may not have enough people, and thus ratings, in the neighborhood; with too high a degree, we will overgeneralize.

The predictions from the linear regression method and the social neighborhood method are averaged to give the final prediction.

Note that the social neighborhood graph is explicitly given. It is not constructed. This stands in strong contrast with the graph of section 4.5.1 which, although structurally very similar, is constructed from the known ratings.

Training

We tested social neighborhoods with degrees between 1 and 5. The performance increased up to degree 3 and went back down thereafter. Table 4.2 gives the

	Second	Third	Fourth
p@5	0.107823	0.147738	0.135386
p@10	0.103589	0.123460	0.096503
MAP	0.077488	0.096026	0.075459
AUC	0.980484	0.982339	0.980253

Table 4.2: Evaluation metrics for predictions based on degree 2, 3, and 4 social neighborhoods.

results for degrees 2 to 4.

4.7.1 Evaluation

Our results are obtained by evaluating the algorithm on all possible movie/user-pairs. To make this computationally feasible, one extra rule was added to the system: if a movie is not rated by at least 250 users, the prediction for that movie, for any user, is the lowest rating possible, i.e. it will not be recommended. As it turns out, this rule does not only make prediction more efficient, it also increases predictive performance by a large margin. This caused us to evaluate a trivially simple prediction method: recommend a movie, irregardless of the user, if and only if it is rated by over 100 users. This gave us a remarkably high AUC of 0.943. Thus, the simple rule "recommend a movie if many people have seen it" seems to work well, and might be useful as a reference point.

We compare our method to two other approaches. The first consist of kNN-approach using an elaborate distance measure. The second approach consists of an inductive logic programming (ILP) method. Both algorithms are discussed in Piccart et al. (2010).

The results of the three approaches are summarized in Table 4.3. For the first approach, results for $k = 100$ and $k = 3000$ are reported. Between these values, all evaluation criteria increase monotonically with k. LR performs worse in terms of precision, but has a higher AUC. The ILP approach performs somewhat similar to the first k-NN approaches, with higher AUC but comparable or lower precisions. Our proposed social neighborhood based method performs much better on all metrics, and combining it with LR gives a small further improvement (results shown for $n = 3$).

These results are somewhat approximative. First, the P@5, P@10, MAP and AUC values reported here are obtained based on a ranking over all user/movie pairs; this gives a micro-average of the corresponding values per user. MAP is usually defined as a macro-average of average precision. Second, P@5 and

	100-NN	3000-NN	LR	Soc	LR+Soc	ILP
p@5	0.057	0.092	0.023	0.148	0.148	0.062
p@10	0.056	0.086	0.024	0.123	0.123	0.018
MAP	0.009	0.019	0.019	0.096	0.097	0.013
AUC	0.576	0.612	0.960	0.982	0.988	0.652

Table 4.3: Final results for the three approaches. LR+Soc evaluated on subset only.

P@10 are actually approximated by P@R, where R is the smallest recall value at least equal to 0.05 or 0.10, respectively. For micro-averaged values this gives a good approximation. Generally, the results are such that we expect LR+Soc, with $n = 3$, to make the best predictions on the evaluation set.

4.8 Conclusions and Further Work

This chapter addressed the sparsity and cold start problems by switching to a probabilistic representation of the rankings. We first proposed a number of variations on the Manhattan distance that make use of this representation (EMD-GD and EMD-Uni) or that are inspired by it (MD-Penalty). Next, we employed this representation in probabilistic graph based collaborative filtering (PGBCF), an algorithm that propagates probabilistic ranking predictions through the user graph.

Experiments show that the new distance measures (in particular, MD-Penalty and EMD-GD) consistently outperform common similarity and distance measures such as Pearson correlation and the scaled Manhattan distance. They should definitely be considered in practical applications. Further results show that PGBCF may outperform nearest neighbor on sparse datasets or on new users who rated only a few items.

Finally, we proposed a simple but well performing method to handle Context-Aware Movie recommendation. The method uses a user's social neighborhood and makes predictions on the assumption that friends have a preference for similar movies.

Chapter 5

Learning in two-way datasets

5.1 Introduction

The standard multi-target setting (see section 2.1.1) learns a function $F : X \to Y$, mapping the input variables X to the target variables Y. This function is learned using a dataset S containing pairs $(\mathbf{x}, \mathbf{y})$ with $\mathbf{x} \in X$ the input vector and $\mathbf{y} \in Y = Y_1 \times \cdots \times Y_n$ the target vector. Denote with $y_i \in Y_i$ the i'th component of $\mathbf{y}$.

Now imagine that we have a set of objects $A \subseteq \mathcal{A}$ and a set of objects $B \subseteq \mathcal{B}$, and the data set contains pairs $(\mathbf{a}, \mathbf{b})$ with $\mathbf{a} \in A$ and $\mathbf{b} \in B$, with a label $y \in Y$ assigned to each pair. We call this a two-way dataset. The corresponding learning setting is called two-way predictive learning.

The task of two-way predictive learning is defined as follows:

Given: a data set D that consists of a set of objects $A \subseteq \mathcal{A}$ and $B \subseteq \mathcal{B}$ and for some combinations of $\mathbf{a}$ and $\mathbf{b}$ a corresponding label y, that is,

$$D = \{(\mathbf{a}, \mathbf{b}, y) | \mathbf{a} \in A, \mathbf{b} \in B, y \in Y\} \text{ with } A \subseteq \mathcal{A} \text{ and } B \subseteq \mathcal{B},$$

Find: a function $f : (A, B) \to Y$, such that $\sum_{(\mathbf{a},\mathbf{b},y)\in D} L(f(\mathbf{a}, \mathbf{b}), y)$ is minimized, with L some loss function over Y.

This two-way setting can be reduced to the first setting. (see Figure 5.1 for an example) However, this reduction hides the structure of the data: it hides the fact that X is of the form $A \times B$, that is, that each $\mathbf{a}$ is paired with each $\mathbf{b}$ in the dataset. Because of this complete pairing, the data are not i.i.d. (independent

and identically distributed): when $(\mathbf{a}, \mathbf{b})$ occurs in D, we know that there will be other examples with the same $\mathbf{a}$ but a different $\mathbf{b}$. The elements of A and B may themselves be i.i.d., but the tuples in D are not. Some learning methods assume data to be i.i.d.; the performance of such methods can deteriorate in the presence of non-i.i.d. data (see, e.g., Jensen and Neville (2002)). Also most theoretical work on computational learning assumes i.i.d. data; predictive learning in the two-way setting has not been studied in much detail. Thus, not everything we know about predictive learning will apply in the two-way learning setting.

The two-way setting is clearly a special case of relational learning (De Raedt 2008). Indeed, we predict an attribute of a relation that connects two different types of objects, based on the information available in those objects. Many things known from relational learning do apply to two-way learning. However, the two-way setting is more specific than the general relational learning setting, and exploiting this specificity may lead to better predictive performance.

The two-way setting is relevant for many applications. We list but a few. First, consider molecular biology. Microarray data form a full matrix that shows the expression level of a set of genes A under a set of conditions B. Biclustering is an example of a task that can be performed on such data: clusters have to be found simultaneously on both rows and columns of the data. Biclustering is an unsupervised learning problem, but predictive learning can just as well be useful on such data (and in fact the biclustering is often a first step towards prediction of, for instance, gene functions).

Second, consider recommender systems, which recommend items (A) to users (B) by predicting a score for user-item-pairs, and recommending to a user the items that score highest for her. This is a predictive two-way learning setting and is visualized in figure 5.1.

Third, several toy examples in statistical relational learning, such as the student-course-grade example (Getoor et al. 2001), are models that essentially describe a two-way prediction problem such as the one stated above.

The above list is not exhaustive, but suffices to show that the two-way learning setting occurs frequently, in very diverse application domains. Different types of solutions have been proposed in different domains. Identifying two-way predictive learning as a separate learning task may lead to more generally applicable solutions, and to easier knowledge transfer between the different domains.

In this chapter we initiate an investigation into two-way predictive learning. We introduce the two-way setting in more detail in Section 2. We present some thoughts and insights about learning in this setting in Sections 3 and 4. In

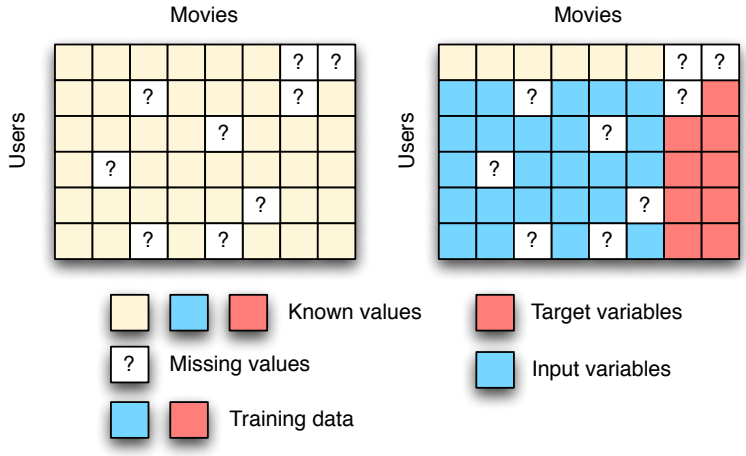


Figure 5.1: Recommender system as an example of a two-way dataset and how it reduces to a multi-target problem. The objects A and B consist of the users and movies and the values are the scores for each movie-user pair. To predict a score for the last two movies, one could build a MT target model which takes all but the last two variables as input and the last two as target variables, thus reducing the two-way prediction problem to a MT prediction problem.

Section 5 we evaluate some of these ideas experimentally, showing that they can lead to improved predictive performance. We conclude in Section 6.

5.2 Two-way learning

We consider two-way learning as a special case of relational learning. In general, consider the following context: we are given two types of objects $\mathcal{A}$ and $\mathcal{B}$, and a relation $\mathcal{R}$ between them. The objects of type $\mathcal{A}$ have attributes A_i, $i = 1, \ldots, n_A$; the objects of type $\mathcal{B}$ have attributes B_i, $i = 1, \ldots, n_B$; and the tuples in $\mathcal{R}$ have attributes R_i, $i = 1, \ldots, n_R$ as well as a special attribute T called the target attribute. We denote the set of attributes of $\mathcal{A}$, $\mathcal{B}$, $\mathcal{R}$ as $Attr(\mathcal{A})$, $Attr(\mathcal{B})$, $Attr(\mathcal{R})$, and their respective extensions as A, B, R. The relation R is complete: there is a relationship between each $\mathbf{a} \in A$ and each $\mathbf{b} \in B$. Figure 5.2 summarizes this in an entity-relationship diagram.

The task is to predict T from the other available information. We can consider multiple settings here, depending on what attributes are available. Table 5.1 provides an overview. While the ER-diagram allows for $\mathcal{R}$ to have its own

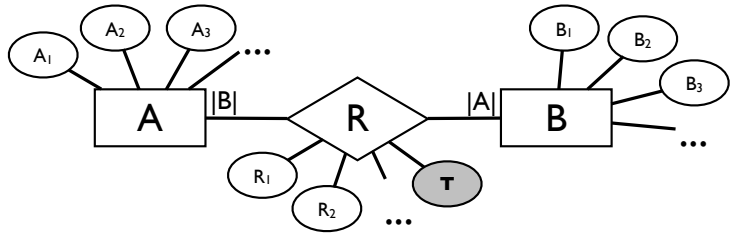


Figure 5.2: ER-diagram summarizing the data types available for learning. T is the target attribute.

Table 5.1: Overview of two-way learning settings. In the table, * means 'non-empty'.

$Attr(\mathcal{A})$	$Attr(\mathcal{B})$	$Attr(\mathcal{R}) \setminus \{T\}$	
$\emptyset$	$\emptyset$	$\emptyset$	bare two-way learning
*	$\emptyset$	$\emptyset$	single-decorated two-way learning
$\emptyset$	*	$\emptyset$	single-decorated two-way learning
*	*	$\emptyset$	double-decorated two-way learning
$\emptyset$	$\emptyset$	*	relational learning with deterministic background
*	$\emptyset$	*	relational learning with deterministic background
$\emptyset$	*	*	relational learning with deterministic background
*	*	*	relational learning with deterministic background

attributes, besides T, in this chapter we focus on cases where $Attr(\mathcal{R}) = \{T\}$. In those cases, the available data can be represented as illustrated in Figure 5.3. We know discuss the different settings from Table 5.1 in turn.

5.2.1 Bare two-way learning

When R is complete, $Attr(\mathcal{A}) = Attr(\mathcal{B}) = \emptyset$, and $Attr(\mathcal{R}) = \{T\}$, the data set D is essentially a matrix. For each $\mathbf{a}_i$ and $\mathbf{b}_j$, we denote the corresponding T value as t_{ij}. This is the type of data we get in microarray data. We call this *bare* two-way learning, "bare" referring to the fact that the objects in $\mathcal{A}$ and $\mathcal{B}$ are not decorated with attributes.

It may seem strange to try to predict T when $Attr(\mathcal{A}) = Attr(\mathcal{B}) = \emptyset$. How can one predict a target attribute from objects that have no attributes themselves? The point is that the values of T themselves carry information. We can predict

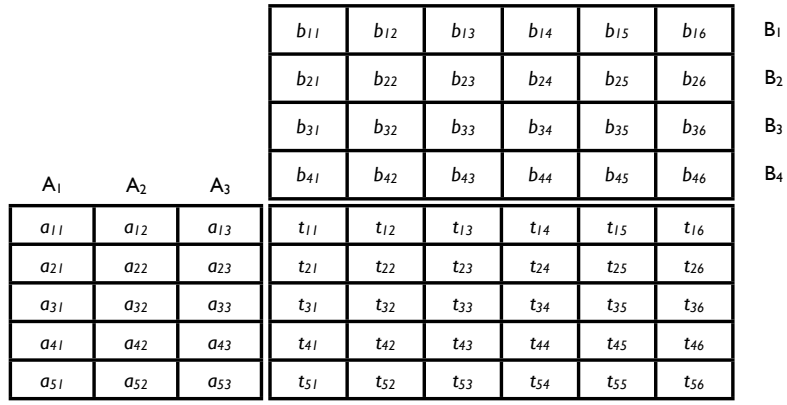


Figure 5.3: Double-decorated two-way learning, illustrated for $n_A = 3$, $n_B = 4$, $|A| = 5$ and $|B| = 6$. In single-decorated two-way learning, the B matrix would be absent; in bare two-way learning, the A and B matrices are absent.

t_{ij} from the information in the $t_{ik}, k \neq j$, or from the $t_{kj}, k \neq i$, or even from the $t_{kl}, k \neq i, l \neq j$.

As an example of this setting, take biclustering. Clusters are formed in $\mathcal{A}$ based on the T values obtained when combining an $\mathbf{a} \in \mathcal{A}$ with the objects in B, and vice versa. In other words, given sets of objects $A \subseteq \mathcal{A}$ and $B \subseteq \mathcal{B}$, the columns $T_{\cdot j}$ are seen as n_B attributes describing the objects $\mathbf{a}_i \in A$, and, vice versa, the rows $T_{i\cdot}$ are seen as n_A attributes describing the objects in B.

Bare two-way predictive learning can be addressed in different ways. Suppose we need to predict a single T_{ij} element. We distinguish the following approaches:

Row-based, inductive: We learn a function f that predicts $T_{\cdot j}$ from $T_{\cdot k}, k \neq j$. That is, we reduce the task to a standard learning task, treating the rows as instances and the columns as attributes. The target attribute is $T_{\cdot j}$, and the predictive attributes are $T_{\cdot k}$ with $k \neq j$.

Column-based, inductive: We learn a function f that predicts $T_{i\cdot}$ from $T_{k\cdot}, k \neq i$. That is, we reduce the task to a standard learning task, treating the *columns* as instances and the *rows* as attributes. The target attribute is $T_{i\cdot}$, and the predictive attributes are $T_{k\cdot}$, $k \neq i$. We call this *transposed learning*, as it really corresponds to transposing the matrix that represents the data set and then using a standard learning method.

Transductive: In transductive learning, the task is not to learn a predictive model, but simply make the predictions. Any t_{ij} can be predicted from any t_{kl}, $(k, l) \neq (i, j)$.

Bare two-way predictive learning is encountered, for instance, in the context of recommender systems (Adomavicius and Tuzhilin 2005a). The two types of objects are users and items, and the T values are preference scores that users may give to items.

5.2.2 Single-decorated two-way learning

When $Attr(\mathcal{A})$ is not empty, but $Attr(\mathcal{B})$ and $Attr(\mathcal{R}) \setminus \{T\}$ are, we talk about *single-decorated* two-way learning. (The case where $Attr(\mathcal{A})$ is empty but $Attr(\mathcal{B})$ is not, is of course equivalent; we do not treat it separately.)

The situation here is as follows: we have a data set with elements $(\mathbf{a}, \mathbf{b}, f(\mathbf{a}, \mathbf{b}))$ with $\mathbf{a} \in A$ and $\mathbf{b} \in B$; the elements of A are described by attributes A_i but B is just a set of elements without attributes. We then have $|B|$ values associated with each $\mathbf{a}$, namely one for each $\mathbf{b}_i$, $i = 1, \ldots, |B|$. We can create separate attributes for each of them. This yields a dataset with the following schema:

A_1	A_2	$\ldots$	A_{n_A}	T_1	T_2	$\ldots$	$T_{\lvert B\rvert}$

where for any $\mathbf{a}$, T_i has the value $f(\mathbf{a}, \mathbf{b}_i)$ with $B = \{\mathbf{b}_1, \ldots, \mathbf{b}_{|B|}\}$.

By treating the $f(\mathbf{a}, \mathbf{b}_i)$ values as different targets T_i, we reduce the single-decorated two-way learning setting to a multi-target prediction problem. Problems of this type have been studied and methods exist for solving them (Aho, Zenko, and Dzeroski 2009). Closely related settings are multi-task learning (Caruana 1997b), multi-label classification (Tsoumakas, Katakis, and Vlahavas 2010) and structured output prediction (Tsochantaridis et al. 2005).

Placing these settings in the context of two-way predictive learning facilitates transfer of solution strategies from these settings to general two-way predictive learning, and vice versa. We will see in Section 5.5 how multi-target problems can be solved in alternative ways, resulting in improved predictive performance.

5.2.3 Double-decorated two-way learning

When both $Attr(\mathcal{A})$ and $Attr(\mathcal{B})$ are non-empty, but $Attr(\mathcal{R}) \setminus \{T\} = \emptyset$ is, we have *double-decorated* two-way learning. This setting does not correspond

to any settings identified earlier, except that it is a special case of relational learning. The most natural schema for this setting is the schema that results from joining A, R and B, which gives

A_1	A_2	...	A_{n_A}	B_1	B_2	...	B_{n_B}	T

Figure 5.3 shows schematically the structure of the data in this setting. For single-decorated or bare two-way learning, either the A_i or B_i, or both, are absent.

5.2.4 Relational learning with deterministic background knowledge

When $Attr(\mathcal{R}) \setminus \{T\}$ is non-empty, we have a setting that we call *relational learning with deterministic background knowledge*. Joining A, B and R gives the following schema:

A_1	A_2	...	A_{n_A}	B_1	B_2	...	B_{n_B}	R_1	R_2	...	R_{n_R}	T

As the R_i attributes have unique values for each $(\mathbf{a}, \mathbf{b})$ pair, there is no possibility to reduce the number of tuples in this case, though redundancy in the table (because of repeated **a** and **b** values) can be avoided by adopting a relational representation, e.g.,:

a(ida, a_1, a_2, ..., a_n).
...
b(idb, b_1, b_2, ..., b_m).
...
r(ida,idb,r_1, r_2, ..., r_k, t).
...

Note that the relationship between R and A (and also between R and B) is many-to-one: with each R-tuple exactly one A-tuple and B-tuple is associated. As a result, the join contains as many tuples as the original R table. In inductive logic programming terminology (Lavrač and Džeroski 1994), R contains the examples and A and B constitute background knowledge. The background knowledge is deterministic because each R-tuple matches with exactly one tuple in each background relation.

One could argue that this setting does not really require a relational learner, since the relations can simply be joined into one table without loss of information, after which a standard (non-relational, or propositional) learner can be applied. This is only partially true; two issues arise.

First, the tuples in the joint table are no longer i.i.d.: when a value occurs for some attribute in one tuple, it will occur in other tuples as well. It is in general not a good idea to treat data as i.i.d. (which propositional learners naturally do) when they are not; it can lead to undesired bias (Jensen and Neville 2002).

Second, although the join does not cause loss of information from the relational database point of view, a propositional learner will still have no access to information that may be relevant for a tuple, but is not included in that tuple. As an example, consider the possibility of adding to R's attributes a new attribute that contains the mean value of all the t values in R-tuples with the same *ida* value as the current tuple. Such an attribute might be relevant for prediction, but it cannot be deduced from the information included in a single tuple in the attribute-value representation.

Thus, even though this type of data can easily be transformed into the propositional format (a table with one row per instance) and a propositional learner used afterwards (this approach is taken by, for instance, the ILP system DINUS (Džeroski, Muggleton, and Russell 1992)), such a transformation may destroy important information.

5.3 Different types of predictive learning

Apart from having different types of input data, we can consider different types of learning as well. Let us first distinguish transductive from inductive learning. In transductive learning, the result of the learning process is a set of predictions. In inductive learning, the result is a function that takes certain inputs and produces predictions from that. Such functions can differ strongly with respect to the type of inputs they take, and hence, the conditions under which they are applicable (as their input must be available at prediction time).

Transductive learning

The transductive learning task is defined as follows: given a data set where some values for a given target attribute are missing, predict those values.

			b_{11}	b_{12}	b_{13}	b_{14}	b_{15}	b_{16}
			b_{21}	b_{22}	b_{23}	b_{24}	b_{25}	b_{26}
			b_{31}	b_{32}	b_{33}	b_{34}	b_{35}	b_{36}
			b_{41}	b_{42}	b_{43}	b_{44}	b_{45}	b_{46}
a_{11}	a_{12}	a_{13}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}	t_{16}
a_{21}	a_{22}	a_{23}	t_{21}	?	t_{23}	t_{24}	t_{25}	t_{26}
a_{31}	a_{32}	a_{33}	t_{31}	t_{32}	t_{33}	t_{34}	?	t_{36}
a_{41}	a_{42}	a_{43}	t_{41}	t_{42}	t_{43}	t_{44}	t_{45}	t_{46}
a_{51}	a_{52}	a_{53}	t_{51}	t_{52}	?	t_{54}	t_{55}	t_{56}

Figure 5.4: In transductive learning, we can predict any missing values from any known data.

Transductive learning can fill in missing values in a given dataset, but does not generalize to other datasets. It is flexible in the sense that any set of missing values can be predicted from any set of known values. E.g., given a matrix T, whenever the value of T_{ij} is missing, we can use all the known values of T_{kl} to predict it. Figure 5.4 illustrates this setting.

Inductive learning

In inductive learning, we learn a function $f : \mathcal{I} \to \mathcal{O}$ with $\mathcal{I}$ the input space for the function and $\mathcal{O}$ its output space. We then need to commit to a specific $\mathcal{I}$ and $\mathcal{O}$ before learning starts.

Let us first look at bare two-way prediction. We can distinguish row-based and column-based learning. In row-based learning, we can predict a number of target columns from the other columns; without loss of generality we assume $\mathcal{I} = T_{.1} \times \cdots \times T_{.d}$ and $\mathcal{O} = T_{.d+1} \times \cdots \times T_{.n}$. We then predict the values of the last $n - d$ columns from those in the first d columns. This is a standard multitarget learning task, or a standard single target learning task if $d = n - 1$.

We can do exactly the same on the transposed matrix. This corresponds to column-based learning on the original matrix.

In single-decorated two-way prediction, assuming we have A_i attributes but no B_i attributes, with row-based learning we can choose $\mathcal{I} = A_1 \times \cdots \times A_{n_A}$ and $\mathcal{O} = T_{.1} \times \cdots \times T_{.n}$, but also $\mathcal{I} = A_1 \times \cdots \times A_{n_A} \times T_{.1} \times \cdots \times T_{.d}$ and

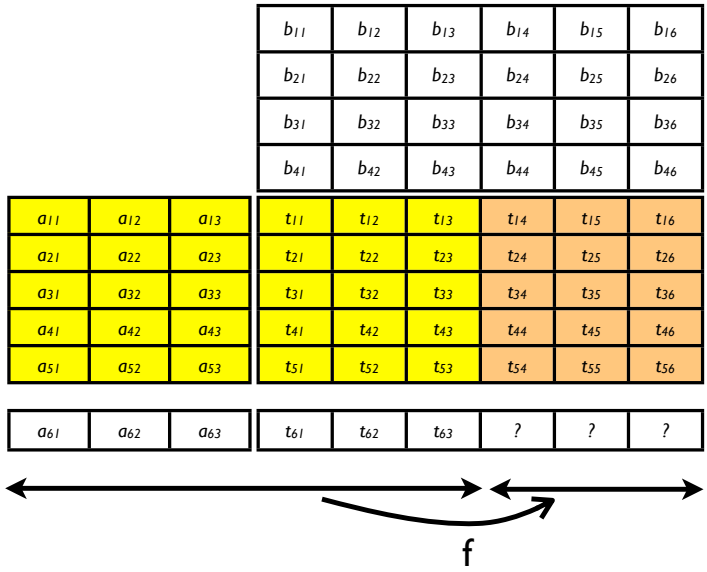


Figure 5.5: We can learn a function f that generalizes over $\mathcal{A}$, predicting target values $t_{.i}$ from $a_{.j}$ and $t_{.k}$. The shaded area indicates the training data for f.

$\mathcal{O} = T_{.d+1} \times \cdots \times T_{.n}$. (In the latter case, we assume that some T_{ij} will be available at prediction time to predict the others.) Alternatively, we can perform column-based learning here, but then the A_i attributes cannot be used.

In double-decorated two-way prediction, we can choose either row-based or column-based learning. In one case we are able to include the A_i attributes as well as the $T_{.i}$ attributes; in the other case we can include the B_i attributes as well as the $T_{i.}$ attributes. Figures 5.5 and 5.6 illustrate the row-based and column-based prediction options.

It may not seem obvious how a single learner can use both the A_i and B_i attributes, and both column- and row-based T attributes. But in fact there is a simple solution: one can simply learn a row-based function $f_{\mathcal{A}}$, a column-based function $f_{\mathcal{B}}$, and then combine these models or their predictions using standard methods from the literature on ensembles and information fusion (Brown et al. 2005).

Another natural way of combining both types of learning is to use propositional learning on the double-decorated two-way learning schema mentioned in Section 5.2.3; we then have $\mathcal{I} = A_1 \times \cdots \times A_{n_A} \times B_1 \times \ldots \times B_{n_B}$ and $\mathcal{O} = T$.

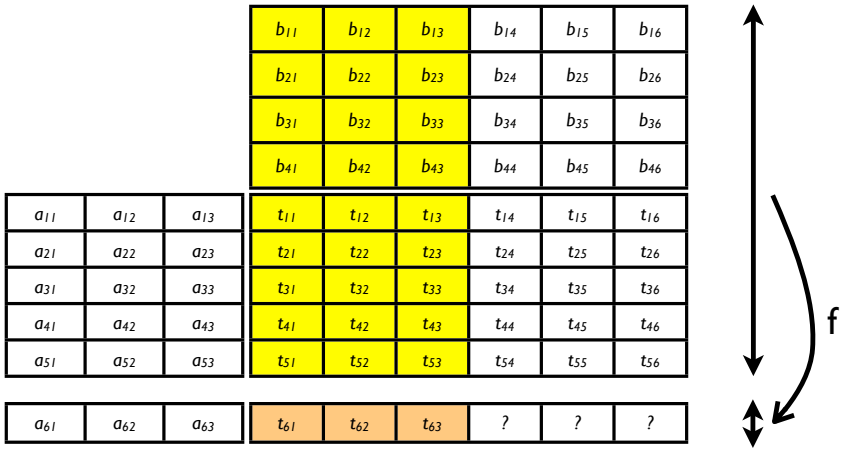


Figure 5.6: We can learn a function f that generalizes over $\mathcal{B}$, predicting target values $t_{i.}$ from $b_{j.}$ and $t_{k.}$. The shaded area indicates the training data for f. In this case, the $t_{i.}$ targets will only become available when the new example becomes available.

Note that the $T_{.i}$ or $T_{i.}$ values cannot be incorporated here (they are not part of the schema). Further, this learning approach makes different assumptions about the structure of the data. This is easiest to see for the degenerate case $Attr(\mathcal{B}) = \emptyset$: a propositional learner that uses the schema from Section 5.2.3 learns a function that maps a single $\mathbf{a}_i$ to a single t value, when in fact multiple t values should be predicted for $\mathbf{a}_i$, namely $t_{i1}, t_{i2}, \ldots, t_{i|B|}$. This shows that the most natural schema from the point of view of data representation is not necessarily the most suitable one for learning.

5.4 The effects of transposition

In the two-way learning setting, rows and columns are to some extent interchangeable. We now discuss some of the implications of this.

5.4.1 What are examples, what are attributes?

A first remark is that, due to the symmetry in the problem, the distinction between examples and attributes becomes blurred. Depending on our view, we can consider rows as examples and columns as attributes, or the other way around. It is worthwhile thinking about how and why we define something to be an example or an attribute.

Arguably, the most natural way of defining attributes and examples is as follows. We learn a function f that will predict certain attributes of new (hitherto unseen) instances. At prediction time, the data schema is unchanged (we have the same input and output attributes), but there are new instances. Thus, if we expect to have to make predictions for new objects of type $\mathcal{A}$, it is natural to see the $\mathcal{A}$ objects as the examples, and the $\mathcal{B}$ objects as the attributes.

An alternative argument is the following. In single-decorated learning, only one type of objects, say, $\mathcal{A}$, have attributes. It is natural, then, to consider these objects the examples. The A_i, possibly together with the T_j, can then be used as predictive attributes, as in Figure 5.5. In the transposed view, where the $\mathcal{A}$-objects are attributes, the information in the A_i can only be interpreted as background information about these attributes. Such information is not handled naturally by most learners.

Thus, in many cases, the problem description naturally decides what is considered an example and what is considered an attribute. However, *the fact that this is our natural notion of examples does not imply that the learner must also be used in this way.* This is an important observation. In the next section, we discuss several applications where a natural view exists on what are the examples, yet a major improvement in predictive accuracy may be obtained by deviating from this view and using a learner in a transposed way.

Let us now look at what happens if we consider the $\mathcal{A}$-objects examples and the $\mathcal{B}$-objects attributes, but have the learner learn from $\mathcal{B}$-objects, using the $\mathcal{A}$-objects as the attributes.

Consider again Figures 5.5 and 5.6. Given a new object $\mathbf{a}$, we need to predict the T_j values for it. First consider the user's point of view: rows are examples, columns are attributes. We get a new example and need to fill in some target values. If we have already learned a function f in the standard way, we can now apply it to predict those values.

Now consider a learner using the transposed view on the data. To this learner, $\mathbf{a}$ is a new attribute, more specifically a target attribute. The goal of this learner is to learn a function f that predicts the components of $\mathbf{a}$ from the corresponding components of other $\mathcal{A}$-objects. The known components of $\mathbf{a}$ are

used as known target values for this learner. Because these components are known only when $\mathbf{a}$ is given, i.e., at prediction time, training this function can only be done at that time. In other words, the inductive learner is used lazily (or, transductively): we need to wait until we see the new example before we can start training. Once the function has been learned, we use it to predict $\mathbf{a}$, and after that we can disregard the function f, since for a new example (in the user's view) $\mathbf{a}'$, a new function will need to be learned (since to the learner this is a new target attribute).

Thus, an inductive learner becomes transductive when we use it on the transposed version of the matrix. Conversely, a transductive method becomes inductive. For instance, a transposed nearest neighbor method, given a column, finds the k nearest neighbors of that column (in user terminology: the nearest attributes). The results of this process can be stored as a model. When a new row $\mathbf{a}$ is presented, any attribute of $\mathbf{a}$ can be predicted by looking up its k nearest attributes (these are stored in the model) and using $\mathbf{a}$'s values for these attributes to compute the prediction.

5.4.2 Multi-target prediction and inductive transfer

Consider the single-decorated learning setting; we repeat the data schema here:

A_1	A_2	$\ldots$	A_{n_A}	T_1	T_2	$\ldots$	$T_{\lvert B\rvert}$

We can handle this in a number of ways. Option 1 is to learn $|B|$ different models, with the i'th model predicting T_i from the A_j. Option 2 is to learn one multi-target model that predicts all T_i at the same time. Option 3 is to construct models that predict one or more T_i from the A_j and (some of) the remaining T_j with $j \neq i$.

If we see the T_i as different learning tasks, then there can be inductive transfer between the tasks in the second and third setting. In the second setting, this transfer is implicit; it is exploited while learning the model (Piccart, Struyf, and Blockeel 2008a). In the third setting, it is explicit: a function that predicts one T_i may use as input another T_j. Of course, this requires that values for T_j will be available at prediction time, since the model uses them as inputs.

Thus, in two-way learning, row-based learning achieves generalization over $\mathcal{A}$ and inductive transfer over $\mathcal{B}$, whereas column-based learning achieves generalization over $\mathcal{B}$ and inductive transfer over $\mathcal{A}$. In two-way learning, generalization and inductive transfer are very similar notions: which term we use depends on what we call examples and tasks.

5.4.3 Transposition switches single-target and multi-target

Assume we have a data matrix, the last row of which is incomplete; we want to predict missing values T_{nj} for $j = k, \ldots, m$. We could learn a row-based function that predicts all T_{nj} from T_{nl}, $l < k$, using a multi-target learner that learns from the data in the first $n-1$ rows (this is Option 1). Alternatively, we could learn a column-based function that predicts T_{nj} from T_{ij}, $i < n$, using the first $k-1$ columns as training examples (Option 2). Depending on the option chosen, we use a multi-target function to predict the multiple labels of a single example, or a single-target function to predict the label of multiple examples. Which of these two works best, may depend on the application. Therefore, in any given application, it is useful to keep these different options in mind.

5.4.4 Summary

This discussion shows that, even when the user has a clear view on what are examples and what are attributes, it may be possible to use a learning system in the transposed way. This has several consequences: (1) inductive learners become transductive and vice versa; (2) attributes describing examples become background information about target attributes, and vice versa; (3) generalization becomes inductive transfer and vice versa; (4) multi-target becomes single-target learning and vice versa.

5.5 Applications

5.5.1 Microprocessor-data

We applied our proposed methods for solving a 2-Way learning problem to Microprocessor data. For this application, the data consists of a set of performance numbers obtained by executing 26 benchmark programs[1] on a number of (different) machines. Based on these data, we wish to predict the performance of each of the machines for new programs. In earlier work (Hoste et al. 2006b), microarchitecture-independent descriptors were used to describe the programs (put in our setting, these form the attributes A_i), and performance was predicted from these descriptors. We here try an alternative approach. When a new program becomes available, the idea is to execute it on a limited number of machines ("predictive machines") and use this information together with the data about the benchmark programs to predict the performance of the

[1] From the SPEC CPU20006 suite, `http://spec.org/cpu2006`.

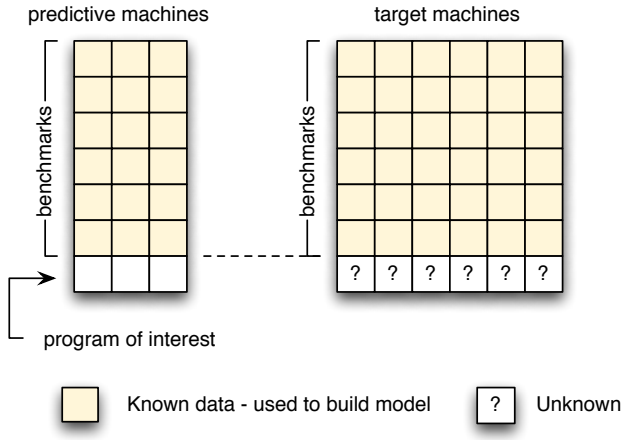


Figure 5.7: Problem statement and terminology

remaining machines ("target machines"). Figure 5.7 illustrates the task. Since we use no descriptors of the machines or the programs, besides the performance numbers, this is a bare two-way learning problem.

As explained in Section 5.2.1, such a problem can be solved in different ways. We here compare row-based and column-based inductive learning. Our goal is to see whether the less natural column-based approach can offer an advantage over straightforward row-based learning.

We used 3 different machine learning algorithms from the Weka collection (Hall et al. 2009): a multi-layer perceptron (MLP), a Support Vector Machine using Sequential Minimal Optimization (SVM-SMO), and linear regression (Linear Regression). The fourth algorithm to which we compare our results consist of non-negative matrix factorization as proposed by Lee and Seung (2000).

This method factorizes the matrix T in two non-negative matrices: $T \approx W \cdot H$, minimizing a cost function such as the euclidean distance $\| T - WH \|$. The matrices express latent factors that explain observed ratings and the method is relatively robust for sparse data (Zhang et al. 2006).

Each algorithm was run both row-based and column-based. Default parameters were used for each algorithm.

The machines in the data set are grouped by processor type, e.g. AMD Turion, Intel Core 2, etc. There are 17 such groups. Because machines in one group are very similar, we use a leave-one-group-out approach: we run 17 experiments, each times using the machines in one group as target machines and the others

Table 5.2: Spearman Rank correlation for the predicted microprocessor data.

	Row-based	Column-based
Neural Network	0.82	**0.93**
SVM	0.77	**0.90**
Linear Regression	0.82	**0.91**
Matrix Factorization	0.85	0.85

as predictive machines.

Each experiment itself uses a leave-1-benchmark-out evaluation: full information on 25 benchmarks is used to predict the performance of the target machines for the remaining benchmark.

We report the Spearman Rank Correlation coefficient, averaged over the benchmarks. This shows how well the method is able to rank the machines; it is relevant when we want to select the fastest machine for a given program, which is indeed the goal of this work.

Table 5.2 shows that column-based prediction yields much better results than row-based prediction. One way to interpret this is that generalization over machines (or inductive transfer, in the multi-task learning interpretation) is easier than generalization over programs. Put differently, given a number of benchmarks and machines, it is difficult to predict how well the machines will perform for a new benchmark, but it is relatively easy to predict how well a new machine will perform on the given benchmarks. Until now, this problem had not been addressed in this manner.

Of particular interest is the result of the Matrix Factorization, which can only be applied to two-way datasets. As can be seen in table 5.2, the MF algorithm outperforms all other algorithms for the row-based approach but the MF method is outperformed by the other algorithms in the column-based approach. The MF method does not benefit from a transposition.

When we make the link with the multitask framework and the notion of inductive transfer (see Chapter 3), we can see that inductive transfer among different tasks is very strong here, the extent that the “signal” (the amount of relevant information) in related tasks is stronger than the “signal” in the inputs.

This application has a few other peculiarities and problems. For example, the selection of the set of predictive machines is important for the accuracy of the model. This brings us to the task of selecting an optimal set of predictive machines. We discuss the application in detail in the following chapter.

Table 5.3: Mean Squared Error (MSE) for the ecological dataset.

	Row-based	Column-based
Neural Network	1.127	**0.9**
SVM-SMO	2.43	**1.46**
Linear Regression	**3.07**	4.04
Matrix Factorization	1.40	1.40

5.5.2 Ecological data

Next, we consider an ecological application (Blockeel, Džeroski, and Grbović 1999). Biologists take samples of river water to measure its quality, recording quantities of a number of micro-organisms, and physico-chemical parameters (such as oxygen concentration) in these samples. The goal is to learn to predict the physico-chemical parameters (PCP, for short) from the micro-organism quantities.

The data matrix consists of 1060 rows that represent samples, and 852 columns that represent measurements (836 on an ordinal scale, for micro-organisms, and 16 on an interval scale, for PCP). The natural view is to consider the samples as examples. Even then, we have two interpretations in our two-way learning framework: (1) we can see the organism data as A_i attributes and the PCP as targets, or (2) we can see all of them as targets (then $Attr(\mathcal{A}) = \emptyset$ and the $\mathcal{B}$-objects are simply "measurements of any type").

The original work used interpretation 1, which gives a standard multi-target problem with 836 predictive variables and 16 target variables. Here, we use interpretation 2; we then have a bare two-way learning problem, and we can perform both row-based and column-based learning.

We use the same three algorithms as for the micro-processor data. We randomly assign rows to the training (80%) and test (20%) set. In row-based learning, we learn (on the training set) 16 single-target functions f that predict a PCP from the organism data, and validate their predictions on the test set. In column-based learning, we learn single-target functions f (each targeting a separate row in the test set) on the organisms, and validate them on the PCP. Figure 5.8 illustrates the setup. Results are shown in Table 5.3.

We see that column-based predictions outperforms row-based prediction for the MLP and SVM, while row-based prediction works better for linear regression. This shows that which direction works best for prediction may depend on the inductive bias of the learner. The transposed view gives more informed predictions here, but only relatively complex models can exploit this.

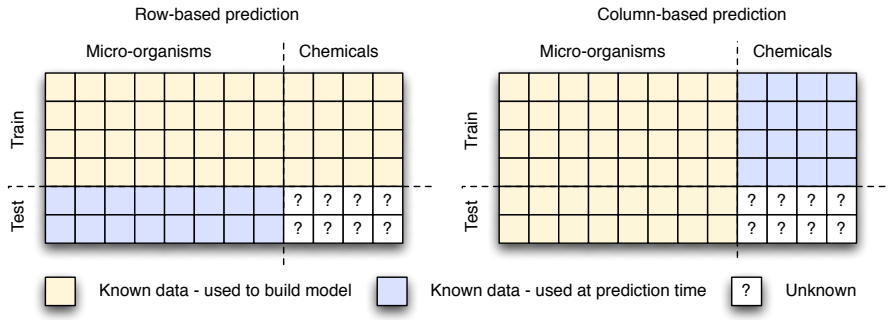


Figure 5.8: Problem statement and evaluation method

5.6 Conclusions

We propose a new setting for machine learning that we call *two-way predictive learning*. The setting is characterized by the existence of two types of data elements, pairs of which are labeled with target values, and it is these target values that we need to predict. The two-way learning setting is encountered in many application domains, including recommender systems, microarray data analysis, microprocessor design, and ecological data mining. It covers multi-target learning as a special case, and is itself a special case of relational learning. The two-way learning setting has many pecularities that motivate a separate study of this setting, as opposed to simply transforming such learning problems into standard learning problems. It forces us to think about the meaning of examples and attributes, which itself sheds new light on a number of other concepts, such as inductive transfer, or the difference between inductive and transductive learning.

Experiments on two different application domains demonstrate the usefulness of this discussion: by running an inductive learner on the transposed data matrix, one can obtain better predictive results. While this approach is practically very simple, it is not straightforward to practitioners, partially because it requires a view of the data that is often unnatural (thinking of attributes as examples and vice versa), and partially because it requires one to use an inductive learner transductively (the learning process can only be started once a new test instance has arrived). Our experimental results show, however, that this alternative approach can yield important performance gains.

There are many opportunities for future work. Apart from the already

demonstrated potential advantage of transposing the data matrix, the new setting may make it possible to: use background knowledge about attributes in a principled way; combine row-based and column-based prediction for even better performance; develop new transductive learners based on existing inductive learners; develop new methods for achieving inductive transfer between tasks; transfer solutions between areas that up till now seemed unrelated; and more. Also, two-way learning could be generalized to n-way learning, making it directly applicable to the star schemas that are popular in data warehousing.

Chapter 6

Application: Processor Performance Prediction

6.1 Introduction

We concluded the previous chapter with a short description of two 2Way problems and how they can benefit from data transposition. This chapter takes a closer look at the first application: predicting the performance of CPU's.

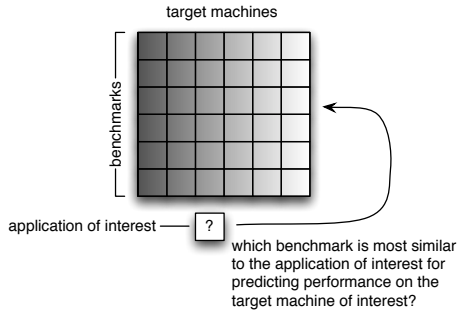
Current practice in benchmarking commercial machines is to run industry-standard benchmarks and report their performance numbers. This practice is adopted by various benchmarking consortia and corporations such as EEMBC[1] for embedded systems, TPC[2] for database systems, and SPEC[3] for high-performance computer systems. The information obtained from these benchmarking experiments provides valuable information for comparing existing commercial machines across a broad range of applications. For example, SPEC provides performance results for various benchmarks from several application domains such as compute-intensive workloads, Java workloads, graphics, web servers, mail servers, network file systems, etc.

Although these benchmarking efforts enable users to compare computer system performance across vendors for different types of workloads, they do not provide insight with respect to which computer system performs best for a given

[1] `http://www.eembc.org`
[2] `http://www.tpc.org`
[3] `http://www.spec.org`

application of interest that is not part of the benchmark suite. In particular, it is unclear which performance numbers to base a purchasing decision on, i.e., it is unclear which benchmark is most similar to an application of interest. This is a ubiquitous and long-standing problem in benchmarking that affects various markets of the computer industry. For example, a phone company needs to decide which processor to include in its next-generation cell phone, however, its software may be very different from what the EEMBC benchmark suite provides. In addition, the phone company most likely will not be willing to distribute its proprietary software to third-party hardware vendors. Similarly, an Internet-service provider or a supercomputer host needs to decide which processors to provide in the data center, however, the software that will be run may be very different from what SPEC provides.

(a) Approach in prior work



(b) Data transposition

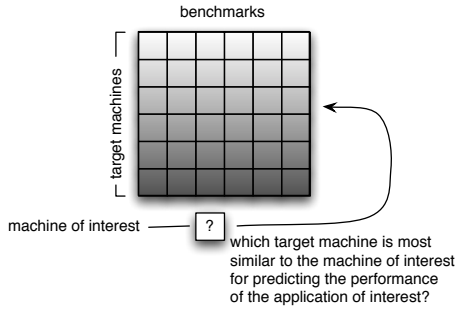


Figure 6.1: High-level conceptual comparison between (a) the approach in the work of Hoste et al. (2006a) and (b) data transposition.

Prior work in this area by Hoste et al. (2006a) approached this problem by identifying a benchmark or a number of benchmarks in the benchmark suite

that are most similar to the application of interest, see Figure 6.1(a). An inherent problem with this approach is that the application of interest may exhibit execution characteristics that are very different from the benchmarks included in the benchmark suite (i.e., the application of interest may be an outlier with respect to the benchmark suite), hence, it is unclear how useful the benchmark suite is for making an accurate performance prediction.

In this chapter, we propose a very different technique. The problem at hand consists of a 2-Way learning problem. As discussed in Chapter 5, this allows us to transpose the data-matrix. Such a data transposition results in solving the dual problem of finding the machine that is most similar to the target machine for predicting performance for an application of interest, see Figure 6.1(b). In other words, instead of finding the benchmark that is most similar to the application of interest for predicting performance on a target machine of interest, data transposition aims at finding the predictive machine that is most similar to the target machine of interest for predicting performance for the application of interest. While both approaches basically solve the same problem, we demonstrate that the data transposition leads to more accurate performance predictions, the fundamental reason being that the data transposition enables capturing outlier workload behavior better. Intuitively speaking, an application of interest that exhibits outlier behavior on one machine is also likely to exhibit outlier behavior on another machine. An empirical model then extrapolates outlier behavior across machines.

Our approach assumes a (potentially large) set of target machines, to which the user has no access but for which benchmarking results are available for a (limited) set of benchmarks, e.g., performance numbers published by a benchmarking consortium such as SPEC. Further, a limited number of so-called predictive machines are assumed to be available to the user on which both the benchmarks and the application of interest can be run. A model learned on the transposed data then predicts the performance of the application of interest on each of the target machines. It does so based on the published performance numbers for the target machines and the benchmarks along with a limited number of measurements that need to be done on the predictive machines using both the benchmarks and the application of interest; the method does not require executing the application of interest on the target machine.

Our experimental evaluation using SPEC CPU2006 and performance numbers for 117 commercial machines demonstrates the method's accuracy. Data transposition predicts the ranking of the commercial machines with a correlation coefficient of 0.93 compared to the ranking obtained with measured performance numbers, whereas prior art achieves a ranking of 0.86. The top-1 machine according to data transposition yields a 1.2% performance deficiency on average (24.8% max) compared to the real top-1 machine for the given the application

of interest. Prior work in this area by Hoste et al. (2006a) is accurate as well for most benchmarks, except for outlier workloads for which we observe deficiencies over 100%. Furthermore, we demonstrate the method's ease of use: we find that only a few predictive machines are sufficient for making accurate performance predictions.

This chapter is organized as follows. We briefly describe prior work in this area in the next section. In Section 6.3, we then present how the data is a 2Way dataset, hence enabling data transposition. We elaborate on how it advances beyond prior work and discuss potential applications in Section 6.4. Section 6.5 discusses our experimental setup, and we presents the results on the accuracy of data transposition in Section 6.6. Finally, we discuss related work in Section 6.7 and conclude in Section 6.8.

6.2 Prior Work

The problem that motivates this work can be summarized as follows. Assume we have an application of interest for which we want to rank a set of commercial machines and predict the best machine or the top-n best performing machines. We therefore rely on an existing performance database that is comprised of performance numbers for a number of benchmarks and machines. The approach taken by prior work was to exploit the similarity between the application of interest and the industry-standard benchmarks across these machines, so that an informed estimate can be made for the performance of the application of interest across the target machines.

In particular, Hoste et al. (2006a) use performance scores of a standardized benchmark suite on the target machines of interest, and in addition, they measure a set of microarchitecture-independent characteristics for the application of interest which they relate to the benchmarks in the standardized benchmark suite. These microarchitecture-independent characteristics capture the inherent program behavior that is unbiased towards a particular microarchitecture. They rely on the notion of similarity between the application of interest and the benchmarks (in terms of their microarchitecture-independent characteristics) to predict the performance of the application of interest. The key issue in this approach is to determine how differences in microarchitecture-independent characteristics translate into performance differences. They use a genetic algorithm to learn this relationship across a variety of machines. In short, Hoste et al. use the standardized benchmarks as proxies for the application of interest based on behavioral similarity.

The method proposed in this chapter is dual to Hoste et al.'s approach. Whereas they identify the benchmark(s) most similar to the application of interest to predict performance on a target machine, our approach transposes the problem and identifies the predictive machine most similar to the target machine to predict target machine performance for the application of interest. The intuition behind Hoste et al.'s approach is that workloads exhibiting similar inherent program behavior are likely to yield similar performance across a range of machines. This approach is effective for applications of interest that show similarity to the benchmarks in the benchmark suite, however, for applications of interest that are dissimilar to any of the benchmarks, so-called outliers, the method is unlikely to yield accurate performance predictions. Data transposition on the other hand overcomes this inefficiency by building on the notion of machine similarity: an application of interest that is dissimilar to any of the benchmarks and that may yield different performance on a particular machine, is also likely to yield different performance on other machines. In other words, an application of interest exhibiting outlier performance on a predictive machine is likely to exhibit outlier performance on the target machines.

In addition to the observation that data transposition leads to more accurate predictions for applications of interest that are outliers compared to the benchmarks in the benchmark suite, it does not require time-consuming profiling runs for collecting microarchitecture-independent program characteristics as in the Hoste et al. approach. Through data transposition, a limited number of real hardware runs on the predictive machines is sufficient for making accurate predictions on the target machines. This makes data transposition both faster and more practical.

6.3 Data Transposition

We first introduce some terminology and definitions.

6.3.1 Data set and definitions

We starts from the data set as shown in Figure 6.2. In both data sets, the rows represent the benchmarks and the columns represent the machines.

This is a 2Way data set, as defined in Section 5.2, with the benchmarks and machines corresponding to the A and B objects respectively. Hence, we can transpose the dataset to generalize over machines instead of benchmarks.

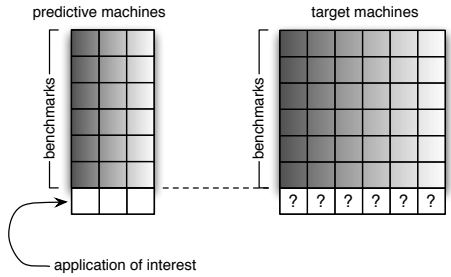


Figure 6.2: Problem statement and terminology

The part of the data set on the left in Figure 6.2 comprises performance numbers for all the benchmarks as well as for the application of interest for the so-called *predictive machines*. These machines are assumed to be available to the user, i.e., the user can run the application of interest as well as the industry-standard benchmarks on these predictive machines to collect performance numbers. Typically, there are fewer predictive machines than *target machines*. The target machines are not available to the user, hence we only have performance numbers for the benchmarks on the target machines, and not for the application of interest.

The part of the data set on the right in Figure 6.2 is provided by a benchmarking consortium; in fact, we use performance numbers from SPEC CPU2006 in our setup, as we will explain later. It comprises performance numbers for all benchmarks and all target machines. The goal now is twofold. First, we want to predict the performance of the application of interest on each of the target machines. Second, we want to both rank these machines and identify the best performing target machine(s) for the application of interest.

6.3.2 Models for performance prediction

We explore two flavors of empirical models to which we apply our data transposition method, namely linear regression and neural networks.

Linear regression

Linear regression builds a linear regression model for each target machine with each predictive machine, see also Figure 6.3. The regression model that yields the best fit across the predictive machines for a given target machine is retained; this model is subsequently used to predict the performance of the application

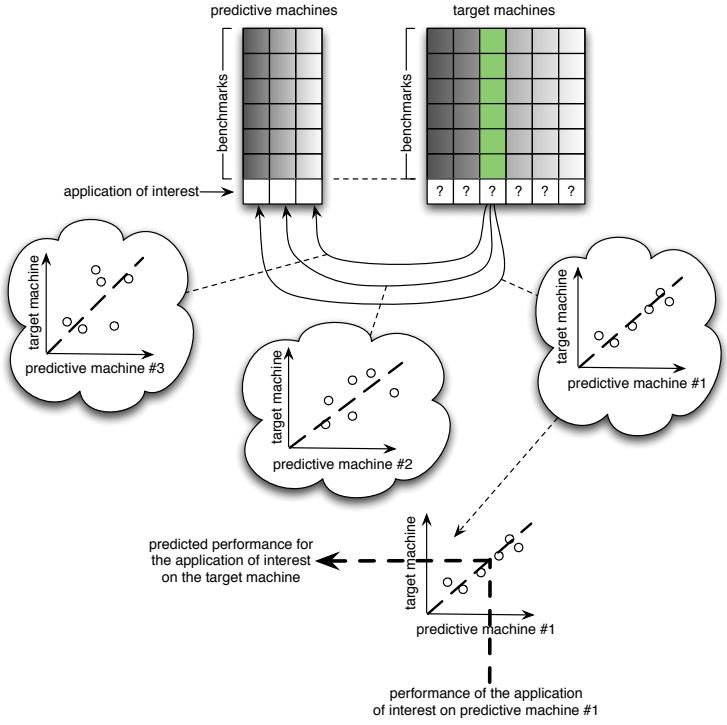


Figure 6.3: Performance prediction through data transposition using linear regression.

of interest on that particular target machine. Put differently, the performance for that target machine correlates best with the performance of the chosen predictive machine. Figure 6.3 illustrates the methodology through an example: three regression models — since there are three predictive machines — are built for target machine #3. In this example, we predict the performance for target machine #3 with the regression model obtained using predictive machine #1 — because predictive machine #1 yields the most accurate linear model for target machine #3. This procedure is repeated for all target machines, which enables us to rank the target machines based on the predicted performance numbers. This ranking provides the relative ordering of target machines for the application of interest; the top-1 machine is predicted to yield the highest relative performance.

Note that this approach is very similar to the one described in Section 4.7.

Neural networks

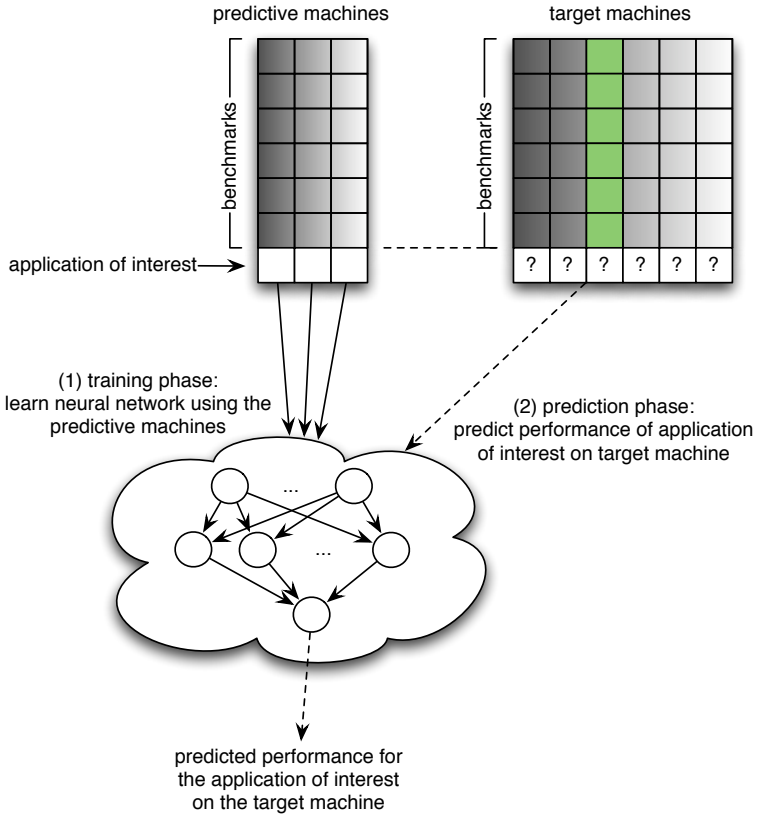


Figure 6.4: Performance prediction through data transposition using neural networks.

Neural networks can also be used for performance prediction through data transposition. Neural networks have the advantage over linear regression models that they can model non-linear relationships. Figure 6.4 illustrates how this is done. The input to the neural network is the performance of the benchmark applications, and the output is the predicted performance for the application of interest, on the target machine. We consider a multi-level perceptron in this work. We train a neural network using the set of predictive machines. Training the neural network involves inputting the performance numbers of the benchmarks on the predictive machines, and expecting the performance for the application of interest at the output. The training algorithm then learns

the neural network to predict the performance for the application of interest based on the performance numbers for the benchmarks. Model training is done using performance data on the predictive machines only. Once the model is trained, it is used to predict performance for the application of interest on each of the target machines. Intuitively speaking, the neural network learns how the performance of the application of interest relates to the other benchmarks. The implicit assumption is that this relationship is similar on the target machines as it is on the predictive machines.

6.4 Potential Applications

We envision several potential applications for data transposition.

Guiding purchasing decisions When purchasing a new computer system, the customer has to rely on published performance numbers as provided by benchmarking consortia and corporations such as SPEC, EEMBC and TPC. These numbers however only quantify performance for a number of a standardized benchmarks. As a result, it is unclear to which benchmark(s) the application of interest is most similar, and by consequence it is unclear which performance number(s) to base a purchasing decision on. Typically, these decisions are driven by average performance figures across the entire benchmark suite, or they are typically based on presumed similarities across applications from the same application domain. Data transposition provides a methodology for ranking machines, which enables making better purchasing decisions, as we will demonstrate later in the evaluation section of this chapter.

Performance prediction of unavailable hardware Prototype hardware or expensive hardware may be hard to obtain for experimentation and measurement. Data transposition provides a solution to performance evaluation on unavailable hardware, i.e., by comparing the performance for the application(s) of interest against a benchmark suite (which is to be run only once on the expensive prototype hardware), useful performance predictions and assessments can be obtained.

Fast design space exploration Simulation-based processor design space exploration is extremely time consuming. Cycle-accurate simulators typically incur a slowdown compared to real hardware of at least 5 orders of magnitude. Hence, simulating one minute of real execution time takes at least two months of

simulation time for evaluating a single microarchitecture design point. Obviously, exploring and refining a microarchitecture design at these speeds is infeasible. Data transposition may help speedup design space exploration: simulating a number of representative benchmarks in detail on the slow simulator is sufficient to predict the performance of other benchmarks and applications.

Task scheduling on heterogeneous systems Research into heterogeneous computer systems is gaining importance at different levels in the computing range. For example, the composition of computing nodes in a grid or data center may be heterogeneous due to upgrades or by design; heterogeneity may also be considered to increase the energy efficiency of a multi-core processor design (Kumar et al. 2003); or, heterogeneity may emerge because of chip technology process variability in a homogeneous multi-core processor (Teodorescu and Torrellas 2008). An important question in heterogeneous system design is how to schedule the applications for maximizing overall system performance. Data transposition may be an enabler to drive the scheduling algorithm on heterogeneous systems by providing performance predictions for each of the computing nodes; the scheduling algorithm can then use these performance predictions to yield better schedules.

6.5 Experimental Setup

6.5.1 Benchmarks and platforms

Our data set contains reported performance numbers for a set of industry-standard benchmarks on a number of commercial machines. In particular, for this study, we use performance numbers reported for the SPEC CPU2006 benchmark suite[4] which includes 29 integer and floating-point performance benchmarks. We use the *speed* ratios with base optimization, i.e., `SPECint_base2006` and `SPECfp_base2006`; these speedup numbers are relative to a reference SUN Ultra5_10 workstation with a 296MHz SPARC processor.

We selected 117 commercial machines out of the 1K+ machines that are available on the SPEC website as of Dec 2009. These 117 machines were chosen such that they are as diverse as possible in terms of their (micro)architecture, instruction-set architecture, technology node, etc., see Table 6.1. For each family of architecturally similar processors, we pick a number of machines by CPU nickname — different CPU nicknames reflect differences in microarchitecture,

[4] `http://www.spec.org/cpu2006/`

Processor family	*CPU nickname*
AMD Opteron (K10)	Barcelona, Istanbul, Shanghai
AMD Opteron (K8)	Santa Rosa, Troy
AMD Phenom	Agena, Deneb
AMD Turion	Trinidad
IBM POWER 5	POWER5+
IBM POWER 6	POWER6
Intel Core 2	Allendale, Conroe, Kentsfield, Merom-2M, Penryn-3M, Wolfdale, Yorkfield
Intel Core Duo	Yonah
Intel Core i7	Bloomfield XE
Intel Itanium	Montecito
Intel Pentium D	Presler
Intel Pentium Dual-Core	Allendale
Intel Pentium M	Dothan
Intel Xeon	Bloomfield, Clovertown, Conroe, Dunnington, Gainestown, Harpertown, Kentsfield, Lynnfield, Tigerton, Tulsa, Wolfdale-DP, Woodcrest, Yorkfield
SPARC64 VI	Olympus-C
SPARC64 VII	Jupiter
UltraSPARC III	Cheetah+

Table 6.1: The machines considered in this study sorted by processor family. Our selection contains 3 machines of each CPU nickname.

chip technology, cache sizes, bus speed, etc. For each nickname we include three machines. For example, our dataset includes 9 AMD Opteron K10 machines in total, see also the first line in Table 6.1: 3 machines with a Barcelona CPU, 3 with an Istanbul CPU, and 3 machines with a Shanghai CPU.

As we have outlined in Section 6.3, we require the data set to be split into two groups: a set of predictive machines versus a set of target machines. Because the selection of predictive machines may have significant impact on the overall accuracy, we will consider different sets of predictive machines throughout our experiments. In all of our experiments, we use a cross-validation setup to allow for a fair evaluation of our methodology. This means there is no overlap between the set of predictive versus target machines: for a given set of predictive machines — a processor family in this study — we remove those machine types from the set of target machines, see also Figure 6.5.

Additionally, we also consider a leave-one-out methodology with respect to the

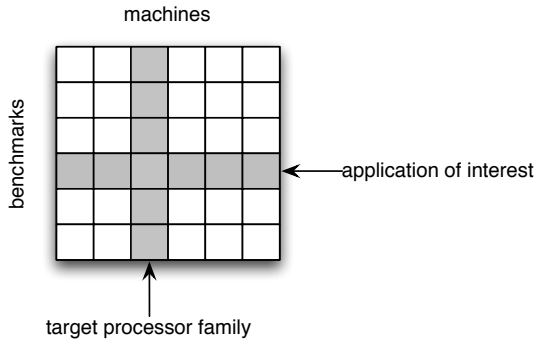


Figure 6.5: Cross-validation and leave-one-out setup: we eliminate the target machine and application of interest from the dataset when training the empirical performance models.

benchmarks. This means we pick a single benchmark out of the benchmark suite — this is our application of interest — and we build a prediction model using the remaining 28 benchmarks, see also Figure 6.5. Once the model is built, we compare the predicted performance for the application of interest against its measured performance on each of the target machines.

6.6 Evaluation

We evaluate performance prediction through data transposition in three different settings: (i) targeting a processor family based on performance numbers for other processor families, (ii) targeting newer machines based on a predictive set of older machines, and (iii) limiting the set of predictive machines. Throughout the evaluation we refer to the prior work proposed by Hoste et al. (2006a) as *GA-kNN*, as it involves a genetic algorithm (*GA*) to learn how to weight microarchitecture-independent workload differences to performance differences and then derives a performance prediction based on the k nearest neighbors (*NN*) in the workload space; we assume 10 neighbors in our setup ($k = 10$). We use the WEKA[5] v3 Multilayer Perceptron implementation with default settings as the neural network. We refer to the data transposition approach using the ^{T} superscript, and we refer to the linear regression approach as NN^T (it selects the best fitting predictive machine or 'nearest neighbor' for making a prediction) and the neural network approach as MLP^T (as it uses a multi-level perceptron).

[5] `http://www.cs.waikato.ac.nz/ml/weka/`

	NN^T	MLP^T	*GA-kNN*
Rank correlation	0.85 (**0.67**)	**0.93** (**0.71**)	0.86 (0.59)
Top-1 error	11.9 (156.7)	**1.21** (**24.8**)	7.30 (104)
Mean error	**4.04** (**31.81**)	**1.59** (**19.4**)	6.25 (51.34)

Table 6.2: Performance comparison for the different methods using processor-family cross-validation. Numbers in bold represent cases where data transposition outperforms the previously proposed *GA-kNN* method. Average numbers are presented; the numbers between brackets give the worst case.

6.6.1 Metrics

We use three different metrics to quantify the accuracy of data transposition: (i) target machine ranking, (ii) the top-1 performance prediction error, and (iii) the average performance prediction error. We now briefly discuss each of these metrics.

Ranking measures the ability of the method to predict the relative ranking of the target machines. This is done by first predicting the performance for the application of interest on each target machine. We then rank the machines according to the predicted performance for the application of interest. The predicted ranking is then compared to the actual ranking using the Spearman rank correlation coefficient. The correlation coefficient ranges between 0 and 1; a correlation coefficient of one means a perfectly predicted ranking.

Once a predicted ranking has been obtained, we compare the speedup of its top machine with the speedup of the actual top machine, yielding the top-1 error. In Section 6.4, we discussed various applications, including the case for guiding purchase decisions. The top-1 error indicates what the loss in performance would be if a purchase is following the performance prediction.

The third metric is the average prediction error across all target machines and benchmarks.

6.6.2 Predicting another processor family

In the first experiment, we consider a single processor family as the set of target machines, and we use the machines from the other families as predictive machines. Following the cross-validation approach outlined in Section 6.5, we have 17 predictive/target pairs on top of the leave-one-out cross-validation at the benchmark level. The results are summarized in Table 6.2; we aggregated the results across both the target machines and the benchmarks. Average numbers

are reported as well as worst-case results; the worst case numbers reports the worst-case result across all target machines and benchmarks, and are shown between brackets. The MLP^T approach beats the other approaches on all three metrics. Compared to *GA-kNN*, the NN^T approach has a slightly lower average rank correlation (0.85 vs. 0.86), but does significantly better in the worst case (0.67 vs. 0.59). We obtain a similar result for the average error, where NN^T outperforms *GA-kNN* (4.04 vs. 6.25). These results indicate that approaches based on data transposition are able to outperform the state-of-the-art when predicting the performance of machines with an unseen architecture.

Figures 6.6 and 6.7 show the same data on a per-benchmark basis for the rank correlation coefficient and the top-1 prediction error, respectively. These results show that data transposition is better capable of accurately predicting performance of outlier benchmarks compared to the previously proposed *GA-kNN* approach (Hoste et al. 2006a). In particular, the leslie3d benchmark is an outlier benchmark compared to the other benchmarks in the benchmark suite, and the *GA-kNN* method has difficulty making an accurate prediction (i.e., rank correlation coefficient of 0.59). Data transposition on the other hand improves the predicted ranking to 0.92. Similarly, for the cactusADM and libquantum benchmarks, data transposition using neural networks is the most accurate approach for predicting the top-1 performing machine, see also Figure 6.7; data transposition using linear regression and the prior state-of-the-art approach are highly inaccurate. These benchmarks are outliers with higher-than-average SPEC scores, and yields the highest performance on an Intel Xeon Gainestown system. The namd and hmmer are outliers at the opposite side of the spectrum: these benchmarks have lower-than-average SPEC scores, and yield the highest performance on Intel Montecito processor systems. Both data transposition and the prior work are accurate at estimating performance for these benchmarks.

Data transposition using neural networks is more accurate than using linear regression. This is apparent from both Figures 6.6 and 6.7. The reason is that neural networks can model non-linear relationships, as mentioned before. Both the average and minimum rank correlation coefficients are higher for the neural networks compared to linear regression. Also, data transposition using neural networks is very accurate when it comes to predicting the top-1 machine. Whereas *GA-kNN* and data transposition through linear regression yields prediction errors that are higher than 100%, data transposition using neural networks brings the error down to 25% at most for one of the benchmarks; for the other benchmarks, data transposition using neural networks predicts the top-1 machine with (near) perfect accuracy.

MLP^T

	2008	2007	older
Rank correlation	**0.93** (**0.71**)	0.80 (0)	0.77 (0.49)
Top-1 error	**3.78** (**50**)	9.23 (119)	6.84 (**43**)
Mean error	**5.50** (**65.61**)	**8.10** (**70.79**)	**8.36** (**64.89**)

NN^T

	2008	2007	older
Rank correlation	**0.92** (**0.76**)	0.82 (0.37)	0.74 (0.31)
Top-1 error	**2.17** (**43**)	**4.31** (**92**)	**2.07** (**29.3**)
Mean error	**4.38** (**35.16**)	**9.22** (**82.13**)	**9.22** (**53.34**)

Table 6.3: Performance comparison for the different methods predicting the performance for machines from 2009 using older machines. Numbers in bold outperform the previously proposed *GA-kNN* method; the numbers between brackets report the worst case result.

6.6.3 Predicting future machines

The previous experiment did not take into account the release date of the different machines and aimed at predicting the performance of a processor family based on other processor families, irrespective of their release date. A relevant case in practice might be to predict the performance for a future processor family. To this end, we now limit the target machines to those released in 2009, using machines that were released before 2009 only as the predictive set. We distinguish three possibilities for the predictive set: the machines released in 2008, 2007 and pre-2007. This distinction allows us to see how far into the future a set of predictive machines can reliably predict performance. The results are summarized in Table 6.3, and we briefly discuss them now.

Predicting one year into the future, (i.e., using the 2008 machines to predict the 2009 machines), works best using the proposed data transposition approaches. On all three metrics, data transposition outperforms the previous state-of-the-art *GA-kNN*. We obtain an increase in the Spearman rank correlation from 0.87 to 0.93 and a reduction in top-1 and mean error from 6.84 and 10.75 to 2.17 and 4.38, respectively. NN^T does somewhat better than MLP^T, which seems to suggest that MLP^T is more sensitive to the training data than NN^T.

If we go back one more year (i.e., using the 2007 machines to predict performance of the 2009 machines), *GA-kNN* achieves a better ranking. Note however that

$MLP\hat{}T$

	10	5	3
Rank correlation	**0.9**	**0.89**	**0.89**
Top-1 error	**6.17**	**2.79**	**3.04**
Mean error	**5.53**	**4.93**	**5.16**

$NN\hat{}T$

	10	5	3
Rank correlation	0.87	0.81	0.81
Top-1 error	**2.17**	**5.49**	**5.49**
Mean error	**5.17**	**6.00**	**6.05**

Table 6.4: Performance comparison for the different methods predicting the performance for machines from 2009 using a small subset from the 2008 machines. The numbers in bold indicate cases where data transposition outperforms the previously proposed *GA-kNN* method.

this method does not rely on data from these predictive machines, and takes only the target machines and the benchmark characteristics into account. Even though the predicted ranking correlates better with the actual ranking under *GA-kNN*, data transposition does better on the mean error score. This is the case even for machines predating 2007. The reason why *GA-kNN* performs relatively better than data transposition when predicting further in the future is that *GA-kNN* bases its prediction on a microarchitecture-independent characterization of the workloads only, which is independent of time. Data transposition on the other hand makes a prediction based on historic performance numbers from older machines..

From this we conclude that data transposition (both NN^T and MLP^T) are more accurate when predicting into the near future than *GA-kNN*. However, when predicting in the distant future (two years and more ahead), then data transposition using linear regression (NN^T) is more accurate than using neural networks. Even though *GA-kNN* achieves a better ranking when predicting in the distant future, both the top-1 and mean error are smaller for data transposition using linear regression (NN^T).

6.6.4 Limited number of predictive machines

We now consider the case where we need to make a performance prediction based on a limited number of predictive machines. The reason is that we need to run benchmarking experiments on the predictive machines for data transposition to work, as explained previously. Limiting the number of predictive machines increases the practicality of method. To evaluate how well data transposition works considering a limited number of predictive machines, we set up the following experiment. The target machines all have been released in 2009, whereas the predictive machine are a subset of the machines released in 2008. We use three subset sizes: 10, 5 and 3.

The results of this experiment are summarized in Table 6.4. As expected, prediction accuracy decreases with a limited number of predictive machines, however the effect is relatively small. The MLP^T method is most robust to a decrease in the number of predictive machines: even with three predictive machines only does MLP^T outperform *GA-kNN*. NN^T suffers more from a limited number of predictive machines. *GA-kNN* is able to rank the machines better than NN^T when we use 5 or less predictive machines, but performs worse compared to NN^T based on the mean error and the top-1-error. These results demonstrate that data transposition (using neural networks) is accurate even when having access to a limited set of predictive machines only, which makes the method practical to use.

6.6.5 Selecting predictive machines

Now that we know that a limited number of predictive machines is sufficient, the question is how to select these predictive machines. An easy-to-implement approach is to select predictive machines randomly. Another approach may be to select predictive machines such that they maximize the coverage relative to the target machines. In other words, by choosing a diverse set of predictive machines one may maximize the likelihood of finding a similar (close-enough) (set of) predictive machine(s) for each target machine. This may improve the accuracy of the performance prediction, and hence, it could be viewed of as the best possible approach. We implement this approach by choosing the predictive machines through k-medoid clustering, which randomly selects k cluster centers (predictive machines), initially, and groups all the remaining machines to their closest cluster. Assigning machines to clusters changes the centroids of the clusters, hence, different machines emerge as cluster centers (i.e., a new set of predictive machines is constructed), after which new cluster centers need to be determined, etc. This process is iterated until convergence or steady-state, i.e., all machines are assigned to a cluster and cluster membership does not change

across iterations of the algorithm. The cluster centers then are the predictive machines. This results in a diverse set of machines, for example an Intel Core 2, Pentium D Presler, Xeon Gainestown and a SPARC 64 VII when selecting 4 predictive machines. Figure 6.8 shows the goodness of fit for random selection versus k-mediod clustering as a function of the number of predictive machines for MLP^T. The key observation from this graph is that k-medoid clustering outperforms random selection by over a factor two, e.g., two predictive machines selected by k-medoid clustering achieve a better fit (0.714) than five randomly selected machines (0.705).

6.7 Other Related Work

We now discuss other related work beyond the prior work discussed in Section 6.2.

6.7.1 Empirical performance modeling

Empirical modeling leverages statistical inference and machine learning techniques such as regression modeling or neural networks to automatically learn a performance model from training data. Joseph, Vaswani, and Thazhuthaveetil (2006b) apply linear regression to processor performance analysis: they build linear regression models that relate micro-architectural parameters (along with some of their interactions) to overall processor performance. Joseph et al. only use linear regression to test microarchitecture design parameters for significance, i.e., they do not use linear regression for predictive modeling.

Linear regression assumes that the response variable behaves linearly with its input variables. This assumption is often too restrictive. Lee and Brooks (2006) advocate spline-based regression modeling in order to capture non-linearity. A spline function is a piecewise polynomial used in curve fitting. A spline function is partitioned in a number of intervals with different continuous polynomials.

An artificial neural network is an alternative approach for building an empirical model. Neural networks are machine learning models that automatically learn to predict (a) target(s) from a set of inputs. The target typically is performance and/or power or any other metric of interest, and the inputs typically are microarchitecture parameters. Neural networks could be viewed of as a generalized non-linear regression model. Several groups have explored the idea of using neural networks to build performance models, see for example Ipek et al. (2006), Dubach, Jones, and O'Boyle (2007) and Joseph, Vaswani, and Thazhuthaveetil (2006a). Lee et al. (2007) compare spline-based

regression modeling against artificial neural networks and conclude that both approaches are equally accurate; regression modeling provides better statistical understanding while neural networks offer greater automation.

All of this prior work shares the commonality that it aims at predicting performance for a target machine for a given set of benchmarks. Their goal is to drive architecture design space exploration. We are addressing a related but very different problem: we aim at predicting performance for a target machine for a *novel* workload; this could be viewed as joint workload/architecture exploration.

6.7.2 Program similarity

Several researchers have proposed methods for quantifying program similarity. Saavedra and Smith (1996) use the squared Euclidean distance computed in a benchmark space built up using dynamic program characteristics at the Fortran programming language level such as operation mix, number of function calls, number of address computations, etc. Yi, Lilja, and Hawkins (2003) use a Plackett-Burman design for classifying benchmarks based on how the benchmarks stress the same processor components to similar degrees. Eeckhout, Vandierendonck, and De Bosschere (2003) use principal component analysis to identify similarities across programs. The input given to the principal component analysis can be microarchitecture-dependent (Eeckhout, Georges, and De Bosschere 2003), microarchitecture-independent (Phansalkar et al. 2005) or mixed of both (Eeckhout, Vandierendonck, and De Bosschere 2003). The work described in this chapter goes one step further and aims at exploiting program similarity and dissimilarity to make performance predictions for new workloads.

6.8 Conclusion

The ubiquitous problem in benchmarking is to predict the performance of an application of interest on a set of target machines the user does not have access to. This chapter presented data transposition, feasible because of the 2Way structure of the data. It builds empirical models for predicting performance across machines using a standard set of benchmarks. By building such models for a limited number of predictive machines and a (potentially) large set of target machines, data transposition allows for predicting performance of an application of interest on a set of target machines by running it on the predictive machines only. The intuition behind data transposition is that if a workload is (dis)similar to a set of benchmarks on (a) predictive machine(s), it is likely to be

proportionally (dis)similar on a target machine. An empirical model is trained to learn how workload (dis)similarity translates into performance differences on actual hardware.

Our experimental results demonstrate the method's accuracy: the ranking achieved through data transposition correlates well with the real ranking (average rank correlation coefficient of 0.93) across a set of 117 commercial machines using the SPEC CPU2006 benchmark suite. We also found that only a few predictive machines are sufficient for achieving accurate predictions. Further, the top-1 machine can be predicted with a 1.2% average prediction error (and 24.8% max error for one workload); in contrast, state-of-the-art method proposed by Hoste et al. (2006a) leads to errors above 100% for some workloads. A key benefit of data transposition is that it can better predict outlier workload performance.

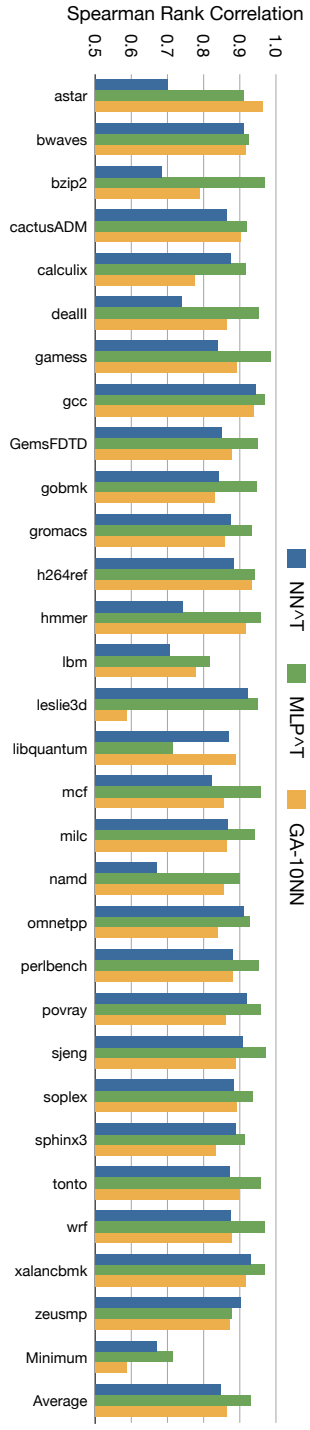


Figure 6.6: Spearman rank correlation coefficient for data transposition (NN^T and MLP^T) versus prior work ($GA\text{-}kNN$).

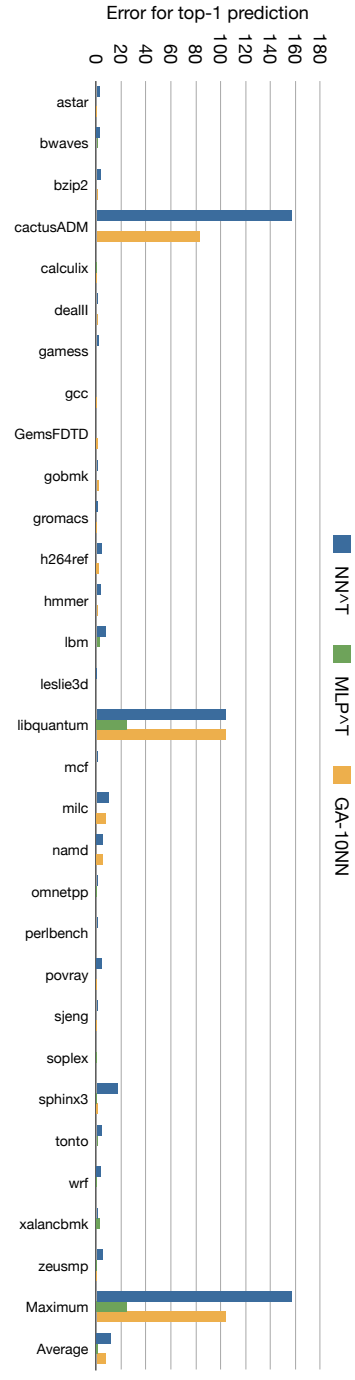


Figure 6.7: Top-1 prediction error for data transposition (NN^T and MLP^T) versus prior work (GA-kNN).

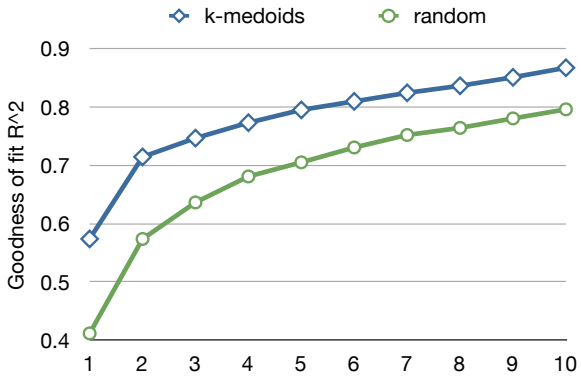


Figure 6.8: Comparing random selection versus k-medoid clustering for selecting predictive machines through MLP^T. For the random case, 50 random selections were averaged.

Chapter 7

Conclusion

During the course of this text, we explored a diverse set of multi-target problems and properties. After giving an overview of existing multi-target approaches, we study inductive transfer in multi-target algorithms and propose an algorithm to maximally exploit this inductive transfer. Next, we demonstrate how recommender systems consist, at the core, out of a multi-target problem. Some application specific problems exist in this setting to which we propose some effective solutions. Finally, we propose a new setting: 2Way learning. This setting turns out to occur often in practice and can be effectively solved using multi-target learning. The usefulness of the 2Way setting is demonstrated with an in depth study of its application to processor performance prediction. We here present an overview of the observations made in the various chapters of this text, and the conclusions arising from them.

Chapter 3 demonstrates how we can effectively use multi-target algorithms to perform transfer learning. We observe that a multi-target algorithm often outperforms a single-target algorithm for some of the targets while the single-target algorithm works better for the remaining targets. From this we conclude that the targets for which the multi-target algorithm performed the best were able to exploit the additional information present in the data for the other targets. That is, there exists a form of inductive transfer between the targets. In order to study this transfer in more depth, we construct a transfer matrix (see 3.2.1). The matrix consists of the performance values for each target when predicted together with each other possible target, i.e., it describes the inductive transfer between each pair of targets. This matrix can be highly asymmetric: it is possible that some target A has a beneficial influence on the predictive accuracy for target B when A and B are predicted simultaneously, while at the

same time, the accuracy for target B goes down. That is, the inductive transfer from A to B is positive, while the inductive transfer from B to A is negative. Hence, clustering related targets (Thrun and O'Sullivan 1996) will result in a suboptimal prediction.

The asymmetric behaviour of inductive transfer in multi-target algorithms leads us to a different approach: select one target, the main target, for which we want to maximize the predictive accuracy. The goal now becomes to select those targets, the support targets, which when predicted simultaneously with our main-target will result in the highest predictive accuracy for the main-target, i.e. those targets which as a set have the biggest positive inductive transfer towards the main-target. We propose the EAST algorithm: a heuristic search for the optimal set of support targets. Targets are added to the set of support targets one by one, each iteration selecting the target which results in the biggest increase in predictive accuracy. Experimental results demonstrate that this approach significantly outperforms both the single-target and full multi-target algorithm.

In the following chapter, we redirect our attention to Collaborative Filtering and Recommender Systems. The underlying problem in these systems consist of a multi-target problem: predict a score for each item (target) for a specific user. They are, however, rarely discussed from a multi-target point of view. Direct application of existing multi-target algorithms did not lead to improved accuracies in comparison to existing state-of-the-art collaborative filtering algorithms. This is the result of the inability of the multi-target algorithms to handle two problems which are specific to the collaborative filtering setting: the sparsity problem and the cold start problem. The former is a result of users rating only a small portion of all items. Hence, the odds of two users rating a large number of the same items are small. Most algorithms use a distance measure based on these co-rated items. Consequently, the distance measure's usefulness, and subsequently the algorithm's performance deteriorate fast due to the large data sparsity. The related cold start problem exists for new users. Those users have rated only a small number of items, making it difficult to make accurate predictions for them.

We propose a graph based solution to alleviate these shortcomings. A user-graph is created with each node representing a user and weighted edges between users are added based on how similarly they co-rate certain items. Suppose user A and B rate some items in a very similar way, user B and C rate some different set of items in a very similar way but A and C did not co-rate any item. The constructed graph would have highly weighted edges between A and B and between B and C but no edge between A and C. We can now use this graph to make more informed recommendations. Suppose we want to predict how well user A likes item X, user B did not rate the item either but C did rate

the item. Most recommender systems would be unable to make a prediction for user A's rating for X because A and C did not co-rate any items, due to the sparsity of the data. Our solution consists of propagating values through the graph. For this example, we would first propagate the known rating from C to B and finally from B to A. The algorithm, described in Section 4.5, uses a probabilistic representation which takes the weights of the edges and the possibly conflicting values of neighboring nodes into account. We ran a number of experiments which demonstrated the effectiveness of our method. As expected, our approach out-performed the other systems when the data set is extremely sparse. As the data gets less sparse, the advantage of our method disappears and the propagated values might even be detrimental to the accuracy of the system.

In some cases, the user-graph is given. For example, the social network of a user might be known by the recommender system. This simplifies making predictions since we do not have to create the user-graph. The rationale for making predictions changes to "friends like the same items", compared to the previous rationale "users who liked the same items will like the same items in the future". We propose a method which recommends items to a user which are liked by other users in the user's so called social-neighborhood. This neigborhood includes immediate friends and friends-of-friends, up to a certain degree of indirect friendship. Experimental evaluation demonstrated that this relatively simple method achieves state-of-the-art performance.

Chapter 5 introduces a new learning setting: 2Way learning. In this setting, we want to learn the attribute of a relation between two types of objects $\mathcal{A}$ and $\mathcal{B}$. This structure is present in many applications, among which: ecological applications, the previously discussed collaborative filtering and processor performance prediction. The training data associated with a 2Way setting can be placed in a matrix format with each row representing an $\mathcal{A}$ object and each column representing a $\mathcal{B}$object. This matrix can be interpreted in a number of ways. The first interpretation consists of considering each row a training example. This results in a setting which adheres to a normal multi-target problem setting. Alternatively, we can consider each column a training example. This is possible due to the interchangeability of the type $\mathcal{A}$ and type $\mathcal{B}$ objects in the relation. Finally, a hybrid interpretation can be made.

The alternative, column-based approach can be transformed to a normal row-based setting by a simple transposition of the data matrix. As a result, we can apply any multi-target algorithm to solve for the column-based interpretation. This data transposition has a number of peculiar effects. First, examples and attributes are switched. Depending on the direction in which we use the data, either the rows represent training examples and the columns represent attributes; or vice versa. Hence, their exists no clear distinction between examples and

attributes in a 2Way learning setting. Second, as described in Section 5.2, transposing the data will use an inductive learner in a transductive way. That is, a model is build when the prediction is requested, when the input data is known. Third, row-based learning achieves generalization over $\mathcal{A}$ and inductive transfer over $\mathcal{B}$, whereas column-based learning achieves generalization over $\mathcal{B}$ and inductive transfer over $\mathcal{A}$. Finally, multi-target and single-target prediction can be switched by transposing the data. A single-target problem for which we need to predict one value for n test-instances becomes a multi-target problem with n targets which we need to complete for only one test-instance; and vice versa.

We ran experiments with both approaches in two domains: one ecological problem, the other concerning processor performance prediction. From these experiments we conclude that the optimal direction to use the data is dependent on the multi-target learning algorithm. Relatively complex learners, such as a neural network and a support vector machine, achieve better accuracy with transposed approach. For the ecological dataset, the linear regression works better in the normal row-based approach but is still less accurate than the other algorithms using a column-based approach.

Chapter 6, discusses the aforementioned processor performance prediction problem in-depth. Previous work in this area used micro-architecture independent characteristics to describe the behaviour of a certain application. The training dataset consisted of a number of benchmark applications for which the characteristics were measured and of the performance values for these applications on a large number of CPU architectures. Given a new application of interest, the approach consisted of finding the benchmark application which resembles that application the most. The performance values of the most similar application would subsequently be used as predictions for the application of interest.

We propose a different method which improves on prior work in a number of ways. We run the application of interest on a small number of (predictive) machines to determine the performance values of our application on those machines. This results in a 2Way setting: one type of object being the machines, the other type being the applications, and the performance values the attribute of the relation between those two types of objects. Hence, we can apply data transposition. Experimental evaluation showed that for this problem, the column-based approach consistently results in more accurate predictions. This is consistent with the following intuition: if an application is (dis)similar to a set of benchmarks applications on (a) predictive machine(s), it is likely to be proportionally (dis)similar on a target machine. Prior work was based on program similarity, whereas our approach is based on machine similarity.

Our experimental results lead us to the following conclusions: the data transposition approach is able to make predictions for new architectures for which no examples occur in the training data; the method is able to predict the performance of future machines; works accurate with a small number of predictive machines; and performs well at predicting outlier performance.

From these observations and applications concerning multi-target algorithms, it should be clear that they can provide an excellent solution to many real world problems. Although we covered many multi-target related topics and provided effective solutions to a number of problems, there is always room for improvement. Future work includes improving our support set search algorithm (EAST); efficient implementation of our graph based collaborative filter; hybrid row-based, column-based approaches and so on. We hope to address these, and more, questions in future work.

We briefly summarize the main contributions made in this work.

- we have shown that inductive transfer is asymmetric, and that acknowledging this fact can lead to better multi-target or multi-task learners
- we have shown that multi-target learners can be used for building recommender systems
- we have proposed a novel solution for the cold-start and sparsity problems in recommender systems
- we have proposed a new viewpoint on multi-target learning, which we call two-way learning
- we have proposed a novel method for microprocessor performance prediction that is more accurate than the state of the art

List of publications

- Piccart, B.,H. Blockeel, A. Georges and L. Eeckhout (2012). "Predictive learning in two-way datasets" In: Latest advances in inductive logic programming *21st International Conference on Inductive Logic Programming*
- Piccart, B., A. Georges, H. Blockeel and L. Eeckhout (2011). "Ranking commercial machines through data transposition" In: *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)* pp. 3-14
- Piccart, B., H. Rahmani, D. Fierens and H. Blockeel (2010). "Three complementary approaches to context aware movie recommendation" In: *Proceedings of the Workshop on Context-Aware Movie Recommendation* pp. 57-60
- Piccart, B., J. Struyf, H. Blockeel (2010). "Alleviating the sparsity problem in collaborative filtering by using an adapted distance and a graph-based method" In: *Proceedings of the Tenth SIAM International Conference on Data Mining* pp. 189-199
- Piccart, B., J. Struyf, H. Blockeel (2008). "Empirical asymmetric selective transfer in multi-objective decision trees" *International conference on Discovery Science* In: Lecture Notes in Computer Science vol. 5255 pp. 64-75
- Piccart, B., J. Struyf, H. Blockeel (2008). "Selective inductive transfer" In: *Proceedings of Benelearn'08: The Annual Belgian-Dutch Machine Learning Conference* pp. 63-64

Bibliography

Adomavicius, G. and A. Tuzhilin (2005a). "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions". In: *IEEE Transactions on Knowledge and Data Engineering* 17.6, pp. 734–749.

Adomavicius, G. and A. Tuzhilin (2005b). "Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions". In: *IEEE Transactions on Knowledge and Data Engineering* 17.6, pp. 734–749. ISSN: 1041-4347. DOI: 10.1109/TKDE.2005.99. URL: http://dx.doi.org/10.1109/TKDE.2005.99.

Aho, T., B. Zenko, and S. Dzeroski (2009). "Rule Ensembles for Multi-target Regression". In: *ICDM 2009, The Ninth IEEE International Conference on Data Mining, Miami, Florida, USA, 6-9 December 2009*. Ed. by W. Wang, H. Kargupta, S. Ranka, P. S. Yu, and X. Wu. IEEE Computer Society, pp. 21–30. ISBN: 978-0-7695-3895-2.

Blockeel, H., L. De Raedt, and J. Ramon (1998). "Top-down Induction of Clustering Trees". In: *15th Int'l Conf. on Machine Learning*, pp. 55–63.

Blockeel, H., L. Schietgat, J. Struyf, S. Džeroski, and A. Clare (2006). "Decision Trees for Hierarchical Multilabel Classification: A Case Study in Functional Genomics". In: *10th European Conf. on Principles and Practice of Knowledge Discovery in Databases*, pp. 18–29.

Blockeel, H., S. Džeroski, and J. Grbović (1999). "Simultaneous prediction of multiple chemical parameters of river water quality with TILDE". In: *3rd European Conf. on Principles of Data Mining and Knowledge Discovery*, pp. 32–40.

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

Brown, G., J. L. Wyatt, R. Harris, and X. Yao (2005). "Diversity creation methods: a survey and categorisation". In: *Information Fusion* 6.1, pp. 5–20.

Candillier, L., F. Meyer, and F. Fessant (2008). "Designing Specific Weighted Similarity Measures to Improve Collaborative Filtering Systems". In: *ICDM '08: Proceedings of the 8th industrial conference on Advances in Data Mining*. Leipzig, Germany: Springer-Verlag, pp. 242–255. ISBN: 978-3-540-70717-2. DOI: http://dx.doi.org/10.1007/978-3-540-70720-2_19.

Caruana, R. (1997a). "Multitask Learning". In: *Mach. Learn.* 28, pp. 41–75.

— (1997b). "Multitask Learning". In: *Machine Learning* 28 (1), pp. 41–75. ISSN: 0885-6125. DOI: 10.1023/A:1007379606734. URL: http://portal.acm.org/citation.cfm?id=262868.262872.

Caruana, R. (1993). "Multitask Learning: A Knowledge-Based Source of Inductive Bias". In: *Int'l Conf. on Machine Learning*, pp. 41–48.

Clare, A. and R. King (2001). "Knowledge discovery in multi-label phenotype data". In: *5th European Conf. on Principles of Data Mining and Knowledge Discovery*, pp. 42–53.

De Raedt, L. (2008). *Logical and Relational Learning*. Springer.

Demšar, D., M. Debeljak, C. Lavigne, and S. Džeroski (2005). *Modelling pollen dispersal of genetically modified oilseed rape within the field*. Abstract presented at The Annual Meeting of the Ecological Society of America.

Demšar, D., S. Džeroski, T. Larsen, J. Struyf, J. Axelsen, M. Bruus Pedersen, and P. Henning Krogh (2006). "Using multi-objective classification to model communities of soil microarthropods". In: *Ecol. Model.* 191.1, pp. 131–143.

Dimou, A., G. Tsoumakas, V. Mezaris, I. Kompatsiaris, and I. Vlahavas (2009). "An Empirical Study of Multi-label Learning Methods for Video Annotation". In: *Proceedings of the 2009 Seventh International Workshop on Content-Based Multimedia Indexing*. CBMI '09. Washington, DC, USA: IEEE Computer Society, pp. 19–24. ISBN: 978-0-7695-3662-0. DOI: 10.1109/CBMI.2009.37. URL: http://dx.doi.org/10.1109/CBMI.2009.37.

Dubach, C., T. M. Jones, and M. F. P. O'Boyle (Dec. 2007). "Microarchitecture Design Space Exploration Using An Architecture-Centric Approach". In: *Proceedings of the IEEE/ACM Annual International Symposium on Microarchitecture (MICRO)*, pp. 262–271.

Džeroski, S., S. Muggleton, and S. Russell (1992). "PAC-learnability of determinate logic programs". In: *Proceedings of the 5th ACM Workshop on Computational Learning Theory*. ACM Press, pp. 128–135.

Eeckhout, L., A. Georges, and K. De Bosschere (Oct. 2003). "How Java Programs Interact with Virtual Machines at the Microarchitectural Level". In: *Proceedings of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Languages, Applications and Systems (OOPSLA)*, pp. 169–186.

Eeckhout, L., H. Vandierendonck, and K. De Bosschere (Feb. 2003). "Quantifying the Impact of Input Data Sets on Program Behavior and its Applications". In: *Journal of Instruction-Level Parallelism* 5. http://www.jilp.org/vol5.

Elisseeff, A. and J. Weston (2005). "A Kernel Method for Multi-Labelled Classification". In: *Annual ACM Conference on Research and Development in Information Retrieval*, pp. 274–281. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.18.2423.

Evgeniou, T. and M. Pontil (2004). "Regularized multi–task learning". In: *Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 109–117.

Fürnkranz, J., E. Hüllermeier, E. Loza Mencía, and K. Brinker (Nov. 2008). "Multilabel classification via calibrated label ranking". In: *Mach. Learn.* 73.2, pp. 133–153. ISSN: 0885-6125. DOI: 10.1007/s10994-008-5064-8. URL: http://dx.doi.org/10.1007/s10994-008-5064-8.

Getoor, L., N. Friedman, D. Koller, and A. Pfeffer (2001). "Learning Probabilistic Relational Models". In: *Relational Data Mining*. Springer, pp. 307–334.

Ghosn, J. and Y. Bengio (2003). "Bias learning, knowledge sharing". In: *Neural Networks, IEEE Transactions on* 14.4, pp. 748–765.

Godbole, S. and S. Sarawagi (2004). "Discriminative Methods for Multi-labeled Classification". In: *PAKDD*, pp. 22–30.

Hall, M., E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten (2009). "The WEKA Data Mining Software: An Update". In: *SIGKDD Explorations* 11.1.

Hofmann, T. (2004). "Latent semantic models for collaborative filtering". In: *ACM Transactions on Information Systems* 22.1, pp. 89–115.

Hoste, K., A. Phansalkar, L. Eeckhout, A. Georges, L. K. John, and K. De Bosschere (Sept. 2006a). "Performance Prediction based on Inherent Program Similarity". In: *Proceedings of the 2006 International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pp. 114–122.

Hoste, K., A. Phansalkar, L. Eeckhout, A. Georges, L. K. John, and K. De Bosschere (2006b). "Performance prediction based on inherent program similarity". In: *Proceedings of the 15th international conference on Parallel architectures and compilation techniques*. PACT '06. New York, NY, USA: ACM, pp. 114–122. ISBN: 1-59593-264-X. DOI: http://doi.acm.org/10.1145/1152154.1152174. URL: http://doi.acm.org/10.1145/1152154.1152174.

Huang, Z., H. Chen, and D. Zeng (2004). "Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering". In: *ACM Trans. Inf. Syst.* 22.1, pp. 116–142. ISSN: 1046-8188. DOI: 10.1145/963770.963775. URL: http://dx.doi.org/10.1145/963770.963775.

Huang, Z., D. Zeng, and H. Chen (2007). "A Comparison of Collaborative-Filtering Recommendation Algorithms for E-commerce". In: *Intelligent Systems, IEEE* 22.5, pp. 68–78. DOI: 10.1109/MIS.2007.4338497. URL: http://dx.doi.org/10.1109/MIS.2007.4338497.

Huang, Z., W. Chung, T.-H. Ong, and H. Chen (2002). "A graph-based recommender system for digital library". In: *JCDL '02: Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries*. Portland, Oregon, USA: ACM, pp. 65–73. ISBN: 1-58113-513-0. DOI: http://doi.acm.org/10.1145/544220.544231.

Ipek, E., S. A. McKee, B. R. de Supinski, M. Schulz, and R. Caruana (Oct. 2006). "Efficiently Exploring Architectural Design Spaces via Predictive Modeling". In: *Proceedings of the Twelfth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 195–206.

Jensen, D., J. Neville, and B. Gallagher (2004). "Why collective inference improves relational classification". In: *10th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, pp. 593–598.

Jensen, D. and J. Neville (2002). "Linkage and Autocorrelation Cause Feature Selection Bias in Relational Learning". In: *ICML*. Ed. by C. Sammut and A. G. Hoffmann. Morgan Kaufmann, pp. 259–266. ISBN: 1-55860-873-7.

Joseph, P. J., K. Vaswani, and M. J. Thazhuthaveetil (Dec. 2006a). "A Predictive Performance Model for Superscalar Processors". In: *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 161–170.

— (Feb. 2006b). "Construction and Use of Linear Regression Models for Processor Performance Analysis". In: *Proceedings of the 12th International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 99–108.

Kampichler, C., S. Džeroski, and R. Wieland (2000). "The application of machine learning techniques to the analysis of soil ecological data bases: Relationships between habitat features and Collembola community characteristics". In: *Soil. Biol. Biochem.* 32, pp. 197–209.

Kumar, R., K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen (Dec. 2003). "Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction". In: *Proceedings of the ACM/IEEE Annual International Symposium on Microarchitecture (MICRO)*, pp. 81–92.

Lavrač, N. and S. Džeroski (1994). *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, New York.

Lee, B. and D. Brooks (Oct. 2006). "Accurate and Efficient Regression Modeling for Microarchitectural Performance and Power Prediction". In:

Proceedings of the Twelfth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pp. 185–194.

Lee, B., D. Brooks, B. R. de Supinski, M. Schulz, K. Singh, and S. A. McKee (Mar. 2007). "Methods of Inference and Learning for Performance Modeling of Parallel Applications". In: *Proceedings of the 12th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP)*, pp. 249–258.

Lee, D. D. and H. S. Seung (2000). "Algorithms for Non-negative Matrix Factorization". In: *NIPS*, pp. 556–562.

Miyahara, K. and M. J. Pazzani (2000). "Collaborative Filtering with the Simple Bayesian Classifier". In: *In: Proceedings of the 6th Pacific Rim International Conference on Artificial Intelligence*, pp. 679–689.

Nanopoulos, A. (2007). "Collaborative Filtering Based on Transitive Correlations Between Items". In: *Advances in Information Retrieval*.

Papagelis, M., D. Plexousakis, and T. Kutsuras (2005). "Alleviating the sparsity problem of collaborative filtering using trust inferences". In: *iTrust*. Ed. by P. Herrmann. Springer-Verlag Berlin Heidelberg, pp. 224–239.

Phansalkar, A., A. Joshi, L. Eeckhout, and L. K. John (Mar. 2005). "Measuring Program Similarity: Experiments with SPEC CPU Benchmark Suites". In: *Proceedings of the 2005 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 10–20.

Piccart, B., J. Struyf, and H. Blockeel (2008a). "Empirical Asymmetric Selective Transfer in Multi-objective Decision Trees". In: *Discovery Science*. Ed. by J.-F. Boulicaut, M. R. Berthold, and T. Horváth. Vol. 5255. Lecture Notes in Computer Science. Springer, pp. 64–75. ISBN: 978-3-540-88410-1.

Piccart, B., H. Blockeel, and J. Struyf (Apr. 2010). "Alleviating the sparsity problem in collaborative filtering by using an adapted distance and a graph-based method". In: *Proceedings of the Tenth SIAM International Conference on Data Mining,* SIAM, pp. 189–199. URL: `https://lirias.kuleuven.be/handle/123456789/297418`.

Piccart, B., J. Struyf, and H. Blockeel (May 2008b). "Selective inductive transfer". Benelearn'08: : The Annual Belgian-Dutch Machine Learning Conference, Spa, Belgium, 19-20 May 2008. URL: `https://lirias.kuleuven.be/handle/123456789/202004`.

Piccart, B., H. Blockeel, H. Rhamani, and F. Daan (2010). "Three complementary approaches to context aware movie recommendation". In: *Proceedings of the Workshop on Context-Aware Movie Recommendation*. Barcelona, Spain, pp. 57–60.

Piccart, B., A. Georges, H. Blockeel, and L. Eeckhout (2011). "Ranking commercial machines through data transposition". In: *Proceedings*

of the IEEE International Symposium on Workload Characterization (IISWC), pp. 3–14. DOI: 10.1109/IISWC.2011.6114192. URL: https://lirias.kuleuven.be/handle/123456789/332638.

Piccart, B., H. Blockeel, A. Georges, and L. Eeckhout (2012). "Predictive learning in two-way datasets". In: *Latest Advances in Inductive Logic Programming,* Accepted. Imperial College Press. URL: https://lirias.kuleuven.be/handle/123456789/332636.

Radcliffe, N. and P. D. Surry (1995). "Fundamental Limitations on Search Algorithms: Evolutionary Computing in Perspective". In: *LECTURE NOTES IN COMPUTER SCIENCE 1000*. Springer-Verlag, pp. 275–291.

Read, J., B. Pfahringer, G. Holmes, and E. Frank (2009). "Classifier Chains for Multi-label Classification". In: *ECML/PKDD (2)*, pp. 254–269.

Resnick, P., N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl (1994). "GroupLens: An Open Architecture for Collaborative Filtering of Netnews". In: ACM Press, pp. 175–186.

Saavedra, R. H. and A. J. Smith (Nov. 1996). "Analysis of Benchmark Characteristics and Benchmark Performance Prediction". In: *ACM Transactions on Computer Systems* 14.4, pp. 344–384.

Sain, S. and P. Carmack (2001). "A MIXTURE APPROACH FOR MULTIVARIATE REGRESSION TREES". In: *Proceedings of the Annual Meeting of the American Statistical Association*.

Sarwar, B. M., G. Karypis, J. A. Konstan, and J. T. Riedl (2000). "Application of Dimensionality Reduction in Recommender System - A Case Study". In: *In ACM WebKDD Workshop*.

Schein, A. I., A. Popescul, L. H., R. Popescul, L. H. Ungar, and D. M. Pennock (2002a). "Methods and Metrics for Cold-Start Recommendations". In: *In Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM Press, pp. 253–260.

Schein, A. I., A. Popescul, L. H. Ungar, and D. M. Pennock (2002b). "Methods and metrics for cold-start recommendations". In: *SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*. Tampere, Finland: ACM, pp. 253–260. ISBN: 1-58113-561-0. DOI: http://doi.acm.org/10.1145/564376.564421.

Silver, D. and R. Mercer (1996). "The parallel transfer of task knowledge using dynamic learning rates based on a measure of relatedness". In: *Connect. Sci.* 8.2, pp. 277–294.

Silver, D. L. and R. E. Mercer (2001). "Selective Functional Transfer: Inductive Bias from Related Tasks". In: *IASTED Int'l Conf. on Artificial Intelligence and Soft Computing*, pp. 182–189.

Srebro, N., J. D. M. Rennie, and T. S. Jaakkola (2005). "Maximum-margin matrix factorization". In: *Advances in Neural Information Processing Systems 17*, pp. 1329–1336. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.59.118.

Struyf, J. and S. Džeroski (2006). "Constraint based induction of multi-objective regression trees". In: *Knowledge Discovery in Inductive Databases, 4th Int'l Workshop, Revised, Selected and Invited Papers*, pp. 222–233.

Su, X. and T. M. Khoshgoftaar (2006). "Collaborative Filtering for Multi-class Data Using Belief Nets Algorithms". In: *ICTAI '06: Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence*. Washington, DC, USA: IEEE Computer Society, pp. 497–504. ISBN: 0-7695-2728-0. DOI: http://dx.doi.org/10.1109/ICTAI.2006.41.

Suzuki, E., M. Gotoh, and Y. Choki (2001). "Bloomy Decision Tree for Multi-objective Classification". In: *Principles of Data Mining and Knowledge Discovery*. Ed. by L. De Raedt and A. Siebes. Vol. 2168. Lecture Notes in Computer Science. 10.1007/3-540-44794-6$_3$6. Springer Berlin / Heidelberg, pp. 436–447. ISBN: 978-3-540-42534-2. URL: http://dx.doi.org/10.1007/3-540-44794-6_36.

Teodorescu, R. and J. Torrellas (June 2008). "Variation-Aware Application Scheduling and Power Management for Chip Multiprocessors". In: *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pp. 363–374.

Thrun, S. and J. O'Sullivan (1996). "Discovering structure in multiple learning tasks: The TC algorithm". In: *13th Int'l Conf. on Machine Learning*, pp. 489–497.

Torgo, L. (1998). "Error estimators for pruning regression trees". In: *10th European Conf. on Machine Learning*, pp. 125–130.

Trafalis, T. B. and O. Oladunni (2005). "Pairwise multi-classification support vector machines: quadratic programming (QP-PAMSVM) formulations". In: *Proceedings of the 6th WSEAS international conference on Neural networks*. World Scientific, Engineering Academy, and Society (WSEAS), pp. 205–210. ISBN: 960-8457-24-6.

Tsochantaridis, I., T. Joachims, T. Hofmann, and Y. Altun (2005). "Large Margin Methods for Structured and Interdependent Output Variables". In: *Journal of Machine Learning Research* 6, pp. 1453–1484.

Tsoumakas, G., I. Katakis, and I. P. Vlahavas (2010). "Mining Multi-label Data". In: *Data Mining and Knowledge Discovery Handbook*. Ed. by O. Maimon and L. Rokach. Springer, pp. 667–685. ISBN: 978-0-387-09822-7.

Vens, C., J. Struyf, L. Schietgat, S. Džeroski, and H. Blockeel (Nov. 2008). "Decision trees for hierarchical multi-label classification". In: *Mach. Learn.*

73.2, pp. 185–214. ISSN: 0885-6125. DOI: 10.1007/s10994-008-5077-3. URL: http://dx.doi.org/10.1007/s10994-008-5077-3.

Wicker, J., B. Pfahringer, and S. Kramer (2012). "Multi-label classification using boolean matrix decomposition". In: *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. SAC '12. Trento, Italy: ACM, pp. 179–186. ISBN: 978-1-4503-0857-1. DOI: 10.1145/2245276.2245311. URL: http://doi.acm.org/10.1145/2245276.2245311.

Yi, J. J., D. J. Lilja, and D. M. Hawkins (Feb. 2003). "A Statistically Rigorous Approach for Improving Simulation Methodology". In: *Proceedings of the Ninth International Symposium on High Performance Computer Architecture (HPCA)*, pp. 281–291.

Zhang, S., W. Wang, J. Ford, and F. Makedon (2006). "Learning from Incomplete Ratings Using Non-negative Matrix Factorization". In: *SDM*.

Biography

Beau Piccart was born in Hasselt, Belgium, in 1983. As a little child already, he showed great interest in technology. Clocks, radios and power plugs alike, needed unscrewing to reveal their mystery, causing occasional distress to his caretakers. However, this at first slightly destructive interest in exploring the mechanics of machinery would later on prove to be particularly useful. Around the age of twelve, a new 'toy' became available: the personal computer. Soon enough, Beau was building his own PCs, he learnt the ways of the internet and evolved into an adequate web-developer. At the age of just fifteen, he received his first assignment as a professional web-developer. Aged seventeen, he was invited to develop the website of local television station TV Limburg. He continued working as a web-developer throughout high school. As a logical next step, he decided to study Computer Science at the renowned University of Leuven. He wrote his Master thesis in computer graphics on an interactive BRDF-modeller and graduated cum laude. Beau then ambitioned a Ph.D. in Engineering in Computer Science. This dissertation, on algorithms for Multi-Task Learning, is presented in partial fulfilment of the requirements to obtain that Ph.D. To date, Beau has published and presented at numerous conferences, including: the International Conference on Inductive Logic Programming, the IEEE International Symposium on Workload Characterization, the SIAM International Conference on Data Mining and the International conference on Discovery Science.

Arenberg Doctoral School of Science, Engineering & Technology
Faculty of Engineering
Department of Computer Science
Declarative Languages and Artificial Intelligence
Celestijnenlaan 200A box 2402
B-3001 Heverlee