TECHNISCHE
UNIVERSITÄT
DARMSTADT

Machine Teaching

A Machine Learning Approach to Technology Enhanced Learning

Dissertation

Zur Erlangung des akademischen Grades
eines Doktor-Ingenieur (Dr.-Ing.)

Eingereicht von
Diplom Wirtschaftsinformatiker Markus Weimer
geb. in Hadamar

Angenommen vom Fachbereich Informatik
der Technischen Universität Darmstadt

Gutachter:	Prof. Dr. Max Mühlhäuser (TU Darmstadt)
	Prof. Dr. Alexander J. Smola Australian National University, Canberra, Australien Yahoo! Research, Santa Clara, CA, USA
	Prof. Dr. Petra Gehring (TU Darmstadt)
Tag der Einreichung:	14.07.2009
Tag der Disputation:	24.09.2009

Darmstädter Dissertationen, D17 Erschienen in 2010 in Darmstadt.

Abstract

Many applications of Technology Enhanced Learning are based on strong assumptions: Knowledge needs to be standardized, structured and most of all externalized into learning material that preferably is annotated with meta-data for efficient re-use. A vast body of valuable knowledge does not meet these assumptions, including *informal knowledge* such as experience and intuition that is key to many complex activities.

We notice that knowledge, even if not standardized, structured and externalized, can still be *observed* through its application. We refer to this observable knowledge as PRACTICED KNOWLEDGE. We propose a novel approach to Technology Enhanced Learning named MACHINE TEACHING to convey this knowledge: Machine Learning techniques are used to extract machine models of Practiced Knowledge from observational data. These models are then applied in the learner's context for his support.

We identify two important subclasses of machine teaching, General and Detailed Feedback Machine Teaching. GENERAL FEEDBACK MACHINE TEACHING aims to provide the learner with a "grade-like" numerical rating of his work. This is a direct application of supervised machine learning approaches. DETAILED FEEDBACK MACHINE TEACHING aims to provide the learner with in-depth support with respect to his activities. An analysis showed that a large subclass of Detailed Feedback Machine Teaching applications can be addressed through adapted recommender systems technology.

The ability of the underlying machine learning techniques to capture structure and patterns in the observational data is crucial to the overall applicability of Machine Teaching. Therefore, we study the feasibility of Machine Teaching from a machine learning perspective.

Following this goal, we evaluate the General Feedback Machine Teaching approach using state-of-the-art machine learning techniques: The exemplary Machine Teaching system is sought to provide the learner with quality estimations of his writing as judged by an online community. The results obtained in this evaluation are supportive of the applicability of Machine Teaching to this domain.

To facilitate Detailed Feedback Machine Teaching, we present a novel matrix factorization model and algorithm. In addition to addressing the needs of Machine Teaching, it is also a contribution to the recommender systems field as it facilitates ranking estimation. An Evaluation in a Detailed Feedback Machine Teaching scenario for software engineers supports the feasibility of Machine Teaching in that domain.

We therefore conclude that machine learning models capable of capturing important aspects of practiced knowledge can be found in both, General and Detailed Feedback Machine Teaching. Machine Teaching does not assume the knowledge to be externalized, but to be *observable* and therefore adds another body of knowledge to Technology Enhanced Learning not amenable to traditional Technology Enhanced Learning approaches.

Zusammenfassung

Viele erfolgreiche E-Learning Systeme basieren auf strengen Annahmen: Das zu vermittelnde Wissen muss strukturiert und standardisiert in Lerninhalte externalisiert sein. Diese wiederum sollten mit Metadaten angereichert sein, um ihre Wiederverwendung zu ermöglichen. Diese strikten Anforderungen verhindern es, für viele Aktivitäten entscheidendes informelles Wissen, also unter anderem Erfahrung und Intuition, zu vermitteln.

Wissen, auch wenn es weder standardisiert, strukturiert noch externalisiert wurde, manifestiert sich in Aktivitäten seiner Träger. Wir nennen dieses beobachtbare Wissen PRAKTIZIERTES WISSEN. In dieser Dissertation wird MACHINE TEACHING eingeführt, ein neuer Ansatz zum E-Learning, der diese Tatsache wie folgt ausnutzt: Aus Beobachtungsdaten werden mit Methoden des maschinellen Lernens Modelle extrahiert, die dann im Kontext des Lerners zu seiner Unterstützung eingesetzt werden.

Innerhalb dieses Ansatzes werden zwei wichtige Teilaufgaben eines Machine Teaching Systems identifiziert: Generelles und Detailliertes Feedback. Ziel von Machine Teaching für Generelles Feedback ist es, die Arbeit des Lerners zu bewerten, etwa durch „Zensuren". Dies kann durch aktuelle Verfahren des überwachten maschinellen Lernen geleistet werden. Machine Teaching für Detailliertes Feedback soll den Lerner hingegen mit feingranularen Hinweisen zu seiner Arbeit unterstützen. Wir zeigen, dass ein großer Anteil dieser Aufgabe mittels angepasster Recommender Systems Technologie bearbeitet werden kann.

Die Nützlichkeit zukünftiger Machine Teaching Systeme wird vor allem davon abhängen, wie gut es mittels maschineller Lernverfahren möglich ist, das Praktizierte Wissen in Form von Mustern und Strukturen aus den Beobachtungsdaten zu extrahieren. Folglich wird in dieser Dissertation untersucht, ob und in wie weit dies möglich ist.

Die erste Evaluation hierzu erfolgt am Beispiel eines Machine Teaching Systems für generelles Feedback. Es wird ein System evaluiert, das Texte automatisch bewertet. Dies geschieht auf Basis vergangener Bewertungen anderer Texte durch eine Internet Community. Aus der Leistung des Systems bei dieser Aufgabe folgt, dass Machine Teaching für generelles Feedback hier erfolgreich eingesetzt werden kann.

Um Machine Teaching für detailliertes Feedback zu ermöglichen, stellen wir ein neues Modell für Recommender Systeme vor. Dieses Modell stellt eine Erweiterung des Matrixfaktorisierungsansatzes dar. Neben seiner Ausrichtung auf Machine Teaching ist der Algorithmus der erste, der Reihenfolgevorhersagen für Recommender Systeme ermöglicht. Wir evaluieren ihn in einem Machine Teaching Ansatz für detailliertes Feedback im Bereich Softwareentwicklung. Basierend auf einer Quelltextdatenbank soll dieser auf fehlende Aufrufe hinweisen. Auch hier lassen die empirischen Ergebnisse schliessen, dass Machine Teaching in dieser Domäne anwendbar ist.

Machine Teaching stellt also eine machbare Erweiterung des E-Learning dar, dessen Einsatzbreite mit dem Fortschritt des maschinellen Lernens wächst. Es erweitert bisherige Ansätze um beobachtbares und damit eben auch informelles Wissen, das bisher im E-Learning nur schwer vermittelbar ist.

Acknowledgements

This work would not have been possible without the support and encouragement of my advisors, colleagues and fellow PhD students. I am grateful to all those who contributed to the outcome of this thesis.

First and foremost, I would like to thank my advisors Max Mühlhäuser (TU Darmstadt) and Alex Smola (Australian National University, Yahoo! Research). They supported me with sharing their respective expertise in Technology Enhanced Learning and Machine Learning, but even more so with excellent advice on research in general.

My work in the past three years has been defined by fruitful and intense collaborations: The most influential has been with Alexandros Karatzoglou who I worked with as if he shared an office with me, despite the fact that he held several different positions at TU Vienna (Austria), CIRO (Sydney, Australia) and INRIA (Rouen, France) during our collaboration. I am also thankful for the collaboration with and advice from Marcel Bruch(TU Darmstadt) and Quoc Viet Le (Stanford University), Choon-Hui Theo (Australian National University) and my colleagues at NICTA (Canberra, Australia) and Yahoo! Research (Santa Clara, California) during my respective visits.

I am grateful for having been a member of the interdisciplinary postgraduate school "eLearning" at TU Darmstadt, the context in which my work was funded. I thank all advisors and all Ph.D. students of the postgraduate school for providing an inspiring atmosphere and hosting cross-disciplinary discussions. In particular, Andreas Kaminski and Petra Gehring have been of tremendous help in sharpening many of the arguments in the present thesis by asking the right, tough questions before and in the review process.

Many thanks are due to all present and former members of the Telecooperation group at TU Darmstadt. They provided a very friendly place to work and supported this thesis in many respects. I am also grateful for the support by the Frankfurt Center for Scientific Computing in running many of the experiments reported in this thesis.

Last but not least, I am very grateful to my family which always supported me.

Contents

List of Algorithms

List of Figures

1 Introduction

Contents

1.1 Motivation: The need for Machine Teaching

Today, Technology Enhanced Learning is applied in many instances. Most commonly known to students are Learning Management Systems (LMS). A learning management system is used to support the processes in a formal learning setting such as within universities by providing means for content distribution, communication and collaboration. Well known examples include the open source packages Moodle [Com09a] and Sakai [Com09b] as well as commercial products, e.g. the Blackboard system [Bla09] used by universities worldwide.

Tools like Power Trainer [QABC09] and Lecturnity [AG09] support the teachers in creating the content to be made available through the learning management systems. There are even mature content standards such as the Sharable Content Object Reference Model (SCORM) [Lea09] and the ones set by the IMS Global Learning Consortium[1]. These standards facilitate the use of sophisticated meta-data schemes that describe the content to support the mix-and-match of content from different authors.

Adaptive hypermedia approaches such as AHA! [DBSS06] build upon these systems and standards to allow the teacher to build adaptive content by expressing rules like "To understand concept X, the student shall know concept A, B and C". These rules are then used to support the student's navigation in the content.

Despite these successful applications of technology enhanced learning, there is a growing interest in what is called "eLearning 2.0" to overcome the inherent limits of the dominant approaches to technology enhanced learning, including those mentioned so far. This thesis contributes to this movement by introducing a machine learning based approach to technology enhanced learning.

Before presenting a critique of the traditional technology enhanced learning approaches, we introduce the following notation:

Notation 1 (Learner)**.** *We use the term "learner" throughout this thesis deliberately instead of "student", as the proposed approach is primarily aimed at informal learning and not limited to formal learning scenarios like those found in university courses.*

Assumptions of Traditional Technology Enhanced Learning

The approach presented in this thesis departs from traditional technology enhanced learning by overcoming certain limiting assumptions regarding the knowledge and the learners which shall be argued below:

Assumptions with respect to Knowledge

We argue that knowledge in traditional technology enhanced learning approaches is assumed to be standardized, externalized and structured: Content authoring for technology enhanced learning is costly. Thus, there is a focus on *standardized* knowledge that applies to a wide audience. By its very definition, technology enhanced learning

[1]`http://www.imsglobal.org`

enforces the knowledge to be *externalized* into content, also frequently called learning material, e. g. in the form of web based training material.

Lastly, one can observe a tendency towards *structured* knowledge or representations thereof. Standards like the aforementioned SCORM [Lea09] represent the spearhead of this movement: They facilitate the exchange of learning materials between courses by standardizing meta data regarding the sequence of that material.

(Slightly) Exaggerated Conclusion: If one follows this line of thought to the extreme, content and therefore knowledge is treated like source code to facilitate automated processes upon this content such as re-purposing content from one course to another. The meta-data to support these processes is to be created by the content authors in addition to the content itself.

Assumptions with respect to Learners

Following their focus on formal learning settings, e. g. in higher education, traditional technology enhanced learning approaches frequently assume the learner to be a student. Students can be assumed to be motivated to learn and to be focused on learning without distraction.

Following this strong assumption, elaborate models from cognitive science have been applied to model and even predict the student's behavior as he uses the technology enhanced learning system. One example is the cognitive architecture ACT-R that has been applied to technology enhanced learning as described e. g. in [AG01] and [LMA87]. The use of a cognitive architecture allows a system following these approaches to model the cognition of the learner from his interaction with the system.

(Slightly) Exaggerated Conclusion: If one follows this line of thinking to the extreme, the learner is thought of as a computer, whose behavior can be modeled and therefore predicted by a technology enhanced learning system.

Challenging the Assumptions

These assumptions regarding the content and learner in a technology enhanced learning scenario are far from unexpected: In fact, their prevalence follows best practices in computer science, where each of its application domains is typically modeled in a similar manner to the one described above to facilitate its handling through computers. However, the aforementioned assumptions are not always met in many instances where technology enhanced learning could be applied.

First and foremost, not all knowledge can be standardized, nor can it always be externalized in a structured way. The prime example is *implicit knowledge* such as experience, intuition and "know-how". Many activities are based upon implicit knowledge in addition to *explicit knowledge*. In an informal way, explicit knowledge can be defined as textbook, formal, objective or standardized knowledge. Implicit knowledge, on the other hand, is subjective, vague and informal.

Consider the following examples of activities where implicit knowledge is an important aspect:

Example 1 (Cases of Implicit Knowledge)**.**

Bike Riding: *Studying the physics of a bike is not sufficient to be able to successfully ride one.*

Programming: *Programming can only be taught to some extend explicitly through books and university courses. An important aspect in programming is the experience and intuition of the programmer. Additionally, the question of what "elegant code" exactly is, will probably never be answered in an explicit form.*

Game Play: *The rules for games can be spelled out in great detail and very explicitly. However, the skills of* winning *the game are inherently hard to convey explicitly. Instead, this knowledge is built by making experiences with the game.*

In addition to implicit knowledge that is impossible to externalize in a standardized, structured way, that process is hard or undesirable in other instances for the following reasons:

Externalizations are costly: Externalizing knowledge is a time consuming and therefore expensive process. Thus, it is frequently omitted for knowledge that is not needed in a similar fashion by a large audience.

Externalizations are easily outdated If the knowledge changes quickly, its externalizations are frequently outdated. Constant updates of the externalizations only add to the cost of creating and maintaining them.

The second assumptions of the learner being a student restricts technology enhanced learning approaches to formal, explicit learning scenarios such as courses in higher education institutions. This restriction excludes important aspects of life long learning, most importantly learning-while-doing.

However, the implicit and hard to externalize knowledge is typically transferred in learning-while-doing scenarios in traditional learning, namely through the apprenticeship. Thus, the exclusion of this knowledge from traditional technology enhanced learning and an its assumptions about the learner being as student as opposed to an apprentice are congruent.

Therefore, we define the following problem to be discussed in this thesis:

Problem Statement: The field of technology enhanced learning is faced with a large body of important and valuable knowledge that is not amenable to the current technology enhanced learning methodology as it is not externalized in a way suitable to traditional approaches in the field.

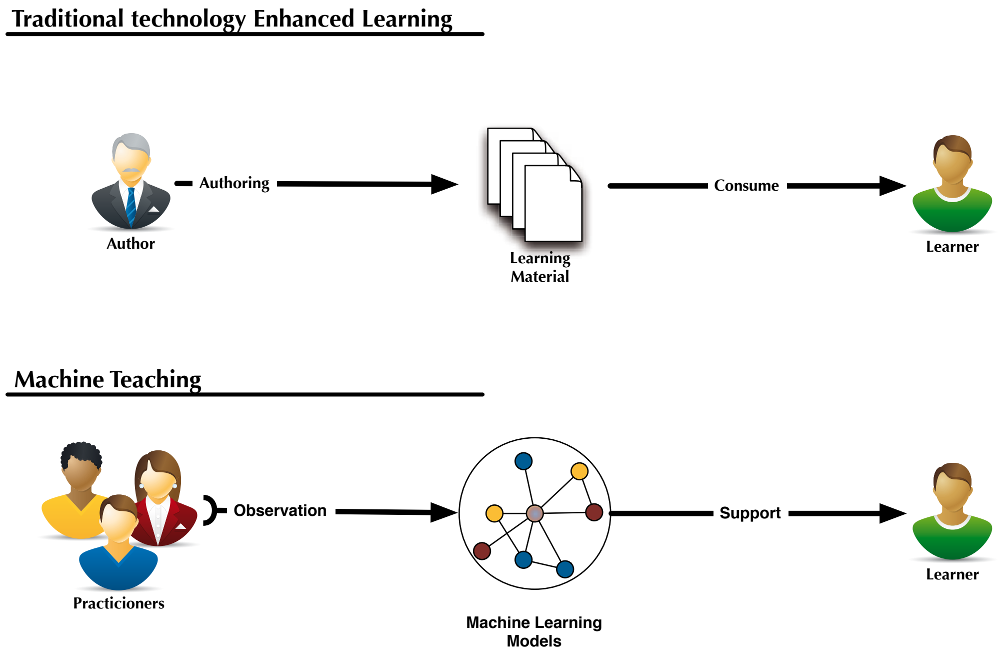


Figure 1.1: Visualization of Machine Teaching as an alternative to the traditional technology enhanced learning approach.

1.2 Approach taken

Reconsider the example of programming from above: Even though the knowledge needed to program well is partially implicit, the resulting source code is easily available and serves as a trace of that knowledge. We will refer to knowledge whose traces are observable as PRACTICED KNOWLEDGE:

Definition 1 (Practiced Knowledge)**.** PRACTICED KNOWLEDGE *denotes all knowledge that contributed to an observable activity. Activities in that sense include but are not limited to the creation of artifacts and the following of informally or formally defined processes.*

Practiced knowledge therefore includes the knowledge in the focus of traditional technology enhanced learning, but is not limited to it: it also consists of the implicit or simply not externalized knowledge needed for the observed activity.

The goal of this thesis is to broaden the scope of technology enhanced learning to practiced knowledge. In order to transfer practiced knowledge, we propose to use MACHINE LEARNING at the core of a novel technology enhanced learning approach named MACHINE TEACHING.

For the purpose of this thesis, machine learning shall be defined as follows:

Definition 2. *A* MACHINE LEARNING *system builds models from data. The process of building these models is called (machine) learning or training. The data which is used in training is therefore called* TRAINING DATA.

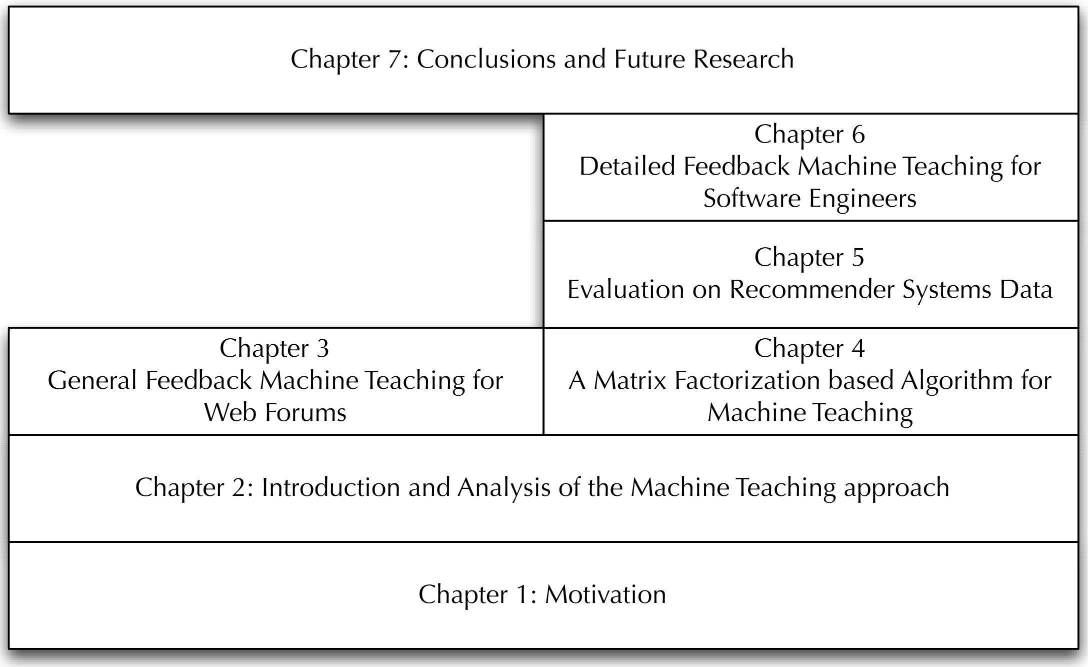

Chapter 7: Conclusions and Future Research
Chapter 6
Detailed Feedback Machine Teaching for Software Engineers
Chapter 5
Evaluation on Recommender Systems Data
Chapter 3
General Feedback Machine Teaching for Web Forums
Chapter 4
A Matrix Factorization based Algorithm for Machine Teaching
Chapter 2: Introduction and Analysis of the Machine Teaching approach
Chapter 1: Motivation

Figure 1.2: Structure of this thesis

The key idea of our approach is to use the observations of successful activities as the training data of machine learning systems. Therefore, a model is built that captures the structure of these observations. This model is then brought into the context of the learner to retrieve feedback, suggestions or general support from it.

One way of visualizing the relationship between traditional technology enhanced learning approaches and Machine Teaching is depicted in Figure 1.1: Where traditional technology enhanced learning presents the learner (student) learning material *authored* for this purpose, Machine Teaching resorts to *automatically learned models* build from data about the activity of practitioners.

Research Question: We hypothesize that machine learning models are capable of thereby transferring practiced knowledge. The validity of this hypothesis shall be the topic of this thesis.

1.3 Organization of this Thesis

Figure 1.2 visualizes the structure of this thesis. Chapter 2 introduces the Machine Teaching approach to technology Enhanced Learning more formally, after a brief introduction to machine learning.

The definition of the approach is followed by an theoretical analysis of its properties and possible applications in the same chapter. An analysis of the major components of a hypothetical technology enhanced learning system following this approach is used to identify research results needed in order to facilitate the realization of such a system:

Sensors that allow the Machine Teaching system to monitor both the learners and experts.

Machine Learning Models capable of capturing the knowledge from the sensor data in order to provide the learner with meaningful feedback.

For the purpose of this thesis, the sensors are tied to the artifacts created by the learners and experts alike. The remainder of the thesis thus focuses on investigating and developing machine learning models for Machine Teaching.

Machine Learning for Machine Teaching

The first step in this study is presented in Chapter 3, where a system is presented that supports the learner with automated ratings of her web forum posts. These ratings are computed based upon a machine learning model built from web forum posts that have been rated by the users of that web forum. From a machine learning perspective, such a system is built using state-of-the-art supervised machine learning methods.

The good results of machine learning on that task encourage the analysis performed in the Chapters 4, 5 and 6 where detailed feedback in Machine Teaching is investigated. There, the goal is to support the learner with feedback regarding her artifact instead of merely rating it.

Chapter 4 presents a novel matrix factorization based machine learning model and algorithm equally suited for this task and the more broadly known task of Recommender Systems. The algorithm is evaluated in Chapter 5 on data sets from the recommender systems literature with promising results.

In Chapter 6, this model and algorithm are applied to build a Detailed Feedback Machine Teaching approach for software engineers. Empirical evaluations show that the system is indeed capable of supporting a software engineer with meaningful suggestions.

Based on results presented in Chapter 6 and Chapter 3 it is safe to conclude that machine learning can in fact be used to build Machine Teaching systems.

1.4 Contributions of this Thesis

This thesis investigates using machine learning to help convey implicit knowledge. It thereby makes the following contributions to the state of the art in machine learning and technology enhanced learning.

Scientific Contributions

Machine Teaching Approach: This thesis presents a novel approach to technology enhanced learning that facilitates the transfer of implicit knowledge. The approach takes into account different scenarios of general and detailed learner feedback.

Automatic Quality Assessment of Text: The ability to write appropriately for a community is a skill that cannot be transferred to explicit knowledge and thus is not amenable to the traditional technology enhanced learning methodology. However, it is shown that the quality judgment of a community can be replicated by a machine in order to facilitate the self-learning of this skill. This work has been published in [WG07] and [WGM07]; the feature extraction software developed as part of this has been described in [GMM$^+$07].

A generalized Matrix Factorization Method: Many instances of the Machine Teaching approach can be encoded into matrix factorization problems, much like recommender systems. Where the latter suggest items to users, the former suggest actions to take or attributes of artifacts to be created. This thesis introduces a new model and algorithm for this task and makes the following contributions to this field:

1. A generalization of matrix factorization models to per-row loss functions and
2. an optimization procedure to do so efficiently.
3. A procedure for the direct optimization of the Normalized Discounted Cumulative Gain (NDCG) ranking score.
4. An algorithm for the computation of the ordinal regression loss in $\mathcal{O}(n \log n)$ time as opposed to the algorithms of $\mathcal{O}(n^2)$ time complexity that have been previously known in the literature.
5. An extension of the matrix factorization approach to hybrid recommenders that can use features in addition to the user-item interaction data used in collaborative filtering recommender systems.
6. Adaptive regularization for matrix factorization.
7. The integration of a graph kernel to model the binary interaction e.g. between users and movies in addition to the ratings provided by the users.

The algorithm has been evaluated on recommender systems data with favorable results. The model, algorithm and extensions have been published in [WKLS08], [WKS08b], [WKS08c] and [WKS08a]. The first version of the system with the NDCG loss function was presented at NIPS 2007 under the name CofiRank in the paper [WKLS08]. It was the first collaborative filtering system capable of predicting rankings as opposed to ratings and will be presented in the Preference Learning book [KW10]. The recent paper [WKB09] at the ACM Recommender

Systems Conference presents the results of the application of the Machine Teaching approach to the software development domain.

A machine teaching approach for programming: Learning to program with a new code library is a challenging task for software developers. It is shown that a code recommender system can be built for this task based on the matrix factorization algorithm introduced above that is capable of easing this process.

Data and Software Contributions

During this thesis, the following additional contributions have been made to the scientific community:

Software: An implementation of the matrix factorization algorithm in C++ has been found to be one of the fastest available. It has been released as Open Source Software and is available from the project website `http://cofirank.org` for download.

Data Sets: Most of the experiments reported in this thesis were conducted on publicly available data sets. Following this example, the data set used for the evaluation in the software engineering application has been released on the project website, too. The data set is described in detail in Section 6.4 and consists of caller-callee relations mined from the Eclipse source code calling the SWT user interface framework.

2 The Machine Teaching Approach

Contents

The goal of this chapter is two-fold: First, MACHINE TEACHING is introduced as a new technology enhanced learning approach which emphasizes on conveying practiced knowledge. Second, this chapter also presents the research focus of this thesis within the broader approach of Machine Teaching.

The chapter is therefore structured as follows: Machine Learning is introduced in Section 2.1 to facilitate the discussion in subsequent sections. Machine Teaching is then formally defined and its properties are analyzed based upon that definition in Section 2.2. The major components of a Machine Teaching system are described in Section 2.3, including the relation between machine learning as an enabling technology and Machine Teaching. The remainder of this chapter, namely the Sections 2.4 and 2.5, introduce two levels of feedback a Machine Teaching system can provide, general and detailed feedback.

2.1 Preliminaries: A Brief Introduction to Machine Learning

In this section, we present a brief overview of machine learning to facilitate further discussion of the Machine Teaching approach in subsequent parts of this thesis. Therefore, it is decidedly high-level. For a more detailed description of machine learning see e. g. the books [Bis06], [SS02], and [WF05].

Recall the definition of machine learning as introduced in Chapter 1:

Definition 3. *A* MACHINE LEARNING *system builds models from data. The process of building these models is called (machine) learning or training. The data which is used in training is therefore called* TRAINING DATA.

Machine Learning as a field of research therefore concerns itself with the study of problems to be addressed through machine learning, the development of machine learning models and methods to learn these models from data. Below, we will give a brief overview of these aspects to machine learning.

2.1.1 Machine Learning Problems

Machine learning is applied to a wide range of problems where an explicit, hand coded solution is hard or undesirable to obtain. This section provides a classification of the major machine learning problems to show the breadth of the approach and therefore hint towards the range of practiced knowledge it can be applied to in Machine Teaching.

The most basic categorization of machine learning problems in that of supervised vs. unsupervised learning:

Supervised Learning: In this case, a model of an input – output relation is sought. Given training data consisting of a set of pairs (x_i, y_i) of samples $x_i \in \mathbb{X}$ and their labels $y_i \in \mathbb{Y}$, a model is learned that can predict the label y_j for a previously unseen sample x_j.

An ubiquitous example of a supervised machine learning system is that of a email spam filter: Given sufficient data about spam and ham (not spam) messages, a model is sought to predict the label (spam or ham) for new email messages.

Unsupervised Learning: The goal of unsupervised learning systems is to uncover patterns in raw data, e.g. by *clustering* the samples x_i.

Unsupervised learning is often applied to data mining applications such as business intelligence.

For the purpose of Machine Teaching, supervised learning settings are the more important ones, as we seek to apply the models to convey mined knowledge between users while an unsupervised machine learning system is primarily used to deduce new insights from data.

Machine learning approaches have been developed for a wide range of problems. Some prominent examples shall be introduced below:

Regression: In this case, the labels y_i are real numbers, $y_i \in \mathbb{R}$. Thus, the machine learning system effectively learns a function $f : \mathbb{X} \rightarrow \mathbb{R}$ such that $f(x_j)$ is a prediction of the true value of y_j.

One Class Classification: All samples given to the system are proper examples. The goal is then to build a model from these examples that is capable of identifying samples that do not "fit in" with the standard set by these examples.

A typical application domain for one class classification arises as a machine learning problem is the detection of credit card fraud: The majority of the transactions is assumed to be valid and one can therefore build a model from them. If a new transaction is inexplicable through the model, it may raise concerns.

Classification: In this case, the labels y_i are taken from a set of possible classes. In the spam filter example, these classes would be $\mathbb{Y} = \{SPAM, HAM\}$. The model sought of a machine learning system in this case can be equated again to a function. The function value $f(x_j)$ is the predicted class of x_j.

Ranking: In this problem, the goal is to rank items based upon ranked training samples. This problem is sometimes also referred to as ordinal classification or ordinal regression, stemming from the fact that the ranking of the items is typically expressed on a ordinal scale.

An obvious application of ranking systems are search engines which can be personalized by learning the ranking function for each user or group of users by observing their past interactions with the search results.

Sequence Prediction: In many instances, the data consists of sequences, e. g. the sequence of web pages visited by a user. A model of such data can be used to predict the likely next step, given the previous steps. Typically, Markov Models are used in this context, where a Markov Model of order k uses the last k steps to predict the next step in the sequence.

Recommender Systems: In this problem, the known data consists of past interactions between users and items. The goal is to predict future interactions between yet unseen user – item pairs.

Depending on the nature of the underlying data, these interactions can give rise to any of the above problems. In shopping basket analysis, the recorded interactions are binary: Whether or not an item has been bought by a user. In the movie recommender systems scenario, the interactions are usually recorded as ratings given on a discrete, ordinal scale.

Density Estimation: In many applications, one is interested in (conditional) probabilities of variables within the data. Failure analysis of complex systems is a popular example: The producer of e. g. a car is often interested in the relationship between different sensory data about a car and the likely cause of a breakdown. Thus, the probability of a breakdown given that sensory data is sought.

Each of these cases has possible applications in Machine Teaching: Regression and Classification approaches can e. g. be used to model quality judgments to support learners with an estimation of the quality of their work. See Section 2.4 below for more details. Recommender systems and density estimation allow a Machine Teaching system to build models of the detailed structure or properties of activities. These models can then be used to support the learner with detailed feedback with respect to his work as introduced in more detail below in Section 2.5.

2.1.2 Machine Learning Models

Above, we gave an introduction to some of the most important problems machine learning is applied to. However, we did not introduce the actual models used for these instances. To facilitate a concise, yet detailed discussion of important aspects of ML models, we restrict ourselves to the example of linear models for binary classification below. Much of the description also applies to other linear models such as those for multi-class classification and regression. A more broad description of machine learning models can be found in [Bis06].

Linear Models

Linear models in general assume the label y to be a linear function of the sample x. The samples are represented by real valued feature vectors ($\mathbb{X} = \mathbb{R}^n$). The process of turning samples into these vectors is called feature extraction: Each dimension of the samples x refers to one feature (attribute) of the sample.

Example 2. *When dealing with text samples (documents), the so called* bag of words *is a basic feature vector. For each document, a vector is constructed where each dimension represents the number of times a certain word is present.*

The text "This is an example within an example." results in a feature vector where the value in the dimensions for "This", "is" and "within" is 1. *The ones for "example" and "an" assume the value of* 2.

Given these feature vectors x, the model then consists of a *weight vector* of equal dimension, typically denoted by $w \in \mathbb{R}^n$. In any linear model, the predicted label y_i^{pred} is then computed as a function h of the scalar (or inner) product between w and the sample x_i:

$$y_i^{pred} = h\left(\langle w, x_i \rangle\right) = h\left(\sum_{k=1}^{k=n} w_k x_{i,k}\right)$$

Linear Models for Binary Classification

In the case of binary classification problems, the labels y are encoded to be either 1 or -1, that is $\mathbb{Y} = \{+1, -1\}$. This allows us to define the function h from above to be the sign of its argument:

$$sign\left(x\right) = \begin{cases} 1 & x >= 0 \\ -1 & else \end{cases}$$

Putting it all together, a linear model for binary classification consists of a weight vector w that is applied to parametrize the prediction function f:

$$y_i^{pred} = f_w(x_i) = sign\left(\langle w, x_i \rangle\right)$$

2.1.3 Machine Learning Methods

Until now, we did not introduce the process of actually finding the model, given training data. The techniques for this part of a machine learning system are referred to as MACHINE LEARNING METHODS. Again, there are many options, even for the one model and problem. We will thus introduce the general concept using the same example as above, linear models for binary classification.

Training a machine learning model can be seen as picking the right function out of a class of possible functions. Given the linear models above, that function class would encompass all linear functions of the n features. And in that case, picking the right function amounts to choosing the weights w as a linear function can be identified with its weight vector.

To choose the "right" function, one needs to define the notion of "right" more formally. The goal of a the training of a machine learning system is to obtain good performance on yet unseen data, as measured by a LOSS FUNCTION $l : \mathbb{Y} \times \mathbb{Y} \to \mathbb{R}$. The loss

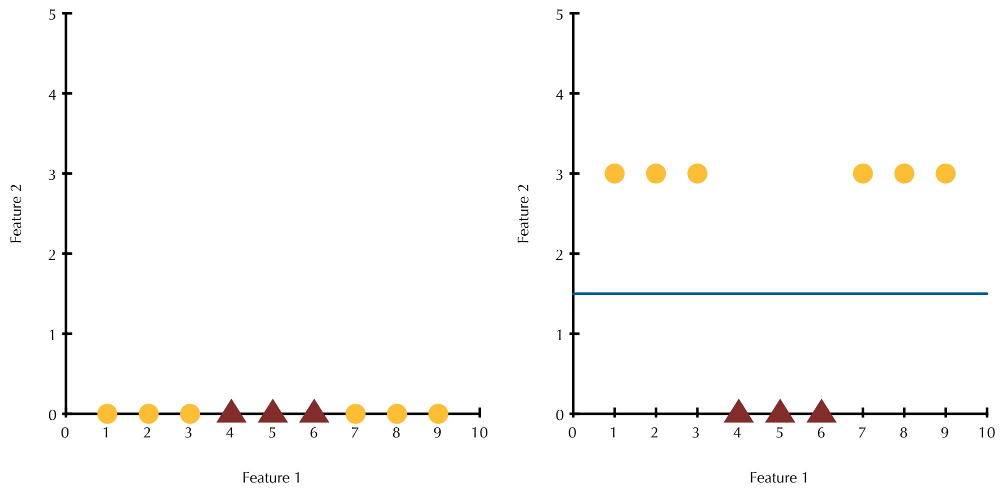


Figure 2.1: **Underfitting:** In the left graph with only one feature, the two classes cannot be separated by a linear function. If another feature is added, as in the right figure, that linear separator can be found as depicted.

function l determines the discrepancy between the predicted label f and its true value y. Section 4.4 presents a number of loss functions in detail. For binary models, the loss function shall indicate the errors made by the model:

$$l(f,y) = \begin{cases} 1 & sign(f) \neq sign(y) \\ 0 & sign(f) = sign(y) \end{cases}$$

Obviously, this future data is not available at training time. Hence, one resorts to minimizing the loss on the available training data. This quantity is referred to as EMPIRICAL RISK and the process therefore as EMPIRICAL RISK MINIMIZATION. The Empirical Risk of the model w is computed as the sum of the losses of the model on the training data X:

$$R_{emp}(w,X) = \sum_{xinX} l(f_w(x),y)$$

In the Empirical Risk Minimization, one faces the following two problems:

Underfitting: In this case, the model cannot capture the fidelity of the underlying data. This often is the result of missing features in the data.

Example: Spam emails can partially be labeled based on the colors used in the email (spam often uses red, while legitimate email doesn't). If the colors used in the text are not part of the features extracted, the machine learning model cannot capture this information and will suffer from poor performance.

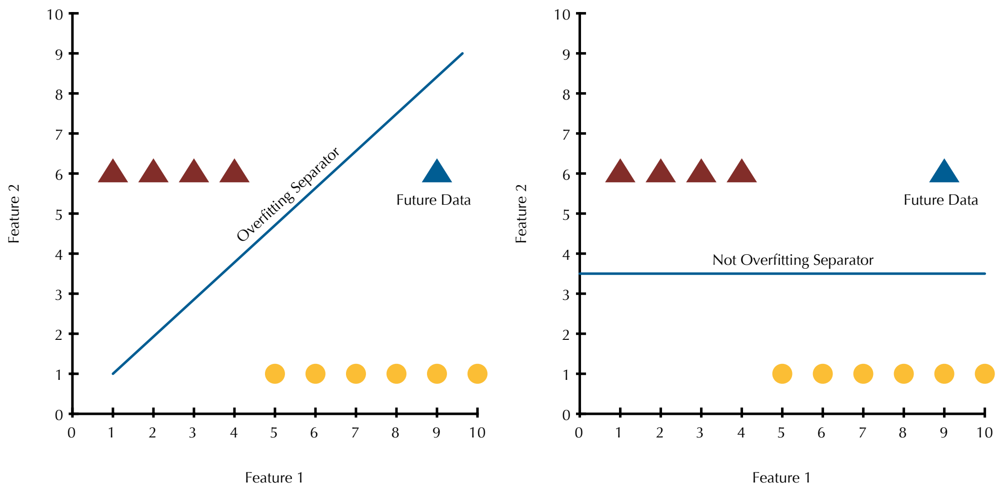


Figure 2.2: **Overfitting:** The separator function in the left graph does explain the training data perfectly well. However, it cannot generalize to the new data point as opposed to the more simple separator depicted in the right graph.

Figure 2.1 depicts this situation with a one dimensional feature space ($\mathbb{X} = \mathbb{R}$) where it's impossible to find a linear function to separate the two classes. Adding another feature ($\mathbb{X} = \mathbb{R}^2$) can resolve this problem.

Overfitting: In this case, the model can encompass more complicated structures than suggested by the data. This leads to the problem where the model explains the training data perfectly, yet does not express the underlying structure properly.

Figure 2.1 visualizes this situation: The separator function on the right is simpler than the one on the right: It separates the data using only Feature 2, as opposed to using both features on the left hand side. It thus is capable of extracting the true underlying structure better than the more complicated model on the left.

These problems are addressed differently for different models. We follow the Regularized Empirical Risk Minimization here, which introduces the following two steps to counter them: First, a sufficiently large model class is used such that a solution can always be found, e. g. through the Kernel Trick introduced below in Section 2.1.4. This eliminates the risk of underfitting. Second, a Regularizer Ω is introduced that measures the model complexity.

Many choices of the regularizer are conceivable. For the sake of concise presentation, we restrict ourselves to the squared L_2 norm (Euclidean norm) here:

$$\Omega(w) = ||w||_2^2 = \frac{1}{2}\sum_{i=1}^{i=n} w_i^2$$

Optimization

The training process of a machine learning model is an *optimization problem*: The empirical risk as well as the regularizer are minimized in a joint objective function:

$$\begin{aligned} O(w, X) &= R_{emp}(w, X) + \lambda\,\Omega(w) && (2.1) \\ &= R_{emp}(w, X) + \frac{\lambda}{2}\sum_{i=1}^{i=n} w_i^2 && (2.2) \end{aligned}$$

Here, λ is a constant that defines the relative trade-off between the model complexity and the loss on the training data. An intuitive explanation of this observation can be given as follows: A model is sought which agrees with the observed training data as much as possible (as measured through the empirical risk), but which on the other hand is as simple as possible (as measured through the regularizer). This follows OCCAM'S RAZOR which states that the simplest explanation that agrees with reality is the most likely one.

The result of the minimization is the model $\hat{w}$ which minimizes the objective function $O(w, X)$:

$$\begin{aligned} \hat{w} &= argmin_w\,(O(w, X)) && (2.3) \\ &= argmin_w\,\left(R_{emp}(w, X) + \lambda\,\Omega(w)\right) && (2.4) \end{aligned}$$

To facilitate efficient optimization, the loss function is (re-)formulated as a convex function in the prediction f to facilitate efficient optimization. If the loss function is convex in the prediction f, it is also convex in w for linear models. As the L_2 norm is convex in w, too, the whole training process then amounts to minimizing a convex function to find the model w which minimized the objective function (2.1).

In the binary classification case, the loss function is typically formulated as the HINGE LOSS:

$$l_{Hinge}(f, y) = \max(0, 1 - f\,y)$$

This loss vanishes if the prediction f and the truth y agree. Additionally, the Hinge Loss is a linear and therefore convex function in f. See Section 4.4.1 for a detailed description of the hinge loss. In Section 4.4 within the same Chapter, more examples of loss functions are given, in particular for regression and ranking problems.

Given a convex loss function, the process of training the machine learning model has therefore been identified with that of optimizing a convex function. Numerous algorithms are available for this task.

2.1.4 The Kernel Trick

The linear models above are of rather limiting expressiveness: If the data can only be linearly separated based upon the joint observation of two or more features, no linear model can be found that separates the data.

To heal this, one would have to define an additional feature that encodes the co-occurrence of the two other features to make the data linearly separable. The feature engineering therefore encodes non-linear features of the domain into dimensions of the feature space to facilitate a linear model. This obviously is undesirable as it introduces a huge number of features of questionable relevance.

The concept of a kernel based algorithm as described e. g. in [SS02, Vap95, VGS97, Vap98] generalizes this idea to effectively turn most linear models to non-linear models. The process is commonly referred to as the KERNEL TRICK. To apply it to a linear model, one follows these steps:

1. One needs a formulation of the prediction rule as well as the optimization algorithm that does not operate on the samples x directly, but instead only uses inner products $\langle x_i, x_j \rangle$ between samples.

   The inner products can be regarded as a measure of similarity between the samples. Example: Let x_i and x_j be bag-of-words representations of texts as in the example above. The inner product $\langle x_i, x_j \rangle$ then increases if x_i and x_j share more words. It assumes the value of 0 if the texts do not share any word.

2. The invocation of the inner products $\langle x_i, x_j \rangle$ are then replaced with those of a KERNEL FUNCTION $k(x_i, x_j)$. This function gives rise to a gram matrix $K_{i,j} = k\left(x_i, x_j\right)$. If that matrix is positive semi-definite $\left(xKx^\top \geq 0\, \forall x \in \mathbb{R}^n\right)$, it can be shown that the kernelized algorithm can be equated to a linear algorithm operating in a space induced by that kernel.

The net effect of the application of this trick is: Many models that are linear in the samples x can be transformed into models that are not linear in these samples. Thus, the trick has been applied to a wide variety of models to broaden their applicability. Major implementations can e. g. be found in the software package `kernlab` [KSH09, KSHZ04].

2.2 Introducing Machine Teaching

Building upon the general idea introduced in Chapter 1, Machine Teaching in the broadest sense shall be defined as follows:

2.2.1 Definition

Definition 4 (Machine Teaching). MACHINE TEACHING *conveys* PRACTICED KNOWLEDGE *through machine learning models built from observational data of the application of that knowledge.*

It follows immediately from this definition that any machine learning model and method can be applied to in this sense. Consider the following hypothetical examples of Machine Teaching applications:

Craft: A sequence prediction model can be built from observing experienced craftsmen. That model contains the knowledge about work sequences and can be applied to support other craftsmen by providing hints on possible "next steps" in their work.

Photography: Given the exposure data and e. g. light sensor input of a corpus of images, a regression model can be trained to reflect the common exposure settings used in certain light.

Writing in Novel Text Genres: Many genres of text found e. g. in communities on the world wide web expose new styles of writing, spelling and even grammar. A machine learned structure model of this grammar can be applied to support people new to the community in writing for it.

Machine Programming: Many machines nowadays are controlled by computers and are therefore programmed for each product. Creating these programs requires skill, experience and intuition. The performance of the programs is, however, rather explicit: The time it takes to produce the desired product, the amount of wear and tear the production induced, etc. A machine learning model of these programs can be used to support new programmers and to foster the reuse of knowledge from successful programs.

These examples show how broad the applicability of the approach is. In this thesis, we focus on a subset of the possible applications where the machine learning model is used to provide feedback to the learner in a way similar to the master in an apprenticeship or master-student situation:

Definition 5 (Apprenticeship Machine Teaching)**.** *An* APPRENTICESHIP MACHINE TEACHING *system supports learners during activities by providing ratings of and/or suggestions regarding these activities. It follows the concept of an apprenticeship where the apprentice is offered ratings of and/or suggestions regarding her work from one or several experienced practitioners.*

To provide these ratings and suggestions, a Machine Teaching system needs a machine model of the activity in question which captures the knowledge needed for this activity. The models are extracted from past observations of the same or similar activities by more experienced people by means of machine learning.

Based upon these models and observational data about the activity of a learner, the Machine Teaching system generates ratings and/or suggestions and presents them to this learner.

Note that the term *learning* is overloaded in this thesis. It may either refer to the human learning or to the learning in the machine learning sense. Thus, we define:

Definition 6 (Learning)**.** *To distinguish between the two meanings of* learning*, we use the following terms wherever the meaning is not clear from the context:*

Learning *denotes the learning of the humans, frequently called learners.*

Machine Learning *denotes the model building through a machine learning algorithm.*

The same nomenclature is applied to the verb "to learn", where we introduce the form "to machine learn".

2.2.2 High-Level Example of a Machine Teaching Scenario

Consider the following example taken from Chapter 6 to illustrate the idea of Machine Teaching:

Many of the practices in a team of programmers are never written down. Assume that the practices include:

> Whenever something is written to the database, we put an entry to the log file starting with "`Database Access:`".

A programmer new to the team will most probably hear about this practice once she fails to adhere to it: The fellow programmers will point out that mistake and the programmer will adhere to this practice in the future.

The latter process, pointing out the error, is where Machine Teaching is introduced: Given enough code, a model of that code can be machine learned. This model encompasses the practice quoted above. Then, by observation of the code of the new programmer, the system can point out instances where the code does not match the machine learned model and thereby the practices formed by the team.

In this process, the Machine Teaching system assumes the role of the fellow programmers in the current process. It is also apparent that the Machine Teaching system does not require the programming team to define or otherwise externalize their practices. Nor do the programmers have to provide the system with "model code" to learn from. Instead, the Machine Teaching system analyzed the code they already produced. This significantly lowers the effort needed to deploy this Machine Teaching system when compared to a more traditional technology enhanced learning system.

2.2.3 Machine Teaching Properties

Now that Machine Teaching has been defined, this and the following section provide an analysis of the approach which follows from these definitions. In this section, the properties of such a system are discussed, also contrasting them to those of a traditional technology enhanced learning system. The section thereafter will explicate the assumptions regarding the learner and the scenario that underlie the Machine Teaching definition above.

It immediately follows from its definition that *a Machine Teaching system is not dependent on externalized knowledge.* The following paragraphs will introduce and discuss additional important properties of any Machine Teaching system that falls within the scope of the definition above, while subsequent sections give insights on more specific instances of Machine Teaching systems.

Machine Teaching can operate on non standardized knowledge: Machine Teaching extracts the model from observational data. Thus, the knowledge that is needed to perform the observed activities needs not to be standardized. Depending on the machine learning model used and the amount of data available, even conflicting observations can be co-existing in a Machine Teaching system.

Machine Teaching is focused on the practice, not the ideal: A Machine Teaching system operates on observations of activities and therefore has no access to the ideal way of performing these activities. Such an ideal view would typically be found in traditional teaching materials such as textbooks and instructional videos.

Thus, a Machine Teaching system captures and subsequently teaches a different quality of the activities when compared to traditional learning material: How they *are* done as opposed to how they *should be* done.

Machine Teaching is geared towards long-time use: Typical technology enhanced learning tools such as an online course are focused on teaching the needed knowledge in a comparatively short period of time. Machine Teaching, on the other hand, is more suitable in a long term setting: It provides feedback to the learner based on her activities. As these change over time, a Machine Teaching system can accompany the learner through different learning tasks, possibly even with an *ambient learning* setting.

Depending on the use of a Machine Teaching system, it may constantly machine learn by updating its model to the observed practices, too. Thus, not only the human learning is long-term, the machine learning is, too.

A Machine Teaching system makes mistakes: Even if the used machine learning models capture the observed activities perfectly and make no mistakes (an unlikely condition), these activities need not be executed perfectly at all. Thus, mistakes of a Machine Teaching system are to be expected just as mistakes of the humans observed are to be expected, too.

However, these mistakes do not inhibit learning: The Machine Teaching system makes these mistakes based upon vast amounts of observations of past activities. Therefore, even if the Machine Teaching system makes an objectively false suggestion, it may still provide the learner with the information that her current activity is different from the mainstream as extracted from these observations. That information alone can therefore trigger important reflections within the learner.

Given these properties, it becomes apparent that Machine Teaching not only does away with the dependence on externalized, possibly structured knowledge but also exhibits teaching qualities which are new to the field of technology enhanced learning, such as being more suitable to teaching the practice as opposed to the ideal.

It also became clear from this analysis that the performance of any future Machine Teaching system is crucially dependent on the quality of the available machine learning models and methods.

2.2.4 Machine Teaching Assumptions

In addition to these attributes of a Machine Teaching system, its definition also exhibits certain assumptions regarding its applicability:

Data availability: Departing from traditional technology enhanced learning, Machine Teaching does not require externalized or even formalized knowledge. Instead, it requires observational data to build its machine model. Thus, it can only be applied to domains where that data is available or can be gathered easily.

Availability of suitable machine learning techniques: As Machine Teaching relies on machine learning models and methods at its core, only practiced knowledge that can be represented in those models can be conveyed through Machine Teaching system. However, any advance in the breadth and depth of available machine learning techniques also adds new potential to the Machine Teaching approach.

Example: Machine translation is one active field of research in machine learning. However, the techniques to machine learn to translate texts from pairs of translated texts have not matured enough to be considered for a Machine Teaching system. Once substantial progress is made in this field, a Machine Teaching system could be built that supports the education of human translators.

Learner Experience: An Apprenticeship Machine Teaching system provides ratings of and / or suggestions regarding activities of the learner as observed by the system. It follows immediately that the learner needs to be capable at least of an attempt of the said activity. Otherwise, the Machine Teaching system will be unable to provide meaningful feedback. Thus, the learners who use a Machine Teaching system cannot be complete novices of the field.

Situations where observational data is easily available and where learners have at least minimal experience are plentiful, especially in the envisioned long-term usage scenarios.

2.3 Major Components of a Machine Teaching System

Departing from the rather abstract level of the definition and theoretical analysis of Machine Teaching, a more systems-oriented perspective is taken in this section. The following introduces the major components of a concrete Machine Teaching system and their relations. This will not only facilitate a more detailed discussion in the remainder of this chapter but also provide the basis for an analysis of the research needed in order to make Machine Teaching feasible.

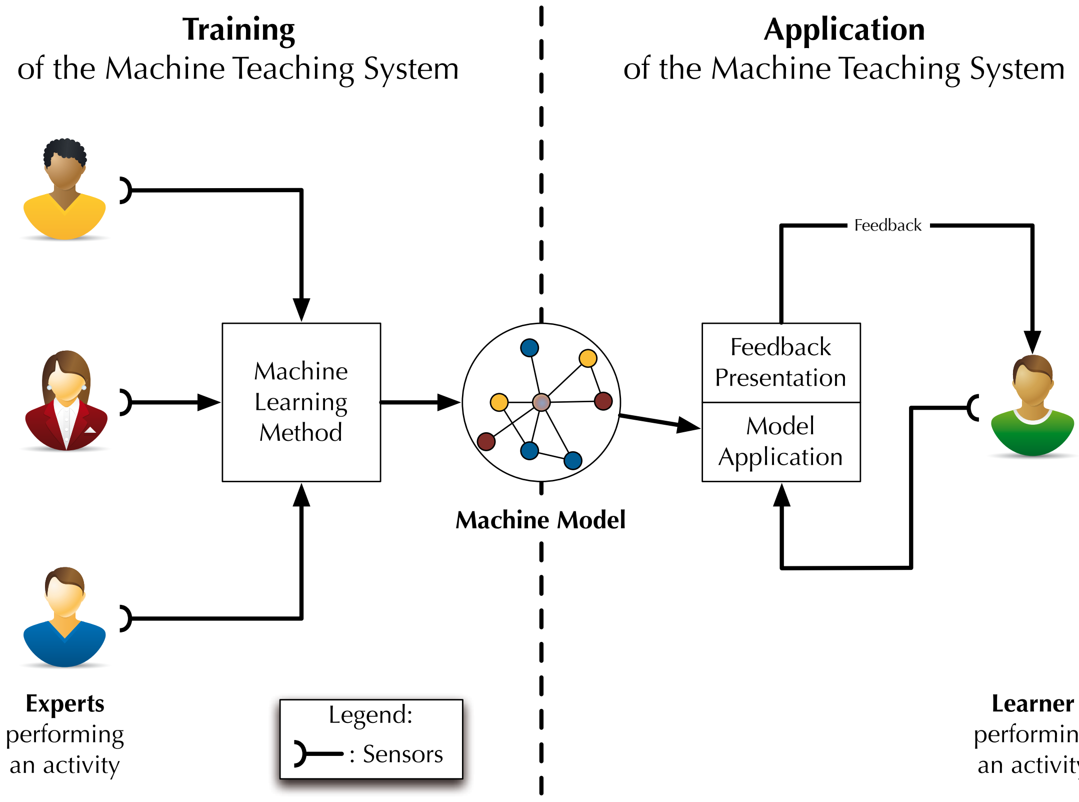


Figure 2.3: Major components of a Machine Teaching system. Note that the distinction of users into Experts and Learners is introduced for clarity of presentation only.

Here, the components of a Machine Teaching system are introduced in two steps: First, they are described by themselves. Second, their dynamic interplay will be discussed. Figure 2.3 shows the major components of a Machine Teaching system:

The Experts: These are the people that are observed by the system in order to machine learn a model of the activity observed. The assumption regarding the experts is that they perform the observed activity well enough to serve as an example for the learner.

How to choose these experts in an application of Machine Teaching is specific to that application. In the software engineering example above, the experts are the fellow programming team members of the learner as they have expert knowledge on the practices established by that team.

The Learner: As introduced above, the learner is able to perform the activity to some extent, but requests or is presented assistance regarding her current activity. Thus, the learner may either be in an active role with respect to the Machine Teaching system or in the role of a consumer.

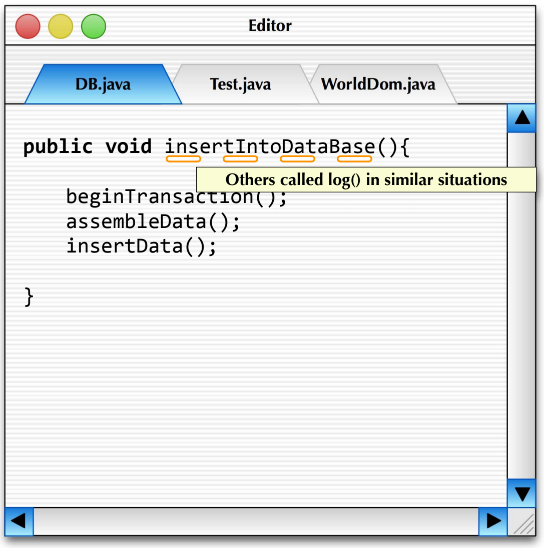
Note regarding these roles: Note that these two roles – experts and learners – are not mutually exclusive. In fact, a learner may very well contribute to the system as an expert, too. This may either be the case if the users of the system are experts in one part of the practiced knowledge and learners in another. Or, more interestingly, the Machine Teaching system could be used to accelerate consent-finding within a group of peers: All activities of all users contribute to the machine model of the practiced knowledge. And all users receive feedback from the system based upon that model which will yield consensual behavior of the users group.

The Sensors: Both the experts and the learner are monitored through these in order to provide the Machine Teaching system with the observational data. The sensors need to be able to capture those attributes of the activity needed to perform the kind of assistance sought of the Machine Teaching system. While the sensors externalize data about the activity, this data hardly resembles knowledge.

Note that the sensors need not to be physical: In the Software Engineering example above, the sensors are formed by code analysis software.

The Machine Learning Method and Model: These two components are inter-dependent and thus are presented together. Given the sensor input, the machine learning method is used to learn the machine learning model. A chosen machine learning model can only be machine learned by a certain set of machine learning methods, hence the interdependence between the two.

The machine learning model (or machine model) is in principle chosen separately for each application of Machine Teaching. However, we will introduce two major classes of Machine Teaching scenarios below, namely general and detailed feedback, as well as appropriate machine model choices for both of them.



```
public void insertIntoDataBase(){

    beginTransaction();
    assembleData();
    insertData();

}
```

Figure 2.4: Mockup of the user interface of a Machine Teaching System for programmers.

Machine Teaching and machine learning are connected, as the abilities of the machine learning method and model define the abilities of a Machine Teaching system. Any progress made regarding the accuracy, speed and expressive power of the underlying machine learning techniques directly reflects upon the same properties of Machine Teaching approaches built upon these techniques.

The Feedback Generation Module: In this module, the observational data of the learner's activity is analyzed in order to provide ratings of and/or suggestions regarding this activity. This module can be thought of as a two layered system:

The lower level consists of the application logic of the machine learning model to new data. This level gets the sensor data as input and provides the higher level with its output, namely predicted rating of and / or suggestions regarding the observed activity.

The higher level is responsible of presenting this information to the user. In the software development example above, this could e. g. happen through a subtle hint as envisioned in Figure 2.4.

After describing the components of a Machine Teaching system, the following will introduce the dynamic interplay of these components.

2.3.1 Dynamics of a Machine Teaching System

There are two phases to be considered in the analysis of the dynamics of a Machine Teaching system: The training phase and the application phase.

Training Phase

In this phase, the system is presented observational data to machine learn a model of these observations. To do so, the sensory input first needs to be made accessible to the underlying machine learning model and method. Subsequently, the actual training of the machine learning model through the machine learning method can occur, either in what is called offline or online learning:

Depending on the nature of the application of Machine Teaching, the observational data may either be available as one batch to train the system or arrive as a constant stream of data. The first case is called offline or batch learning in the machine learning literature. The second case refers to online learning, which makes it possible for the system to constantly update its model upon the arrival of new data.

Obviously, the feedback generated by the Machine Teaching system cannot be ensured to be constant and therefore predictable by the learner in the online learning case. In fact, the system may very well contradict itself after machine learning from new data. These inconsistencies could inhibit the learning process. On the other hand, an online method facilitates a more current tracking of the practices observed by the Machine Teaching system which would lead to fewer inconsistencies between the feedback provided by the Machine Teaching system to the learner and her observation of the practices of experts. Thus, the designer of a specific Machine Teaching system faces a trade-off between constant, predictable feedback and current feedback.

Application Phase

In this phase, the Machine Teaching system is presented observational data of the learner's activity and is sought to provide suggestions regarding and/or ratings of this activity. This can be thought of as a two-step process:

1. Potential feedback is derived from the machine learning model.
2. This feedback is presented to the learner.

The first step is again dependent upon the chosen machine learning model. But in addition to this dependence, it also poses new requirements above the mere application of a machine learned model to new data: The learner needs instantaneous feedback while in many other applications of machine learning, the results of the application of the model can be precomputed.

The second step does in principle not differ from the same step in a system where the feedback is not build upon machine learned models, but upon formalized knowledge embodied in the system. Example: In the software engineering example, it does not matter to the presentation layer whether the desired co-occurrence of logging and database access is machine learned from data or hand coded as rules.

Performance of a Machine Teaching system: It became apparent not only from the systems-oriented view above but also from the analysis provided earlier in this chapter that the performance of a Machine Teaching system is largely determined by the performance of the machine learning method and model that supports it. All other factors such as user interface, data availability etc. being equal, a machine learning method yielding higher accuracy, less error or better predictions in general will also yield a more satisfying Machine Teaching performance. Therefore, it is prudent to evaluate machine learning models in Machine Teaching using the same or very similar techniques to those used in other machine learning applications.

2.3.2 Focus of this Thesis

So far, we have introduced Machine Teaching as a new approach to technology enhanced learning. The remainder of this thesis presents a first step in investigating this new approach by studying its feasibility from a technological point of view.

To do so, we restrict ourselves to a critical set of components of a Machine Teaching system to study: We use the artifacts created by learners and experts as sensors and omit the feedback presentation layer from the analysis in the remainder of this thesis. The main research question in this thesis then is whether suitable machine learning models can be found or developed to support Machine Teaching systems in the future. The following paragraphs present the reasons for these choices.

Presentation of the feedback

The only component introduced above which is not discussed below is the presentation of the feedback to the learner. As mentioned earlier, this component does not differ significantly from the very same component in systems where the suggestions given by the system are hand-coded. Thus, it is safe to assume that such a module is easily devised for a concrete Machine Teaching system.

The Sensors

For the purpose of this thesis, the sensors shall monitor the activities of users, learners and experts alike, through their artifacts. While other sensor setups that monitor processes are conceivable, they also introduce severe challenges of their own that distract from the research challenges of Machine Teaching addressed in this thesis. The technology to reliably monitor human actions and correctly label those observations with activities is still in the realm of research itself, dealing with challenges in sensor technology, ambiguity in human action and not the least privacy concerns of the users. See e. g. [BMRC09] for an approach to classify group activities and e. g. [ZWS09] for an approach to classify the activity of a single user based on sensors worn by that user.

Using artifacts to monitor the users does not suffer these drawbacks. For many decades, whole areas of computer science such as computer vision, natural language processing and speech analysis are devoted to the analysis of artifacts. Thus, a Machine Teaching system can tap into the results of these fields as a resource for the analysis of the artifacts.

Many artifacts are created to serve a purpose. This also means that the extent to which they serve this purpose, their quality, can be used as a factor to choose the artifacts to train the Machine Teaching System on.

Additionally, artifacts can serve as an anchor in a multi-disciplinary collaboration: When people from different disciplines of expertise contribute to an artifact, the artifact and therefore the model machine learned from it contains representations of the knowledge from all these people and their respective disciplines. If that model is then applied to a situation where one or more of the disciplines are missing, the model can provide important pointers to knowledge from other fields.

Machine Learning Method and Model

The main open question in studying the feasibility of Machine Teaching is whether or not a machine learning model can capture the knowledge embodied in the artifacts. Following the task of a Machine Teaching system as defined above, we distinguish two levels of Machine Teaching: General Feedback Machine Teaching which aims at providing grade-like *ratings* of the artifact and Detailed Feedback Machine Teaching which aims at providing *suggestions* regarding the artifacts.

The separation not only follows the two tasks of a Machine Teaching system, it is also justified from the state-of-the art of machine learning. As we will show below, general feedback can be devised using state-of-the-art machine learning techniques while the provision of detailed feedback is more of a challenge to that field. Thus, general feedback can serve as a testbed for Machine Teaching while detailed feedback needs further research regarding the machine learning techniques used.

The following two sections will describe these two levels of feedback in Machine Teaching in detail, including appropriate machine learning methods and models. Both of these sections conclude with the identified research questions in there respective area to form the agenda of the remainder of this thesis.

2.4 General Feedback Machine Teaching

Closely following the definition of Machine Teaching, General Feedback Machine Teaching is defined as:

Definition 7 (General Feedback Machine Teaching)**.** *A* GENERAL FEEDBACK MACHINE TEACHING *system provides the learner with "grade-like" numerical ratings of her artifact. To provide these ratings, a General Feedback Machine Teaching system extracts a model of the ratings from similar artifacts rated by humans. Based upon this model and the artifact presented to the system by the learner, the General Feedback Machine Teaching system generates ratings and presents them to the learner.*

It is important to note the difference between a system matching this definition and already known systems: Computers have long been capable of rating artifacts, e. g. by matching them against a fixed set of rules. The spell checking facility found in just about any modern desktop application is a good example of such a system. Machine Teaching differs from these systems, as the rating computed is no longer based on knowledge embodied in the system, but on models machine learned from rated artifacts.

Appropriate Machine Learning Models

In this section, we identify machine learning technology needed to build a General Feedback Machine Teaching system. From a machine learning perspective, the goal of a General Feedback Machine Teaching system can be rephrased as: Given a set of rated artifacts, find a model that is capable of predicting the rating for a yet unseen, unrated artifact.

This is virtually identical to the task of supervised machine learning systems as introduced above in Section 2.1. Depending on the type of the ratings, different approaches such as classification or regression are appropriate.

Feature Engineering

Supervised machine learning approaches need a digital representation (denoted by x in the definition above) of the artifacts. Typically, FEATURE VECTORS are used: One vector represents one artifact. Each dimension of the vector refers to one feature (property, attribute) of the artifact and contains the value of that feature for this artifact.

Consider the following example from the field of Natural Language Processing (NLP):

Example 3. *When dealing with text, the so called* bag of words *is a commonly used feature vector. For each document, a vector is constructed where each dimension represents the number of times a certain word is present.*

The text "This is an example within an example." results in a feature vector where the value in the dimensions for "This", "is" and "within" is 1. *The ones for "example" and "an" assume the value of* 2.

The process of determining the feature values for an artifact is called FEATURE EXTRACTION, while FEATURE ENGINEERING refers to the finding, development and definition of the features. Much research in fields such as Natural Language Processing and Computer Vision resulted in a multitude of available features in their respective domains.

Key to the prediction performance of a supervised machine learning system and therefore a General Feedback Machine Teaching system is to find appropriate features of the artifacts. In the worst case, where there is no relation between the artifact features extracted and its label, a supervised machine learning system cannot find a function to compute the label from these features.

Research Questions

It also follows that the appropriateness of a feature is task-dependent: A feature that is very successful in systems for one task does not necessarily perform equally well in systems built for other tasks.

As General Feedback Machine Teaching is a new task from the perspective of supervised machine learning, the main research question can be phrased as:

> Can features be found that allow a supervised machine learning system to be part of a General Feedback Machine Teaching system?

This question is addressed via an example in Chapter 3. In that Chapter, features are proposed and evaluated for the task of estimating the rating of web forum posts. It will be shown that features in fact can be found for this task.

To conclude, General Feedback Machine Teaching systems can be built and thus it is prudent to discuss the more ambitious goal of providing the learner with suggestions regarding her artifact through a Detailed Feedback Machine Teaching system.

2.5 Detailed Feedback Machine Teaching

Again, we start the description of Detailed Feedback Machine Teaching with its definition, closely following that of Machine Teaching in general:

Definition 8 (Machine Teaching). *A* DETAILED FEEDBACK MACHINE TEACHING *system supports learners in the creation of artifacts by providing suggestions regarding these artifacts.*

To provide these suggestions, a Detailed Feedback Machine Teaching system needs a machine model of the artifact in question which captures structure of the kind of artifact in question. These models are extracted from similar artifacts created by more experienced people by means of machine learning.

Based upon these models and the artifact presented to the system by the learner, the Machine Teaching system generates suggestions and presents them to this learner.

The remainder of this section discusses the possible choices regarding a machine learning model for this task. We will thereby identify open research questions that are summarized at the end of this section.

Machine Learning Models

To provide suggestions regarding an artifact, a Detailed Feedback Machine Teaching system needs a grasp of the artifact's structure. Similarly to the artifact features introduced above, this structure of the artifact is represented by structural elements of the artifact, which we define as:

Definition 9 (Structural Element)**.** *A* STRUCTURAL ELEMENT *of an artifact is a feature of the artifact that is mutable by the user, respectively learner of the system.*

Structural elements are typed just as features are. Common types include binary, real valued and one-out-of-n.

One artifact may be represented by a vector x of these structural elements, where each dimension corresponds to one structural element. The value of the vector of one artifact in that dimension is the value of the structural element. As an artifact is not required to have all structural elements, the vector may be sparse.

The *goal* of Detailed Feedback Machine Teaching can then be stated as: Given the structural elements of an artifact, suggest changes to the values of these or additional structural elements. Thus, the main difference to the goal of General Feedback Machine Teaching from a machine learning perspective is that instead of predicting a label y for a given sample x, the system is now asked to predict changes to the sample x.

In contrast to General Feedback Machine Teaching, this task does not map directly to one specific machine learning technique. In fact, one could design a statistical model for each artifact type and fit that model to the available data in principle. Graphical models as introduced e. g. in [Bis06] (Chapter 8) provide the conceptual framework to build these models and henceforth machine learning algorithms for them.

The Connection to Recommender Systems

While the task of designing a model for each artifact type seems daunting, the following observation allows us to provide a machine learning model in a large set of Detailed Feedback Machine Teaching settings:

> **Observation:** The process of suggesting (values of) structural elements of artifacts is strikingly similar to that of suggesting products or services to users in a Recommender System.

Recommender Systems are the enabling technology for many e-commerce vendors and have found great research interest. The term "Recommender System" is often only implicitly defined. For the sake of precision, the term shall be defined as follows for the remainder of this thesis:

Definition 10 (Recommender System)**.** *A* RECOMMENDER SYSTEM *suggests items to users. The goal is to suggest those items to a user she is likely to like.*

The recommendations are based upon available data about the user and the items, including interaction data between the users and items, e. g. in the form of user-supplied ratings.

In most cases, a recommendation is computed by first estimating the preference of the user for the items and then presenting those with the highest estimates to the user. This core prediction logic of the system shall be referred to as RECOMMENDER ALGORITHM.

Following this definition, the observation above allows us to define Recommender based Machine teaching as:

Definition 11 (Recommender based Machine Teaching System)**.** *A* RECOMMENDER BASED MACHINE TEACHING SYSTEM *suggests values of structural elements for artifacts. The goal is to suggest those values that fit the artifact well.*

The suggestions are based upon available data about the structural elements and the artifacts, including in particular similar artifacts as represented by their structural elements.

Requirements of a Recommender System

Despite this striking similarity, a Recommender based Machine Teaching system poses a specific set of requirements on the underlying recommender algorithm:

R1: Accurate solutions for partial artifacts: In its application, the Machine Teaching system needs to suggest values for structure elements based on partial artifacts as it is to support the learner during the creation of the artifact. Every time the learner changes the artifact in ways that alter the structural elements, the system needs to compute new or adapted recommendations.

This is similar to the "new user" problem discussed in the Recommender Systems literature: A new user who has rated just a few items expects meaningful suggestions from the recommender system. The presence of new artifacts in this sense is the norm in Machine Teaching scenarios, while it is the exception in a eCommerce recommender system.

R2: Interactive Performance: The Machine Teaching System shall be deployed such that it can provide accurate suggestions just-in-time: The Machine Teaching system must keep up with the speed the learner manipulates the artifact or, at least, the rate at which she demands suggestions from the system.

This is often less of a concern for commercial recommender systems, as the rate at which people buy items is considerably lower than the rate at which learners can manipulate artifacts.

R3: Recommendation Coherence: The recommendations given by the Machine Teaching System may be subject to coherence restrictions such as in the following example:

Example 4. *In the programming example in Section 2.2.2, the system should only recommend an operation on a* `String` *object if an object of that type is either present or its creation is also suggested.*

Note that coherence requirements may be formalized in some cases, while in the more general case they are to be inferred from the data.

R4: Easy Adaptation to Multiple types of prediction: The structure elements of artifacts may be of various types. Thus, the Machine Teaching System needs to be easily adaptable to these changing types. Recommender algorithms, in contrast, are most often developed solely to predict the numerical rating a user would attach to a given item.

Available Recommender Algorithms

The Recommender Systems literature, e. g. in the survey paper [AT05] distinguishes two different approaches: Memory based Recommender Systems and Model based Recommender Systems.

Memory based Recommender Systems:[1] To compute the predicted ratings, these systems search for the users which are most similar to the one the prediction is made for. The notion of similarity can be based on features as defined above, e. g. their age or address, or it is based on past purchase data. Given those similar users, the predictions is computed as the, potentially weighted, average of the labels given to the item in question by those users.

The same approach can be applied from an item perspective: A list of items is retrieved that are similar to the one in question. The weighted average over the labels "given" to the user in question is the prediction of the system.

Model based Recommender Systems: Given past data, these systems build a model of the function that associates a user-item pair with a label. One approach to do so is to derive a function from the features of the users and items to the label. As for Memory based Recommender Systems, the model may also be defined on the past purchase data of this and other users. Systems in that category are referred to as *Collaborative Filtering Systems*. Systems that make use of both user and item features and the collaborative information are referred to as *Hybrid Recommender Systems*.

We will now investigate the suitability of these classes of Recommender Systems in the light of the requirements described above.

Memory based recommender systems need to perform a search for similar artifacts whenever the learner requests assistance. In the desirable case where the Machine Teaching system can draw upon a large database of artifacts, this search considerably limits the real time applicability of the system, violating requirement R2. We therefore exclude memory based recommender systems from further analysis.

As stated in R1, a Machine Teaching System is frequently faced with new artifacts. These artifacts have only very few values for their structural elements and therefore provide little input to the recommender system to build its recommendation upon. It is thus desirable to use a hybrid recommender system which is capable of using additional features of the artifact and the structural element to enhance its prediction accuracy.

Remaining Research Questions

However, to the best of our knowledge there is no model based hybrid recommender system that meets all of the requirements above. In Chapter 4, we present a novel recommender algorithm that meets these requirements. The application of this new algorithm to a Detailed Feedback Machine Teaching System for Software Engineers is subsequently presented in Chapter 6.

2.6 Conclusion

In this chapter, we introduced Machine Teaching as a new approach to Technology Enhanced Learning in which the knowledge formalization and externalization required in traditional Technology Enhanced Learning approaches is replaced with machine learning: A machine learning model is built from observation of experts which is henceforth used to provide feedback to learners.

An analysis of the major components of a hypothetical Machine Teaching system allowed us to determine a critical set of components needed to build such a system. This critical set of components and therefore the feasibility of a Machine Teaching is studied in the remainder of this thesis from a technical perspective.

The remainder of this chapter presented two sub-categories of Machine Teaching systems: Those providing a grade-like rating to the learner and those aiming at giving suggestions regarding the learner activity. The reason for this distinction lies not only in the task but also, as we have shown, in the available machine learning techniques: General feedback can be provided by supervised machine learning techniques, given suitable features of the artifact. Detailed feedback, however, in principle requires specialized machine learning models for each Machine Teaching task. But as we have shown, a large subset of these tasks can be tackled using enhanced Recommender Systems technology.

Next steps in this thesis: The remainder of this thesis is structured based upon this observation: In Chapter 3, we present a application of supervised machine learning to the Machine Teaching task of rating web forum posts. Based upon the encouraging results on this task, the Chapters 4 and 5 introduce a recommender systems model and algorithm capable of meeting the Machine Teaching requirements outlined above. This model and algorithm are then evaluated in the Machine Teaching task of providing software engineers with detailed feedback in Chapter 6.

3 General Feedback Machine Teaching for Web Forum Authors

Contents

3.1 Introduction

In this chapter, the computational elements of a General Feedback Machine Teaching approach are implemented and evaluated. The goal of a General Feedback Machine Teaching system is to provide the learner with "grade-like" ratings of her activity. As introduced earlier, we focus on observing the learner and the experts alike through their artifacts.

From a machine learning point of view, General Feedback Machine Teaching is an application of the supervised learning techniques, as discussed in Section 2.4. In the same section, we also identified feature engineering as the main challenge in building a General Feedback Machine Teaching system.

The remainder of this Chapter investigates the *feasibility* of a General Feedback Machine Teaching system based on an example. The goal is not to present the best possible solution to the machine learning problems involved, but to show that building such a system is indeed feasible. The chosen example domain, namely rating web forum posts, is introduced in Section 3.1.1. As there is no directly related or more so prior work, Section 3.2 presents prior research on useful features for this domain. Section 3.4 describes the evaluation procedure and Section 3.5 presents the evaluation results on a real data set.

3.1.1 Example domain

The example chosen for this evaluation is that of providing ratings of web forum posts. The goal of a General Feedback Machine Teaching system in this example can be phrased as:

> Given a set of human-rated forum posts, a machine model is learned that is capable of predicting a rating for a yet unseen post.

We will now address the question why this is an interesting domain to apply Machine Teaching to. We will do so in three steps: First, we will introduce the example domain in greater detail to motivate it. Second, we will discuss the kind of knowledge involved to motivate the inappropriateness of current technology enhanced learning systems. And lastly, we will present arguments why Machine Teaching should be applicable to this task

Challenges in Teaching to Write for Web Forums

However, the posting habits between different web forums vary greatly. Thus, the knowledge needed to write posts that are regarded highly by the community of a given forum cannot be (easily) transferred to another forum and thereby community. Thus, the task of teaching this knowledge through current technology enhanced learning is hard for the reasons discussed in Chapter 1.

Requirements for Machine Teaching to Write for Web Forums

In order to apply Machine Teaching, we need reliable data consisting of human-rated web forum posts, a machine learning technique to learn from this data and feature extraction procedures to make the data available to the machine learning algorithm. We will now briefly show that each of these requirements can be met in this domain:

Data Availability: Many websites such as Google Groups[1], Yahoo! Groups[2] and Nabble[3] offer their users the possibility to *rate* forum posts in order to filter them. These ratings can be exploited by a Machine Teaching system.

Machine Learning Technique: As introduced earlier, we are facing a supervised machine learning problem. As the field is well researched, the choice of a concrete technique from this realm largely depends upon the type of rating in the data and sought from the Machine Teaching system.

Feature Engineering: Text has long been an application domain for machine learning, e.g. for SPAM filters, automatic text categorization or information retrieval systems. Additionally, the natural language processing community has come up with a countless amount of feature extraction procedures. Thus, we can tap into this resource during feature engineering for the Machine Teaching system.

We can thus conclude the description of the example, Machine Teaching to write web forum posts, faithfully: It is a worthwhile example that poses serious challenges to current technology enhanced learning technology but where all the requirements for an application of Machine Teaching are met.

Below, we will first introduce related research in Section 3.2 before devoting Section 3.3 to feature engineering. The Sections 3.4 and 3.5 present the evaluation procedure and results obtained with these.

3.2 State of the Art

To the best of our knowledge, there is no approach described in the literature that offers exactly the functionality sought in this chapter. There is, however, influential work in the following areas that shall be discussed here:

- Approaches that have a similar goal to the one of Machine Teaching, but operate using fixed rules, namely Automatic essay scoring systems.
- Works that investigate the characteristics of the collaborative rating data.
- Research that provides pointers to features to be extracted for the task at hand.

[1]`http://groups.google.com`
[2]`http://groups.yahoo.com`
[3]`http://www.nabble.com`

3.2.1 Automatic Essay Scoring

The automatic grading of texts has been investigated for quite some time in the research area of automatic essay scoring systems. Numerous systems and approaches have been discussed in the literature, e.g. in [CB04] and [AB06]. Valenti et al. present an overview of the state-of-the-art in [VNC03].

These systems are based on the observation that there exists a fairly well defined notion of what a "good" essay is and how this manifests itself in detectable features of the essay. Additionally, essays tend to follow a very similar structure, at least those that are written to be graded by a teacher. The systems then detect these features of the essay as well as its structure using various heuristics and deduct a grade from these. So in essence, the grading function is *not learned from data*, but *designed* based on specifications. This obviously is impossible in the web scenario, where each topic, website sometimes each single web forum have distinct, implicit quality standards.

3.2.2 Data Characteristics

A question regarding the available data is whether the typically relatively few ratings from a subset of the community can be assumed to be indicative of the community's opinion at large. There is no data available on this subject for web forums, presumably due to the high cost of obtaining it.

However, other websites, most prominently the IT news site Slashdot[4], employ a sophisticated meta-rating scheme that adds a pee-review of the ratings to the system to ensure consistent ratings. Based on data from this peer review process from Slashdot, Lampe and Resnick showed in their empirical study [LR04] that the rating data is indeed consistent: Disputes of the original rating are very rare events.

Whether or not this result can be transferred to web forums is hard to say, as the audience of Slashdot, mostly IT students and professionals, is hardly representative for Web users at large. This result does, however, show that consistent ratings are attainable. And as we will see later, we are able to predict the rating quite well which further substantiates the conclusion that the ratings of posts found in web forums are consistent.

3.2.3 Feature Inspirations

A track of research studies the behavior of students in web forums in conjunction with their academic success. In [KSF$^+$06] it was found that there is a strong relation between student's posting habits and their final grade even to the point that the grade can be *predicted* from the posting habits. This supports our claim that writing well in forums is an worthwhile skill to learn. The main features used in the grade prediction are the number of posts, the average post length and the average number of replies to posts of the student. Thus, it is worthwhile to investigate the same features for the Machine Teaching task at hand.

[4] `http://www.slashdot.org`

Kim et al. present a method to predict product review helpfulness in [KPCP06]. In many web shops, customers can review products both in text as well as formally using a rating scale. Sites like Amazon also allow for the reviewing of these reviews to assure the quality of the reviews submitted to the site. Readers of a review are asked "Was this review helpful to you?" with the answer choices "Yes" or "No". These ratings are very explicit and very standardized, as each reviewer is asked to rate the review based on the same quality of the review, namely its helpfulness. Kim's system than makes use of the explicit content of the review in order to predict its helpfulness with great accuracy. However, such a system is not easily transferable to the domain at hand for a number of reasons. The ratings a Machine Teaching System operates on cannot be assumed to be about the same quality dimension, with helpfulness being merely one of the choices as opposed to being the sole dimension of quality. Additionally, the system presented in [KPCP06] gains much of its performance from the explicit content of a review, such as the numerical rating of the product discussed in the review. Clearly, such information cannot be assumed to be available in all Machine Teaching instances. The study [KPCP06] does, however, show that such structured data is very useful and should be used whenever possible.

Summary: The related work suggests that the task of rating texts is possible, as it has been done by human-designed systems before, albeit with an inherently limited scope when compared to the aim of the approach presented in this chapter. The rating data underlying a machine learning approach can be assumed to be consistent with the community, albeit the available data on that point is sparse. Lastly, we have briefly discussed other machine learning based approaches operating on similar data to provide us with feature ideas for the task at hand.

The next step in investigating the feasibility of a General Feedback Machine Teaching system in the text domain is to present the features developed for the system in Section 3.3 before presenting the evaluation procedure and results in the Sections 3.4 and 3.5

3.3 Feature Engineering

As mentioned earlier, feature engineering is the crucial step to the success of a supervised machine learning application: While we can resort to an off-the-shelve supervised machine learning algorithm, feature engineering needs to be done for each application domain separately.

In this section, the feature extraction procedures developed for the Machine Teaching system are described. As introduced in Section 2.4, the input data which consists of unstructured text needs to be converted into vectors. This process is commonly referred to as feature extraction. Designing and implementing features for this task is mainly a manual process, which is guided by prior work and experience and intuition. Thus, feature engineering adds a systematic bias to the machine learning process at large, but one that is believed to aid in the learning task. For the system at hand, feature extractors from five different classes have been built: *Surface, Lexical, Syntactic, Forum specific and Similarity features.* The features and their extraction procedure are now described in detail.

3.3.1 Surface Features

The first class of features deals with properties of the text that are extractable on the character level of the posts.

Length: It is hypothesized that the length of a post can have an influence on the quality of it according to community standards. Thus, this feature captures the number of tokens as reported by the tokenizer supplied by the Java SDK.

Question Frequency: The fraction of sentences ending with a question mark "?". Depending on the community, the presence or absence of questions and their frequency may be indicative of the perceived quality of the post.

Exclamation Frequency: The fraction of sentences ending with and exclamation mark "!". Frequent use of exclamation marks is often considered rude in web forums.

Capitalized Word Frequency: The fraction of words spelled all CAPITALIZED. Words spelled like this are commonly associated with shouting in the conversation and are thus indicative of rude behavior.

3.3.2 Lexical Features

This class of features is concerned with the actual wording of the posts.

Spelling Error Frequency: It is commonly agreed that texts with a high fraction of misspelled words are considered bad. Thus, this feature detects the percentage of words that are not spelled correctly. In the experiments, the Jazzy spell checking engine[5] was used together with an English dictionary, as only English texts were analyzed.

Swear Word Frequency: This feature extractor stems from the same line of thought as the extractors for the exclamation frequency and the capital word frequency: Rudeness in the text might indicate poor quality according to the community standards. Here, rudeness is detected rather directly by determining the percentage of words that are on a list of swear words. The list of swear words was compiled from public resources like WordNet and the Wikipedia. This swear word list contains more than eighty words like "asshole", but also common transcriptions like "f*ckin" which occur frequently in web forum posts.

[5]`http://jazzy.sourceforge.net`

3.3.3 Syntactic Features

This class for feature extractors is concerned with the syntactic level of the texts analyzed. To do so, the texts are annotated with part-of-speech tags as defined in the PENN Treebank tag set, see [MSM94].

To do so, the TreeTagger [Sch95] was used, parametrized by the parameter files for English texts supplied with it. The fraction of each part-of-speech is then stored as one dimension in the resulting feature vector.

3.3.4 Forum Specific Features

The texts analyzed here stem from web forums, a genre of text that exhibits certain features not present in other forms of text. The presence or absence of these specific features may have an influence on the quality of the posts as perceived by the fellow users of the same forum. The following features were extracted:

IsHTML: The users of a forum are usually offered some means to style their posts. In the case of the Nabble data used below, this was done using standard HTML markup. This feature thus encodes whether or not the author of the post made use of this offering.

IsMail: Nabble also bridges mailing lists into web forums and vice verse. This feature captures the origin of a specific post, namely whether it is originally an email. If not, the post has been entered through the web interface of Nabble.com.

Quote Fraction: When authoring a post, the user may choose to quote another post, e. g. to answer to a specific question raised in that other post. This is often considered good style. However, some posts quote much without an obvious benefit to the post. This feature thus captures the fraction of characters within quotes of a post to allow the machine learning model to capture this property.

Path and URL Counts: In forums where users help one another, a direct pointer to further information may be considered to be a good aspect of a post. In the experiments, two special kinds of pointers are considered: UNIX path names and URLs. Their number is counted and forms a feature in the feature vector.

3.3.5 Similarity features

Web forums are typically organized by topic. Posts which do not match the topic are called "off topic" and are usually considered to be bad posts. In order to capture the relatedness of a post to the topic it is posted in, the cosine between the word vector of the post and the word vector of the topic is used as an additional feature.

Note that this list of features is just an example of the feature engineering process required in the application of the Machine Teaching Process for general feedback. In every application, the set of features to extract needs to be re-evaluated.

Next steps: The remainder of this Chapter will focus on the evaluation of these features for the task of rating web forum posts, starting with describing the procedure for doing so in Section 3.4. The following section will then present results from that evaluation and present conclusions thereof.

3.4 Evaluation Procedure

In this section, we will describe the evaluation process used to evaluate the viability of a General Feedback Machine Teaching system for rating web forum post. We describe the evaluation process in several steps:

- The data used for the evaluation is described and descriptive statistics of it are discussed in Section 3.4.1. That section also goes into detail of the pre-processing and filtering of the data for the experiments reported in this chapter.
- The evaluation method is described in Section 3.4.2, including the cross-validation setup and the choice of the supervised machine learning algorithm and implementation thereof.

The results obtained using this experimental setup are presented in Section 3.5.

3.4.1 Data Set and Pre-processing

The data to evaluate the system was kindly provided by Nabble[6]. Nabble is a web site that hosts forums for a wide range of communities and also mirrors mailing list conversations onto web forums. Below both kinds of content will be referred to as web forums.

The web forums hosted by Nabble are placed in a hierarchy, organized by their topic. Each forum is placed into exactly one category within this hierarchy. The discussions in reach forum are further subdivided into *conversation threads*. All the posts in reply to the same post are part of one thread. As these posts may again attract replies, the forums forum a tree with posts and their respective answers as edges and leaves.

Posts at Nabble can be rated on a five star scale by the users, with five stars being the highest rating. Analysis of the data showed that most of the rated posts are within the "Software" category[7]. To analyze the influence of the topic and thus the community on the quality standards, the data was analyzed in three data sets:

ALL: All rated posts in the database. This is the broadest of all data sets.

SOFT: All rated posts of forums that are in the software category. These are posts that concern closely related.

[6]http://www.nabble.com
[7]http://www.nabble.com/Software-f94.html

Stars	Label	ALL		SOFT		MISC	
⋆	Poor	1928	45%	1251	63%	677	29%
⋆⋆	Below Avg.	120	3%	44	2%	76	3%
⋆⋆⋆	Average	185	4%	69	4%	116	5%
⋆⋆⋆⋆	Above Avg	326	8%	183	9%	143	6%
⋆⋆⋆⋆⋆	Excellent	1732	40%	421	21%	1311	56%

Table 3.1: Categories and their usage frequency at Nabble.

MISC: All posts that are in ALL, but not in SOFT. This data set is very diverse in topic, even more so than ALL, as half of ALL are posts from SOFT. Topics range from discussions amongst Wikipedia community members to discussions of motor bikes.

Table 3.1 shows the distribution of average ratings on the five star scale employed by Nabble. From this statistics, it becomes evident that users at Nabble prefer extreme ratings. Therefore, the task of predicting the post quality can be considered as a binary classification task. Posts with less than three stars are rated as "bad". Posts with more than three stars are "good".

Data Pre-processing

The goal of data preprocessing was to clean the data to a point where the label only contains variance stemming from the preceived quality of the forum posts. To do so, osts with an average rating of exactly 3 were removed, as they cannot be attributed to any of the binary classes. Manual analysis of posts with contradicting votes on the binary scale revealed that they were mostly spam, which was voted high for commercial interest by account associated with the poster and voted down for being spam. Thus, these posts were removed, too. We also filtered out the posts that did not contain any text, but only attachments like pictures and program files as the quality of these posts cannot be attributed to the textual content. Finally, we removed non-English posts using a simple heuristics: Posts that contained a certain percentage of words above a pre-defined threshold, which are non-English according to an English dictionary, were considered to be non-English.

The upper part of Table 3.2 shows how many posts were removed from the three data sets. Note that we did the filtering independently for each filter. Thus, posts that matched several filtering criteria contribute more than once to the statistics. The lower part of that table shows the distribution of good and bad posts after filtering.

3.4.2 Method

Using the feature extractors described above, we compiled a feature vector for each post. Feature values that were not normalized by definition were scaled to the range

	ALL		**SOFT**		**MISC**	
Unfiltered Posts	4291		1968		2323	
All ratings three stars	135	3%	61	3%	74	3%
Contradictory ratings	70	2%	14	1%	56	2%
No text	56	1%	30	2%	26	1%
Non-English	668	15%	361	18%	307	13%
Remaining	3418	80%	1532	78%	1886	81%
Good Posts	1829	54%	947	62%	1244	66%
Bad Posts	1589	46%	585	38%	642	34%

Table 3.2: Number of posts filtered out in the different data sets.

$[0.0, \ldots, 1.0]$. To classify the posts, we use support vector machines. In particular, we used a SVM with a Gaussian kernel as implemented in the LibSVM module in the YALE toolkit [MWK$^+$06] in all experiments.

We perform stratified ten-fold cross validation for performance evaluation. The data is split into ten sub sets where each exhibits the same class distribution as the full data set. The system is then trained on nine of these sub sets and evaluated on the last. We report the mean results of all ten evaluations.

Several randomly chosen experiments were repeated using the leave one out evaluation scheme. There, the system is trained on all but one post and evaluated on the remaining one. The average over all runs over all data points is returned as the performance metric. These experiments yielded comparable results to the ones obtained using cross validation. Thus, we only report the latter.

3.5 Evaluation Results and Discussion

In this section, we present and discuss the evaluation results obtained through the procedure introduced in the previous section. We first present the empirical results in Section 3.5.1. In Section 3.5.2, we discuss the errors the system makes to allow us to draw the conclusions presented in Section 3.6.

3.5.1 Results

Table 3.3 shows the average cross validation accuracy for all combinations of feature and data sets when compared to a baseline system. Note that we did not perform parameter tuning for the different feature sets but kept the parameters fixed[8] and therefore minor performance gains may still be possible.

[8]The parameters where fixed to $\lambda = 0.1$ and $\sigma = 0.1$.

Baseline: A baseline system is a simple system that serves as a plausibility check of the results obtained. In our case, we employ a majority-class classifier as the baseline: Such a classifier always predicts the class which had the majority of instances in the training data. Example: If 70 % of the posts in the training data are labeled as good posts, the majority class classifier labels all unseen posts as good. Clearly, any more sophisticated machine learning system should outperform this baseline.

As shown in Table 3.3 , all results but one (SIM/ALL) are equal to or better than the baseline. The usage of all features results in the best or close to best performance for all data sets. The results on the MISC data set are only slightly better than the baseline. The gains on the SOFT and ALL data sets over the baseline are significant. Naively, one may think that the performance on the ALL data set is the average between the performance on MISC and SOFT, as both form approximately one half of the data in ALL. The results are different, and the performance on ALL is comparable to the performance on SOFT. Thus, the system is able to learn how to classify posts in MISC from posts in SOFT. It is thus plausible to assume that the rating structure in some posts of the MISC data set is very close to the SOFT data set, while the overall rating structure is too diverse to be captured correctly by the system.

The difference in rating structure also shows in the analysis of the best performing feature categories, which are different for each data set. For MISC, the surface features perform best. For SOFT, the forum specific features work best, when only one feature category is used.

It is useful to have a look at the performance of all other feature categories, when the single best one is not present to assess the influence of the best feature category on the overall performance. For MISC, this leads to a performance on the baseline level. For SOFT, the drop in performance is much smaller, yet still measurable. For ALL, the effects are the smallest, being almost zero for the removal of the lexical features.

To evaluate the features in more detail, additional experiments were performed on the SOFT data set with only the features from the best performing category, namely the Forum specific features. Table 3.4 shows that IsMail and Quote Fraction are the dominant features. This is noteworthy, as those features are not based on the domain of discussion.

3.5.2 Performance Analysis

In addition to the numerical evaluation presented above, we will now present a more detailed analysis of the performance of the system.

The first step to do so is to investigate more closely what kind of errors contributed to the average accuracy presented above through confusion tables. In a second step, we present a qualitative evaluation of frequent mistakes of the system. Besides pointing out open issues in the currently implemented system, the latter allows us to draw conclusions regarding the inherent limits of the approach.

SUF	LEX	SYN	FOR	SIM	ALL	SOFT	MISC
√	√	√	√	√	77.53% (1.45)	89.10% (1.44)	71.95% (1.09)
√	–	–	–	–	64.72% (1.21)	61.82% (1.00)	**71.31%** (1.08)
–	√	–	–	–	**74.08%** (1.38)	71.82% (1.16)	65.96% (1.00)
–	–	√	–	–	69.18% (1.29)	82.64% (1.34)	66.70% (1.01)
–	–	–	√	–	**74.08%** (1.38)	**85.05%** (1.36)	65.96% (1.00)
–	–	–	–	√	46.49% (0.87)	62.01% (1.00)	65.96% (1.00)
–	√	√	√	√	75.92% (1.42)	89.10% (1.44)	66.60% (1.01)
√	–	√	√	√	**77.39%** (1.45)	**89.36%** (1.46)	72.00% (1.09)
√	√	–	√	√	76.27% (1.43)	85.03% (1.38)	70.03% (1.06)
√	√	√	–	√	72.82% (1.36)	82.90% (1.34)	71.74% (1.08)
√	√	√	√	–	76.83% (1.44)	88.97% (1.44)	**72.43%** (1.10)
Baseline					53.51% (1.00)	61.82% (1.00)	65.96% (1.00)

Table 3.3: Accuracy with different feature sets. SUF: Surface, LEX: Lexical, SYN: Syntax, FOR: Forum specific, SIM: similarity. The *baseline* results from a majority class classifier.

Confusion Tables

Confusion tables present the performance of a system in more detail by showing the number of all four performance relevant cases of a binary classifier:

True Positives: These are posts that are labeled as good by the users and are predicted to be good posts by the system.

True Negatives: Posts which are labeled and correctly predicted to be bad.

False Positives: The system incorrectly predicted a good label for these posts, while the users labeled it as bad.

False Negatives: In the last case, the system predicted a bad label while the users actually labeled the post as good.

Depending on the system evaluated, a balanced or unbalanced rate of false positives and false negatives may be desired. Example: In the case of an email spam filter, one typically wants to minimize the number of emails that are falsely labeled as spam by the system, even if that means to compromise the rate of spam caught by the filter.

In Machine Teaching, a balanced performance is more desirable to allow the learner to draw her own conclusions from the rating presented to her.

Tables 3.5, 3.6 and 3.7 contain the confusion tables for the system using all features on the three different data sets. The system produces approximately an equal amount of false positives and false negatives on the ALL and SOFT data sets. However, it has a tendency towards false positives on the MISC data set. This indicates that systems

ISM	**ISH**	**QFR**	**URL**	**PAC**	**Avg. accuracy**
√	√	√	√	√	*85.05%*
√	–	–	–	–	73.30%
–	√	–	–	–	61.82%
–	–	√	–	–	73.76%
–	–	–	√	–	61.29%
–	–	–	–	√	61.82%
–	√	√	√	√	74.41%
√	–	√	√	√	*85.05%*
√	√	–	√	√	73.30%
√	√	√	–	√	*85.05%*
√	√	√	√	–	*85.05%*
√	–	√	–	–	84.99%
√	√	√	–	–	*85.05%*

Table 3.4: Accuracy with different forum specific features. ISM: IsMail, ISH: IsHTML, QFR: QuoteFraction, URL: URLCount, PAC: PathCount.

should be trained separately for different topics of discussion and therefore user communities.

Qualitative Analysis of important System Errors

Below, we will give descriptions of common errors of our system as well as some examples from the data. We will also provide conclusions on how to improve the current system to overcome the errors where possible and indicate errors which exemplify the inherent limits of the approach.

Ratings based on domain knowledge: The following post from the SOFT data set shows no apparent reason to be rated badly. The human rating of this post seems to be dependent on deep domain knowledge, which cannot be represented in the Machine Teaching System easily. Thus, these posts are part of the inherit limits of the approach.

	true good	true bad	sum
pred. good	1517	456	1973
pred. bad	312	1133	1445
sum	1829	1589	3418

Table 3.5: Confusion matrix for the system using all features on the ALL data sets.

	true good	true bad	sum
pred. good	490	72	562
pred. bad	95	875	970
sum	585	947	1532

Table 3.6: Confusion matrix for the system using all features on the SOFT data sets.

	true good	true bad	sum
pred. good	1231	516	1747
pred. bad	13	126	139
sum	1244	642	1886

Table 3.7: Confusion matrix for the system using all features on the MISC data sets.

Example 1.

```
> Thank You for the fast response, but I'm not
> sure if I understand you right.  INTERRUPTs can
> be interrupted (by other interrupts or signals) and
> SIGNALS not.

   Yup.  And I responded faster than my brain could shift gears
and got my INTERRUPT and SIGNAL crossed.

   > All my questions still remain!

   Believe J"org addressed everything in full.  That the
compiler simply can't know that other routines have left
__zero_reg__ alone and the compiler expects to find zero there.
As for SREG, no telling what another routine was doing with the
status bits so it too has to be saved and restored before any
of its contents possibly get modified.  CISC CPUs do this for
you when stacking the IRQ, and on RTI.
```

Automatically generated mails: Sometimes, automatically generated mails like error messages end up on the mailing lists mirrored by Nabble. These mails can be written very nicely and are thus misclassified by the system as good posts, while they are bad posts from the point of view of the users. These errors are conceptually easy to avoid by pre-processing. In fact, these posts should never be used for a Machine Teaching system, as there is little to learn from them.

Non-textual content: Especially the SOFT data set contains posts that mainly consist of non-textual parts like source code, digital signatures and log messages from programs. This content confuses our system to miss-classify these posts as bad posts even though the sheer presence of these parts may be very useful to the reader.

To overcome this problem, the non-textual parts need to be marked. They can then be ignored in the quality assessment of the textual content. Additionally, the presence and the amount of non-textual content can be used as an additional feature. A Detailed Feedback Machine Teaching system could then even suggest to the learner to attach supporting material to make their post more valuable to the readers.

Very short posts: Posts which contain only a few words show up as false positives and false negatives equally, as for example a simple “yes” from the grand master of a certain field might be regarded as a very good post, while a short insult in another forum might be regarded as a very bad post. For Machine Teaching, it thus seems advisable not to rate very short posts at all, as the system is very likely to rate the post wrong and thus to confuse the learner.

Opinion based ratings: Some ratings do not rate the *quality* of a post, but the *expressed opinion*. In these cases, the rating is an alternative to posting a reply to the message saying “I do not agree with you”.

Take for example the following post which is part of a discussion amongst Wikipedia community members from the MISC data which has been misclassified as a bad post:

Example 2.
```
> But you would impose US law even in a country where
> smoking weed is legal
Given that most of our users and most significant press
coverage is American, yes.  That is why I drew the line there.
Yes, I know it isn't perfect.  But it's better than anything
else I've seen.
```

Such posts form a hard challenge for automatic systems. However, they may also form the upper bound for this task: Humans are unlikely to predict these ratings correctly without additional knowledge about the rater, either.

3.6 Conclusion

We studied the machine learning aspects of a General Feedback Machine Teaching system for web forum post writing in this chapter. The goal of this study was to answer the question whether suitable features can be found that allow a supervised machine learning method to rate the forum posts well.

The quantitative evaluation presented in this chapter suggests that such features have indeed been found as the system achieves an average accuracy of up to 90 %. The qualitative study also substantiates the claim that proper feature engineering is the key factor for the predictive performance of the system.

The qualitative analysis of the results also suggest important considerations and limits for Machine Teaching:

- Not all ratings given by humans can be predicted by a machine. The examples found in the qualitative study are ratings that express opinions and those which require deep domain knowledge to be given.
- Data pre-processing for Machine Teaching should be done in addition to the pre-processing done to allow the machine learning system to perform well. The study in this chapter follows machine learning best practices when it comes to pre-processing such as filtering ambiguously rated posts. The quantitative analysis of the system's performance however revealed additional need for pre-processing: The data contained artifacts such as automatically generated content that should not be part of the training data for a machine teaching system.
- The accuracy of the system on short posts is to low, even if above random, to justify presenting the predicted ratings to the learner.

Besides the hard limits in machine predictability of human behavior inherent to the approach, all of these considerations can be addressed through engineering.

Thus, and given the good overall performance of the system, it is safe to conclude that the state-of-the-art machine learning techniques as well as the best practices in the field can be applied to build General Feedback Machine Teaching systems.

Next steps in this thesis: Given this conclusion, the remainder of this thesis departs from General Feedback Machine Teaching and focuses on the task of Detailed Feedback Machine Teaching. As this task is more ambitious from a machine learning perspective, Chapter 4 presents an algorithm that forms the foundation for addressing this task. Chapter 5 leaves the field of Machine Teaching in order to evaluate that algorithm on well established Recommender Systems tasks. This allows us to compare the algorithm performance to that of the state-of-the-art in that field. In Chapter 6, a similar study to the one in the present chapter is described, albeit for the Detailed Feedback Machine Teaching case.

4 Generalized Matrix Factorization

Contents

4.1 Introduction

This chapter introduces a novel model and algorithm for the problem of matrix factorization. Before doing so, it is shown how both the Recommender System problem and many Machine Teaching settings can be rephrased into matrix factorization instances.

We will first show that the available data in both Recommender Systems and Machine Teaching can be thought of as a sparse matrix Y for which a dense approximation is sought. The entries at the formerly sparse points of this approximation constitute the prediction. This view on the data is visualized in Figure 4.1 and described in the following paragraphs:

Machine Teaching with Matrix Factorization: In a Machine Teaching setting, each row in Y corresponds to one artifact. The columns of Y contain the values of the structure elements of this artifacts. This matrix is typically sparse, as not all artifacts exhibit all structure elements. To give detailed feedback for the learner, the Machine Teaching System needs to deduct a model from the data which can compute a dense matrix F which contains predictions for the sparse points in Y. In contrast to the recommender system setting, the Machine Teaching system may not only make use of the predictions for the sparse entries of Y but also of the discrepancy between the known entries in Y and those in the model F, e. g. to correct the learner.

Recommender Systems with Matrix Factorization: In Recommender Systems, Y has as many rows as there are users and as many columns as there are item. The entries $Y_{i,j}$ of this matrix are the label given to item j by user i. The type of these entries depends on the kind of label available in the specific data set. Note that this matrix is very *sparse*. In a typical movie recommender scenario, there are in the order of twenty thousand movies and thus columns in Y. Not many users have watched and rated all these movies. The goal for the recommender system based on this sparse matrix Y is to find a *dense* matrix F of equal dimensions that contains predictions for the user-item pairs not present in Y.

The algorithm described in this chapter is applicable to both scenarios: Machine Teaching and Recommender Systems. This is due to the fact that it operates on the rather abstract level of matrices as opposed to their real world correspondences of artifacts, structural elements, users and items.

Honoring this aspect and to facilitate a broad view on the algorithm, the remainder of this chapter is written in an application-agnostic manner. We will give examples from both the Machine Teaching and the Recommender Systems perspective where appropriate.

In this application-agnostic notation, the goal of the algorithm can be phrased as: Given a potentially sparse input matrix $Y \in \mathbb{T}^{r \times c}$ find a dense matrix $F \in \mathbb{T}^{r \times c}$ that explains Y well and predicts entries for the sparse elements in Y.

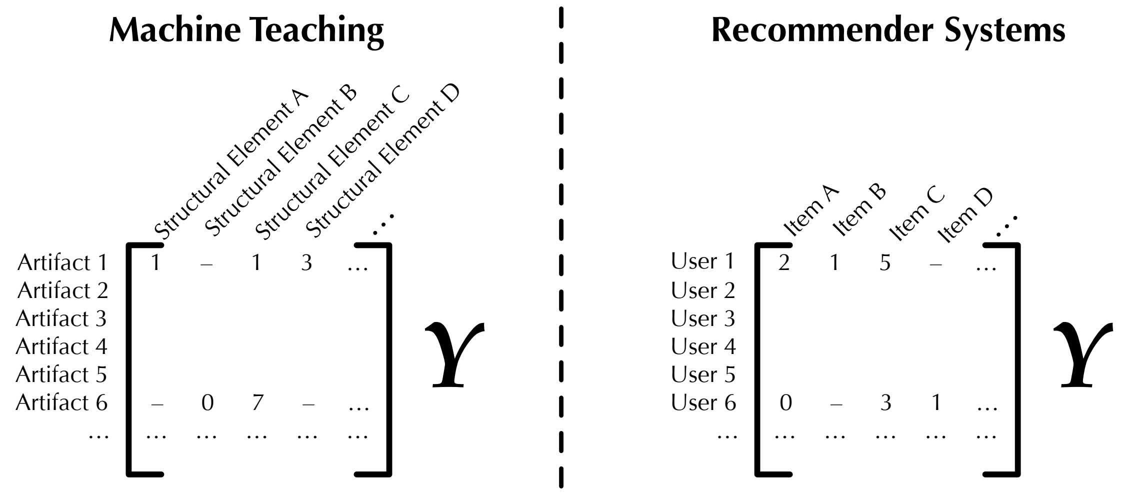


Figure 4.1: Modeling the available data in a Detailed Feedback Machine Teaching System (left) and in a Recommender System (right) as a sparse matrix Y. The symbol "–" indicates a sparse point (missing value)

4.2 State of the Art

The development of factor models and in particular matrix factorization methods are an active field of research. Factor models have shown remarkable performance in many instances, especially in the context of the Netflix prize challenge where all of the top contenders use factor model based approaches.

In matrix factorization methods, the data is assumed to be contained in a sparse matrix $Y \in T^{r \times c}$ ($Y_{i,j}$ in Recommender Systems typically indicates the rating of item j by user i). The basic idea of matrix factorization is to fit this matrix Y with a low rank approximation F. Here, F is computed as $F = RC^\top$ where $R \in \mathbb{R}^{r \times d}$ and $C \in \mathbb{R}^{c \times d}$. More specifically, the goal is to find an approximation that minimizes a loss measure such as the sum of the squared distances between the known entries in Y and their predictions in F.

Below, we introduce major means of finding the approximation F, starting with Singular Value Decomposition before discussing the Regularized Matrix Factorization technique that the algorithm presented here is built upon. The last part of this section will be devoted to the related work on seemingly binary input matrices Y.

Singular Value Decomposition One way of doing this is to compute a Singular Value Decomposition of Y and to use only a small number of the vectors obtained by this procedure. In the information retrieval literature, this numerical operation is commonly referred to as Latent Semantic Indexing as introduced in [Hul94].

Note, however, that this method does not do justice to the way Y was formed as Y is sparse. This fact is ignored by the SVD and depending on the specific implementation, the sparse elements in Y are assumed to be 0 or are replaced with averages in order to compute the SVD. Both approaches potentially misinterpret the data. If e. g. a user did not rate an item this may correspond to a variety of reasons that cannot be captured by these assumptions. Many rating schemes assume lower ratings to be worse, and filling in the sparse elements with ratings of 0 assumes that the user did not like these movies. The filling with (per user) averages fares slightly better, but that assumption is questionable in many instances, too.

The filling poses even more of a problem in Machine Teaching where each value of a structure element may have an impact on the artifact quality. Both 0 and the the average value of the remaining elements are likely completely unrelated to a reasonable value to be filled in. Even worse, this approach ignores the fact that the sparse elements in Y are the requested *output* of the algorithm and should thus not be used as *input*.

Regularized Matrix Factorization In [SJ03], an alternative approach is suggested which aims to find a factorization of Y in two matrices $R \in \mathbb{R}^{r \times d}$ and $C \in \mathbb{R}^{c \times d}$ such that $F = RC^\top$ with the goal to approximate the *observed* entries in Y rather than approximating all entries at the same time. This approach has been shown to work well on e. g. rating data from users on movies such as in the Netflix price competition.

Minimizing the rank of F is intractable in practice for all but the smallest problems. Instead. the question of capacity control of the resulting predictor has been addressed in [RS05] and [SRJ05] by means of matrix norms on F. It has been shown in [SS05] that the Frobenius norm on R and C is a proper norm on F and can thus be used for capacity control. In addition, they proposed a multi-class version of the hinge loss function that together with the Frobenius norm induces a large margin solution. The term Maximum Margin Matrix Factorization (MMMF) has thus been coined for this approach in [SRJ05]. Similar ideas based on matrix factorization have also been proposed in [TPNT07] and in [SM08, BKV07] which mainly focus on improving performance on the Netflix data set. In [SG08] an integration of these different approaches into one model is presented. Note that all these methods are colloquially referred to as "SVD" or "SVD-style", even though they do not use a SVD in the strict sense.

None of these approaches address the problem of a prediction that needs to be consistent per-row as required in the Machine Teaching setting.

Binary data In many instances, the data is seemingly binary, for example when building a Machine Teaching System for software engineering:

Example 5. *We will later model code as a matrix of caller-callee relations. An entry of $Y_{i,j} = 1$ in this matrix indicates a call of caller i to callee j. However, the opposite entry cannot be attributed to a single cause. The absence of a call can e. g. be attributed to a conscious decision of the programmer or a bug. Any of these cases will be encoded as a sparse and thus missing entry in Y, as the true reason typically is unknown.*

Note that the same effect also occurs in commercial Recommender Systems that operate on shopping basket data: There, it is only known for sure whether a customer bought an item. The information that a customer did not buy an item can be attributed to a variety of reasons, including being unaware of the item. In that case, a recommendation should be considered for the item. Thus, it is not advisable to equate the absence of a buy with a dismissal of the item by the customer.

We refer to seemingly binary data like this as *dyadic interaction data*. It can be represented as a sparse matrix where $Y_{i,j} = 1$ indicates that an interaction between row i and column j was recorded. In Machine Teaching such an entry indicates that the artifact i has the structural element j. In a Recommender System it indicates that user i interacted with item j, e. g. by renting the movie j.

Traditionally, rule and memory based systems have been applied in these situations (see e. g. [AT05]). As discussed in Section 2.5, one of the shortcomings of memory based systems is that the prediction time scales super linearly with the number of training samples. This poses a serious restriction, as these systems cannot be used interactively any more when the number of available training data rises which poses a serious limitation on the applicability of these systems to Machine Teaching scenarios where instant feedback is required.

Several approaches have been introduced under the term "binary matrix factorization", see e. g. [ZDLZ07]. In these systems, R and C are assumed to be binary, too. However, this drastically limits the applicability, as finding these matrices then amounts to solving combinatorial optimization problems. Other approaches based on binary latent factors [MGNR06] and Gaussian processes [YC07] extract binary factors as well which induce constrains that would not be particularly useful in the recommender setting we are considering and provide little control over the treatment of unlabeled values or negative examples in the data.

Conclusion: The state of the art provides a solid theoretical and empirical base for matrix factorization approaches. However, most systems do not make use of features, even though their addition is rather straight forward as we will discuss below. Using features is crucial for a Machine Teaching System and helps considerably in overcoming the new user and new item problems in Recommender Systems.

None of the approaches can deal with per-row predictions as they arise in Machine Teaching and ranking prediction in Recommender Systems. Binary data provides a formidable and essentially unsolved challenge.

4.3 Regularized Matrix Factorization

In this section, the notion of Regularized Matrix Factorization is presented in greater detail to allow us to use it as a framework in which to present our algorithm.

As noted earlier, the algorithm presented here is based on the idea of *Factor Models*: Each known entry $Y_{i,j}$ is hypothesized to be explainable by a linear combination of d row factors $R_i \in \mathbb{R}^d$ and d columns factors $C_j \in \mathbb{R}^d$:

$$F_{i,j} := \langle R_i, C_J \rangle \tag{4.1}$$

Symbol	Type	Description
Y	$\mathbb{T}^{r\times c}$	The sparse input matrix
F	$\mathbb{T}^{r\times c}$	The dense prediction matrix
R	$\mathbb{R}^{r\times d}$	The dense matrix of row factors
C	$\mathbb{R}^{c\times d}$	The dense matrix of column factors
r	$\mathbb{N}$	The number of rows in Y
c	$\mathbb{N}$	The number of columns in Y
n	$\mathbb{N}$	The number of entries in Y
$\mathbb{T}$	–	The entry type in Y and F
$L(F,Y)$	$\mathbb{T}^{r\times c}\times\mathbb{T}^{r\times c}\to\mathbb{R}$	The loss function
$\Omega(F)$	$\mathbb{T}^{r\times c}\to\mathbb{R}$	The regularizer
$\lambda_r,$	$\mathbb{R}$	The regularization parameter for R
λ_c	$\mathbb{R}$	The regularization parameter for C
S	$0,1^{r\times c}$	$S_{i,j}=1$ for values present in Y and 0 otherwise.
$g(y)$	$\mathbb{R}$	The weight associated with the label y in the weighted loss functions.
y,f	$\mathbb{T}^{\tilde{c}}$	Dense versions of a row in Y and F where the sparse elements of Y have been omitted.

Table 4.1: Symbols used

Matrix Factorization models built upon this idea by noting that in effect, F is constructed as the multiplication of two matrices $R\in\mathbb{R}^{r\times d}$ and $C\in\mathbb{R}^{c\times d}$:

$$F := RC^\top \tag{4.2}$$

While this model allows to compute the prediction matrix F, it does not indicate how to compute the prediction matrix F from the input data Y. A very obvious requirement for F is that it should be "close" to Y. In this thesis as well as in the literature, the term *loss function* is used for the measure of closeness. This measure is denoted as $L(F,Y):\mathbb{T}^{r\times c}\times\mathbb{T}^{r\times c}\to\mathbb{R}$ in formulas. The value of it will be called *loss value* or loss. Different choices of $L(F,Y)$ facilitate different goals in the prediction. In Section 4.4, several possible loss functions will be discussed.

This notation enables a more formal definition of the goal: The prediction F should be the one with minimal loss:

$$F := \underset{\hat{F}}{\operatorname{argmin}}\, L\left(\hat{F},Y\right) \tag{4.3}$$

As per the prediction rule (4.2), we can rephrase this objective function as:

$$F := \underset{\hat{F}=\hat{R},\hat{C}}{\operatorname{argmin}}\left(L\left(\hat{R}\hat{C}',Y\right)\right) \tag{4.4}$$

In other words, we are searching for the model consisting of the matrices R and C that approximates the known entries in Y best. However, only minimizing the loss will yield poor performance due to the well known tendency of *overfitting*: If d is assumed to be large enough, it is probable that we can find R and C such that the prediction F perfectly matches the input data Y. However, such a predictor is known to generalize badly, as it performs badly on unseen data.

Machine Learning theory shows that limiting the *capacity* of the prediction function overcomes this problem. In intuitive terms, limiting the capacity of the learning machine ensures that the learned model is *simple* and explains the known data well. This follows the intuition that the most simple model is the one that captures most of the structure of the problem and thus generalizes well to future, yet unseen data points.

Following this argument, the objective function is extended by a *regularizer* $\Omega(F)$. Additionally, a regularization parameter λ is introduced to control the trade-off between model complexity and loss:

$$F := \underset{\hat{F}}{\operatorname{argmin}} \left(L\left(\hat{F}, Y\right) + \lambda\Omega\left(\hat{F}\right)\right) \tag{4.5}$$

As above, this objective function can be reformulated in terms of the factor matrices R and C:

$$F := \underset{\hat{F}=\hat{R},\hat{C}}{\operatorname{argmin}} \left(L\left(\hat{R}\hat{C}', Y\right) + \Omega\left(\hat{R}\hat{C}'\right)\right) \tag{4.6}$$

As with the loss, many different choices of the regularizer Ω are possible. The algorithm presented in this thesis follows the *Maximum Margin Matrix Factorization (MMMF)* approach as introduced in [SRJ05]. In MMMF, the L_2 or *Frobenius* norms of R and C are used as the regularizer:

$$||X||_F = \sqrt{\sum_{i,j} X_{i,j}^2} \tag{4.7}$$

This leads to an optimization problem similar to the margin maximization interpretation of support vector machines:

$$F := \underset{\hat{F}=\hat{R},\hat{C}}{\operatorname{argmin}} L(\hat{R}\hat{C}', Y) + \lambda_r||\hat{R}||_F^2 + \lambda_c||\hat{C}||_F^2 \tag{4.8}$$

Here λ_r and λ_c are constants that model the trade-off between the model accuracy and its complexity.

The formulation in equation 4.8 is not only the base for the algorithm presented here but also found in similar forms in the literature. The remainder of this chapter will describe the unique contributions of the algorithm presented in this thesis:

The following sections will describe how to find C and R by means of optimization in Section 4.5 and the choice of loss function in Section 4.4. The main contribution there is to use a state-of-the-art recommender and to introduce row based loss functions. The net effect of these improvements is that now essentially all loss functions known in supervised machine learning are transferable to matrix factorization. Additionally, several extensions to the basic regularized matrix factorization model are introduced in Section 4.6.

4.4 Loss Functions

In the argumentation above, the loss function $L(F, Y)$ was treated as a black box. However, the choice of loss function is crucial to the overall performance of the algorithm. By choosing a loss function, one can adapt the model to different types $\mathbb{T}$ of entries in Y. Additionally, the choice of loss very directly influences what predictions to expect from the system.

Example 6. *If the loss function chosen is designed for binary entries in* Y*, that is* $\mathbb{T} = -1, +1$ *but the real data type is* $\mathbb{R}$*, the system will perform poorly.*

Above, $L(F, Y)$ is assumed to be defined on the *whole matrices* F and Y for notational brevity. However, the loss functions used in real applications decompose either per row or per element of Y and F. Examples of loss functions that decompose per element are cases where F is sought to represent every entry in Y as good as possible. The case where the loss decomposes per row is useful in cases where the prediction F should capture the *relations* between the entries of each row in Y as opposed to their absolute value.

This section will present loss functions of use to either the recommender systems or the Machine Teaching scenario or both. For reasons that will become apparent in the Optimization Section 4.5, convex loss functions or convex approximations thereof are in the focus of this section. The optimization algorithms introduced later also depend on the presence of a (sub-)gradient of the loss function with respect to the prediction F. Hence, each loss function is introduced together with its gradient in this section.

4.4.1 Element Based Loss Functions

Element based loss functions are used to compute matrix factorizations that recreate the entries in Y with great accuracy. Thus, they decompose per element:

$$L(F, Y) = \sum_{i}^{r} \sum_{j}^{c} l(F_{i,j}, Y_{i,j}) \tag{4.9}$$

While this formulation is straight forward, it ignores a crucial aspect: Y may be sparse and this definition of the element based loss function computes a loss value even for i, j where Y has no value. To be able to ignore these entries, the matrix $S \in \{0, 1\}^{r \times c}$ is introduced:

$$S_{i,j} := \begin{cases} 1 & \text{if i,j is present in Y} \\ 0 & \text{otherwise} \end{cases} \tag{4.10}$$

This facilitates the following definition of a element based loss function that ignores entries where Y does not contain a value:

$$L(F,Y) = \sum_{i}^{r}\sum_{j}^{c} S_{i,j} l(F_{i,j}, Y_{i,j}) \tag{4.11}$$

As a consequence of this decomposition, the gradient of $L(F,Y)$ with respect to F decomposes per element, too:

$$\partial_F L(F,Y)_{i,j} = S_{i,j} \partial_{F_{i,j}} l(F_{i,j}, Y_{i,j}) \tag{4.12}$$

Given both equation (4.11) and equation (4.12), it is sufficient to describe the actual loss functions in terms of $l(F_{i,j}, Y_{i,j})$ and $\partial_{F_{i,j}} l(F_{i,j}, Y_{i,j})$.

Squared Error

The squared error is computed as the squared distance between the true value $Y_{i,j}$ and the prediction: $F_{i,j}$

$$l_{squared}(F_{i,j}, Y_{i,j}) = \frac{1}{2}(F_{i,j} - Y_{i,j})^2 \tag{4.13}$$

Using this loss results in a regularized least squares regression model. In recommender systems, this loss is often used when the entries in Y are ratings. However, ratings are typically not given on a real valued scale but as a one out of n selection. In Machine Teaching, the squared error can be used when the value type of the structure elements is $\mathbb{R}$.

The gradient of the Squared Error with respect to $F_{i,j}$ can be derived as:

$$\partial_{F_{i,j}} l_{squared}(F_{i,j}, Y_{i,j}) = F_{i,j} - Y_{i,j} \tag{4.14}$$

ϵ-insensitive Regression

In many cases, a prediction $F_{i,j}$ does not need to be exact but instead needs only to be within ϵ of the exact value. This thought led to the introduction of the ϵ-insensitive loss function as presented in [VGS97]:

$$l_\epsilon(F_{i,j}, Y_{i,j}) = \max(0, |F_{i,j} - Y_{i,j}| - \epsilon) \tag{4.15}$$

Its gradient can be computed as:

$$\partial_{F_{i,j}} l_\epsilon(F_{i,j}, Y_{i,j}) = \begin{cases} 0 & |F_{i,j} - Y_{i,j}| \leq \epsilon \\ sign(F_{i,j} - Y_{i,j}) & else \end{cases} \tag{4.16}$$

Note that this formulation of the loss function is not smooth and that a smooth approximation thereof has been introduced in [DSSS05]. However, the bundle method optimizer used here can optimize non-smooth loss functions.

Hinge Loss

When dealing with binary data in Y, that is $\mathbb{T} = \{-1, +1\}$, the most natural loss to use is the so called zero-one loss:

$$l_{01}(F_{i,j}, Y_{i,j}) = \begin{cases} 1 & F_{i,j} \neq Y_{i,j} \\ 0 & else \end{cases} \tag{4.17}$$

However, this loss function obviously is not convex in $F_{i,j}$ as it is piecewise constant. Hence, a convex relaxation of this loss function, the so called Hinge Loss, is commonly used in machine learning:

$$l_{Hinge}(F_{i,j}, Y_{i,j}) = \max(0, 1 - F_{i,j}Y_{i,j}) \tag{4.18}$$

If the prediction is correct, that is $F_{i,j} = Y_{i,j}$, then the product $F_{i,j}Y_{i,j}$ is 1 and the loss value is zero. The same is true whenever the sign of $F_{i,j}$ and $Y_{i,j}$ are equal with the absolute value of $F_{i,j} \geq 1$, thanks to the *max*. The loss is effectively unbound for the case where the sign of $F_{i,j}$ and $Y_{i,j}$ are not equal. In order to keep the predictions $F_{i,j}$ within $\mathbb{T} = \{-1, +1\}$, a prediction rule is used in practice: Whenever $F_{i,j}$ is less than zero, the prediction is -1 and else 1.

The resulting function is convex in F and its gradient is computed as:

$$\partial_{F_{i,j}} l_{Hinge}(F_{i,j}, Y_{i,j}) = \begin{cases} 0 & F_{i,j}Y_{i,j} \geq 1 \\ 1 & otherwise \end{cases} \tag{4.19}$$

Weighted variant: In many real world scenarios, the data set is not balanced between the two classes. In a typical Recommender System, the number of items bought by a user is typically very small compared to the number of items she could buy. In Chapter 6, an example from the Machine Teaching scenario will be discussed in depth which exposes this characteristic.

In that respect, the matrix factorization problem arising has some similarities with the supervised machine learning from positive and unlabeled data settings addressed in [LL03] and [EN08].

In these cases, it is convenient to attach a weight g to the positive labels. This leads to the following loss function:

$$l_{WHinge}(F_{i,j}, Y_{i,j}) = g(Y_{i,j}) \ \max\left(0, 1 - F_{i,j}Y_{i,j}\right) \tag{4.20}$$

where

$$g(y) = \begin{cases} g^{+} & y > 0 \\ 1 & else \end{cases}$$

Inclusion of the weight into the gradient computations leads to:

$$\partial_{F_{i,j}} l_{WHinge}(F_{i,j}, Y_{i,j}) = \begin{cases} 0 & F_{i,j}Y_{i,j} \geq 1 \\ g(Y_{i,j}) & otherwise \end{cases} \tag{4.21}$$

In [PS09], a solution to a similar problem is proposed based in the work presented in [HKV08]. However, that solution is limited to the Least Squares loss function.

Logistic Regression

In many instances, one is interested in the probability of a 1 than in its prediction. For instance, a Machine Teaching system can only present a limited amount of suggestions to the learner. The predicted probability can be used to filter that list of predictions and also provide the learner with a sense of importance of the suggestions.

To predict this probability, one can resort to the logistic regression loss function as introduced in [CSS00]:

$$l_{logistic}(F_{i,j}, Y_{i,j}) = \log(1 + \exp(-F_{i,j}Y_{i,j})) \tag{4.22}$$

Its gradient is computed as:

$$\partial_{F_{i,j}} l_{logistic}(F_{i,j}, Y_{i,j}) = \frac{-Y_{i,j}}{1 + \exp(F_{i,j}Y_{i,j})} \tag{4.23}$$

Weighted variant: The same reasoning presented for the hinge loss can be applied to the Logistic Regression, too. The

$$l_{Wlogistic}(F_{i,j}, Y_{i,j}) = g(Y_{i,j}) \log(1 + \exp(-F_{i,j}Y_{i,j})) \tag{4.24}$$

Inclusion of the weight into the gradient computations leads to:

$$\partial_{F_{i,j}} l_{Wlogistic}(F_{i,j}, Y_{i,j}) = \frac{-g(Y_{i,j}) - Y_{i,j}}{1 + \exp(F_{i,j}Y_{i,j})} \tag{4.25}$$

4.4.2 Row Based Loss Functions

The loss functions introduced so far compute a loss value per element of the matrices Y and F. Optimizing the objective function (4.8) thus yields to predictions in F that are good *on average* over all elements. This does not ensure that the predictions are coherent as such, e. g. per row.

However, there are important use cases where per-row coherence is important in the predictions. Examples include the need to learn inherent per-artifact coherence requirements from the data in Machine Teaching. In recommender systems, the need for per-row losses stems from the observation that the output of the recommender system is often used to compute a ranked list of recommendations for each user.

While this ranking task can be understood as a rating task followed by sorting, it is wasteful to do so: Modeling power is invested in precision that ultimately does not matter. This yields sub-optimal solutions.

This section will thus introduce several per-row loss functions that can be defined as:

$$L(F,Y) = \sum_i^r l(F_{i*}, Y_{i*}) \tag{4.26}$$

Notation: Here, F_{i*} indicates row i of matrix F. Below, we will use the lower case characters f and y to address *dense* versions of rows in F and Y. The elements of Y_{i*} that are empty are omitted in both y and f.

Example 7. *Let*

$$\begin{aligned} Y_{i*} &= [1,-,3,-,2,-] \\ F_{i*} &= [1,2,3,2,2,4] \end{aligned}$$

Then y and f are be defined as:

$$\begin{aligned} y &= [1,3,2] \\ f &= [1,3,2] \end{aligned}$$

As the loss function is defined on corresponding rows per Equation (4.26), y and f shall refer to the same row of Y and F in the following description of the per-row loss functions such that the loss above can be phrased as:

$$L(F,Y) = \sum_{f,y} l(f,y) \tag{4.27}$$

This notation facilitates the discussion of the row based loss functions in a more concise and elegant form.

4.4.3 A faster Ordinal Regression Loss Function

In this section, we first introduce the Ordinal Regression Loss Function as well as the state-of-the-art algorithm for its computation. In the second part, we present a faster algorithm for its computation.

The most straight forward view on the ranking problem is that of pair-wise sorting: A prediction is correct if it sorts all pairs of entries in f in the same way as they are sorted in y. For each pair where f does not sort them in same way as y, a penalty is added to the loss term.

In a more formal way, this loss, which was described in [HGO00] can be described as follows: For a given pair of entries (u, v) we consider them to be ranked correctly whenever $y_u > y_v$ implies that also $f_u > f_v$. A loss of is incurred whenever this implication does not hold:

$$l_{ord}(f, y) = \sum_{y_u > y_v} C(y_u, y_v)\, \{f_u \leq f_v\} \tag{4.28}$$

Here $C(y_u, y_v)$ denotes the cost of confusing an entry of y_u with one of value y_v and $\{f_u \leq f_v\}$ is a comparator function:

$$\{f_u \leq f_v\} = \begin{cases} 1 & f_u \leq f_v \\ 0 & otherwise \end{cases} \tag{4.29}$$

This comparator function obviously is not convex. Thus, we apply the same soft margin trick as in the hinge loss function introduced in Section 4.4.1:

$$l_{ord}(f, y) = \sum_{y_u > y_v} C(y_u, y_v) \max(0, 1 - f_u + f_v). \tag{4.30}$$

If f_u is less or equal than f_v, the penalty will be at least 1. If f_u is bigger than f_v, the penalty will be at most 1 and it vanishes for values of f_u that are greater or equal than $f_v + 1$. Thus, the step in the original comparator $\{f_u \leq f_v\}$ is replaced by a linear increase in the loss value around the comparison boundary.

Finally, the value of the loss as defined now is also determined by the number of entries in y. This clearly is an undesirable property, as the loss value on different rows of the matrices Y and F is now no longer comparable. Thus, the loss function needs to be normalized by the number of terms in the sum.

Assume that y is of length m containing m_j entries of value j, that is $\sum_j m_j = m$. There are m^2 comparisons possible between m entries. However, comparisons between entries with equal value don't fulfill the comparison, hence we need to subtract these $\sum_j m_j^2$ comparisons. Finally, only half of the comparisons are true. Thus, the number of terms in the sum can be computed as:

$$s = \frac{1}{2}\left[m^2 - \sum_j m_j^2\right] \tag{4.31}$$

Plugging this constant s into the loss formulation in equation (4.30) yields the following normalized and convex differentiable ordinal regression loss function:

$$l_{ord}(f, y) = \frac{1}{s} \sum_{y_u > y_v} C(y_u, y_v) \max(0, 1 - f_u + f_v). \tag{4.32}$$

Computing the gradient: Each element of f appears as f_u and f_v in the loss function (4.32). In each case, the loss function is a linear function in this f_u or f_v. This observation allows us to compute the gradient of the loss function with respect to f in a straight forward fashion:

$$\left[\partial_f l_{ord}(f, y)\right]_i = \frac{1}{s}\left(\sum_{(y_u > y_i)\wedge(f_u \leq f_i)} C(y_u, y_i) - \sum_{(y_i > y_v)\wedge(f_i \leq f_v)} C(y_i, y_v)\right) \tag{4.33}$$

Runtime Performance: Computing the loss involves iterating over all pairs of elements in y and f. Thus, it is a computation of complexity $\mathcal{O}\left(m^2\right)$, with m being the length of y. The computation of the gradient, if naively implemented following Equation (4.33), would be a $\mathcal{O}\left(m^3\right)$ operation, as the computation iterates over all m^2 pairs for all m entries in f. However, both the computation of the loss and the gradient can be folded into one $\mathcal{O}\left(m^2\right)$ operation as shown in Algorithm 4.4.3.

Algorithm 1 Computation of the ordinal regression loss $l_{ord}(f, y)$ and its gradient $\partial_f l_{ord}(f, y)$ in $\mathcal{O}\left(m^2\right)$ time.

```
**input** Vectors $f$ and $y$ score matrix $C$
**output** Loss $l := l(f, y)$ and gradient $g := \partial_f l(f, y)$
$m = length(y)$
$m_j$ = Number of elements of value $j$ in $y$
$n = \frac{1}{2}\left[m^2 - \sum_j m_j^2\right]$
set $l := 0$
set $g := \mathbf{0}$
**for** $i = 1$ **to** $m$ **do**
  **for** $j = 1$ **to** $m$ **do**
    **if** $y_i < y_j \wedge f_i \geq f_j$ **then**
      $l = l + C_{y_i,y_j}(1 + f_i - f_j)$
      $g_i = g_i + C_{y_i,y_j}$
      $g_j = g_j - C_{y_i,y_j}$
    **end if**
  **end for**
**end for**
$l = l/n$
$g = g/n$
**return** l,g
```

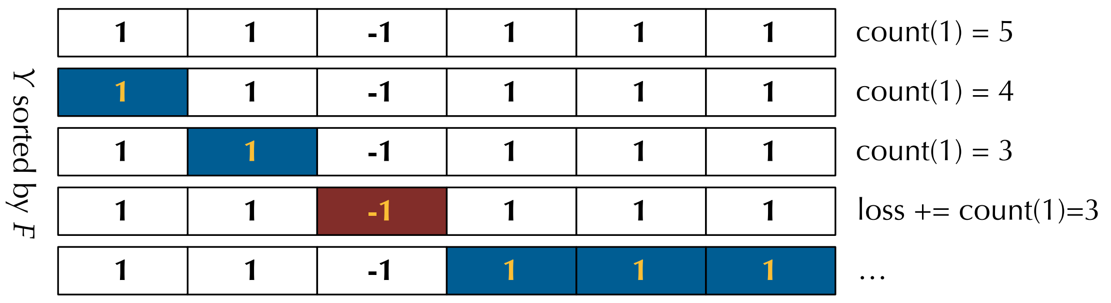


Figure 4.2: The fast procedure to compute the ordinal regression loss.

Fast Ordinal Regression

In many real world recommender systems, an algorithm that scales quadratic in the number of items is unacceptable. These systems often have millions, sometimes hundreds of millions of users and in the order of hundreds of thousands of items. For example, the subset of the data released by Netflix, a commercial movie rental service, as part of the Netflix Prize consists of data about more than seventeen thousand movies and roughly half a million users. When learning to order artifact elements for an instance of the Machine Teaching approach, a quadratic algorithm might be a major hurdle when delivering real time predictions to the learner.

Therefore, this section presents a novel algorithm to compute the ordinal regression loss and its gradient that overcomes this limitation. Its time complexity will be shown to scale in $\mathcal{O}\left(m \log\left(m\right)\right)$ of the input length y, which presents a major increase in performance over the state-of-the-art. First, an intuitive explanation and visualization of the main idea of the new algorithm will be given. This is followed by the presentation of the actual algorithm in more formal terms.

Intuition: Given y and f, one can copy y as z. This vector z is then ordered by the order induced by sorting f. This is can be done in $\mathcal{O}\left(m \log\left(m\right)\right)$ time. Additionally, the number of elements with value j in y, denoted my m_j can be computed in one linear sweep over y. Given the results of both of these operations, the loss value can be computed in a single pass over z my decreasing m_j for each visited element in z. The counters m_j for entries smaller than the current one should all be 0. If not, their sum is added to the loss value. The main computational cost of this procedure is the sorting of f. Thus, the whole process takes $\mathcal{O}\left(m \log\left(m\right)\right)$ time.

Figure 4.2 visualizes this intuition for only two possible labels in y, $\mathbb{T} = \{-1, 1\}$. First, we compute the number of elements with rating 1 as $count(1)$. Then, in one pass over the data, we decrease $count(1)$ if the current value is 1. If it is -1, we know that this -1 is part of $count(1)$ wrong orderings in f. Thus, we can increase the loss value accordingly. This idea was first presented in [Joa06].

Algorithm: This observation is used to form the novel Algorithm 4.4.3. Note that the first loop, beginning in line 5, would be sufficient to compute the loss, but not the gradient. It follows from the gradient in Equation (4.33) that each element in f appears twice in the sums: Once for potentially being too big, and once for being potentially too small in each comparison. The second loop, beginning in line 13, thus computes the remaining contributions to the gradient.

The main computational cost stems from line 3, where the *argsort* of f is computed, a $\mathcal{O}(m \log(m))$ operation. The two following loops, are both linear in m and do not change the complexity of the algorithm. Thus, Algorithm 4.4.3 is of time complexity $\mathcal{O}(m \log(m))$.

Algorithm 2 Computation of the ordinal regression loss $l_{ord}(f, y)$ and its gradient $\partial_f l_{ord}(f, y)$ in $\mathcal{O}(m \log m)$ time

```
1: input Vectors $f$ and $y \in \{1, ..., \hat{z}\}$ score matrix $C$
2: output Loss $l := l(f, y)$ and gradient $g := \partial_f l(f, y)$
3: Initialize $l := 0, g := \mathbf{0}, m := length(y), p := argsort(f)$

   // Test for items placed too early:
4: $m_j :=$ Number of elements of value $j$ in $y$
5: for $i = 1$ to $m$ do
6:    $z := y_{p[i]}, m_z := m_z - 1$
7:    for $k := 1$ to $z - 1$ do
8:       $l := l + m_k C(z, k)$
9:       $g_{p[i]} := g_{p[i]} + m_k C(z, k)$
10:   end for
11: end for

   // Test for items placed too late:
12: $m_j :=$ Number of elements of value $j$ in $y$
13: for $i := m$ to $1$ do
14:    $z := y_{p[i]}, m_z =: m_z - 1$
15:    for $k := z + 1$ to $max(z)$ do
16:       $g_{p[i]} := g_{p[i]} - m_k C(z, k)$
17:    end for
18: end for

   // Normalization:
19: $s := \frac{1}{2}\left[m^2 - \sum_j m_j^2\right]$
20: return $\frac{l}{s}, \frac{g}{s}$
```

Conclusion: The presented algorithm presents a significant improvement in run-time complexity. Therefore, the use of the logistic regression loss function is now possible in areas where the problem size prohibited to do so before.

4.4.4 An NDCG Loss Function for Matrix Factorization

In many scenarios where a ranking is ultimately sought from the recommender system, *ranking measures* are used to evaluate the ranking performance of the system, given known rankings given by the user. These measures are formed based on an understanding of what a good ranking is in the specific domain. One common paradigm in these ranking measures is a weight decay of the ranking importance: Elements ranked highly are more important than elements ranked low on the final list. This follows the observation that users are only willing to consider a certain number of items on a list presented to them and that they scan the list from the top.

In this section, a novel ranking loss function based on one measure, namely the Normalized Discounted Cumulative Gain (NDCG) is introduced. First, the measure itself is described and some important properties are discussed. In the second part of this section, the conversion of this measure to a loss function as well as a convex upper bound on it will be introduced. Finally, the equations for computing its gradient are derived.

Notation

In order to define this measure, several notations need to be introduced. Throughout this section, subscripts (e. g. a_i) and array notation (e. g. $a[i]$) are used interchangeably to allow for a clear and concise presentation.

Definition 12 (Permutation Vector). *A Permutation Vector π is a vector which contains the indices of another vector, here denoted by a. If the vector a is permuted by π, $\pi[i]$ is the index of the element in a that appears at position i after the permutation.*

Example 8. *π_s is a permutation that sorts a decreasingly. Then, the following is true:*

$$a[[\pi_s[i]] \geq a[\pi_s[i+1]] \forall i \in \{0 \ldots n-2\}$$

a_π shall denote the vector a, permuted by π such that the following is true:

$$a_\pi[i] = a[\pi[i]] \quad \forall a, \pi$$

The NDCG measure:

The Normalized Discounted Cumulative Gain (NDCG) is a measure commonly used in Information Retrieval settings to evaluate the ranking performance of a retrieval system [Voo01]. It is appropriate to evaluate recommender systems, too, as the recommender system will retrieve a set of items, given the user as a query. The NDCG

Perfect permutation	★★★★★	★★★★	★★★	★★	★	Σ
Contribution to the DCG	44.72	13.65	5.05	1.86	0.56	65.84

First wrong	★★★	★★★★	★★★★★	★★	★	Σ
Contribution to the DCG	10.10	13.65	22.36	1.86	0.56	48.53

Last wrong	★★★★★	★★★★	★	★★	★★★	Σ
Contribution to the DCG	44.72	13.65	0.72	1.86	3.91	64.85

Figure 4.3: Visualization of the sensitivity of DCG to different errors. Note that the "perfect permutation" might not always be obtainable in real data due to ties.

formulation is based on the following desirable properties for a performance measure in this context:

- Mistakes at the beginning of the ranked list shall be punished more than those at the end of the list. More formally, there should be *decay* of the loss induced over the length of the list.
- The users of a real system are only willing to consider k items, typically in the order of 10. Thus, the measure has a *cutoff* parameter, after which mistakes are not considered any more.
- The measure shall be independent of the type $\mathbb{T}$ of y.

Based on these informal requirements, the DCG and subsequently the NDCG can be defined:

Definition 13 (DCG)**.** *The* DISCOUNTED CUMULATIVE GAIN (DCG) *with cut-off k is defined on a sequence y and a permutation π and can be computed as:*

$$DCG(y, k, \pi) = \sum_{i=0}^{k-1} \frac{2^{y_\pi[i]} - 1}{log_2(i+2)} \tag{4.34}$$

Equation (4.34) is maximized for a permutation π_s that sorts y decreasingly. The denominator assures that small values at the beginning and large values at the end of y_π are yielding little gain. The sum in the numerator of Equation (4.34) runs over k and thus removes the influence of elements permuted to positions after k.

Example 9. *Figure 4.3 shows three computations of the DCG. The first row depicts the perfect permutation. The second row shows that an error in the first half of the permutation has a stronger influence on the outcome as the error shown in the last row that occurred in the second half of the permuted list.*

However, the formulation of the DCG is not invariant to scaling y. In fact, $DCG(l * y, k, \pi) = l * DCG(y, k, \pi)$. Thus, predictions for users who prefer to rate items highly will be evaluated better than those who rate items using lower labels. To overcome this, the *Normalized DCG* (NDCG) is introduced:

Definition 14. *The* NORMALIZED DISCOUNTED CUMULATIVE GAIN (NDCG) *of a vector y, permutation π and cutoff k is defined as:*

$$NDCG(y, k, \pi) = \frac{DCG(y, k, \pi)}{DCG(y, k, \pi_s)} \tag{4.35}$$

where π_s is the permutation which sorts Y decreasingly.

The NDCG is maximized for the permutation π that sorts y decreasingly. Additionally, the NDCG is bounded between 0.0 and 1.0. Thus, the NDCG fulfills the last of the desirable properties outlined above.

Construction of a convex loss function for NDCG

The remainder of this section shows how to derive a convex differentiable loss function from the NDCG in Equation (4.35). This derivation draws inspiration from [TJHA05, TGK04] and is presented in the following steps:

1. Conversion of the gain $NDCG$ into a loss.
2. Construction of a convex upper bound on the so-found loss function.

Finally, it is shown how to compute the loss function value and its gradient by reducing it to a linear assignment problem.

Step 1: Loss conversion

To convert the gain function in Equation (4.35) into a loss function, we rely on the fact that it is bounded from above by 1.0. Thus, the loss function for NDCG for a given cutoff k can be defined as:

$$\Delta(\pi, y) = 1 - NDCG(y, k, \pi) \tag{4.36}$$

This loss function assumes the value of 0 for the permutation π_s that sorts y decreasingly. Note that this function is not convex in π. In fact, it is piecewise constant.

Step 2: Derivation of the upper bound

Definition 15. *Let c be a decreasingly sorted, non-negative vector and f be the prediction. Then we define:*

$$l(f, y, \pi) = \max_{\pi}[\Delta(\pi, y) + \langle c, f_\pi - f \rangle] \tag{4.37}$$

It will now be shown that $l(f, y, \pi)$ as defined in Equation (4.37) is a convex upper bound on $\Delta(\pi, y)$ (4.36).

Lemma 1. *The function (4.37) is convex in f and an upper bound to the loss function (4.36).*

Proof. The proof is done in two phases. First, the convexity of (4.37) in f is shown. Second, the fact that (4.37) is an upper bound to (4.36) is proven.

Convexity: The argument of the maximization over the permutations π is linear and thus a convex function in f. Taking the maximum over a set of convex functions is convex itself, which proves the first claim.

Upper Bound: In order to proof that (4.37) is an upper bound to (4.36), we apply the insight that the inner product between two vectors is maximized if both vectors are sorted by the same criterion.

Let $\pi^* := \text{argsort}(-f)$ be the ranking induced by f and c be a decreasingly sorted, non negative vector. To see that it is an upper bound, we use the fact that

$$l(f, y) \geq \Delta(\pi^*, y) + \langle c, f_{\pi^*} - f \rangle \geq \Delta(\pi^*, y). \tag{4.38}$$

The second inequality follows from the fact that π^* maximizes $\langle c, f_{\pi^*} \rangle$ by the Polya-Littlewood-Hardy inequality.

Thus, it can be concluded that (4.37) is convex in f and an upper bound to the loss function (4.36), which proofs the claim. □

Step 3: Derivation of the gradient

For optimization purposes, the gradient of $l(f, y, \pi)$ with respect to f needs to be derived. The fist step in this derivation is based upon the fact that the gradient is defined on the maximizer $\bar{\pi}$ of $l(f, y, \pi)$:

$$\begin{aligned}
\partial_f l(f, y, \pi) &= \partial_f \max_{\pi}[\Delta(\pi, y) + \langle c, f_\pi - f \rangle] \\
&= \partial_f \langle c, f_{\bar{\pi}} - f \rangle + \partial_f \Delta(\bar{\pi}, y) \\
&= \partial_f \langle c, f_{\bar{\pi}} - f \rangle + 0 \\
&= \partial_f \langle c, f_{\bar{\pi}} - f \rangle
\end{aligned}$$

The sum in the inner product can be decomposed, leaving us with:

$$\begin{aligned}\partial_f \langle c, f_{\bar{\pi}} - f\rangle &= \partial_f \langle c, f_{\bar{\pi}}\rangle - \partial_f \langle c, f\rangle \\ &= \partial_f \langle c, f_{\bar{\pi}}\rangle - c\end{aligned}$$

To solve the gradient $\partial_f \langle c, f_{\bar{\pi}}\rangle$, we use the insight that $\langle a, b_\pi\rangle = \langle a_{\pi^{-1}}, b\rangle$, where π^{-1} denotes the inverse permutation to π:

$$\begin{aligned}\partial_f \langle c, f_{\bar{\pi}} - f\rangle &= \partial_f \langle c_{\bar{\pi}^{-1}}, f\rangle - c \\ &= c_{\bar{\pi}^{-1}} - c\end{aligned}$$

Therefore, we can compute the final gradient as:

$$\partial_f l(f, y, \pi) = c_{\bar{\pi}^{-1}} - c \tag{4.39}$$

Computation of the loss

Computing the value of the loss function (4.37) and its gradient (4.39) is a challenge, as the maximizing permutation $\bar{\pi}$ is needed which in worst case may mean that all possible permutations have to be considered. Below, it is shown that the solution for this maximization can be found by solving a linear assignment problem with a cost matrix specific to this problem.

Lemma 2. *The solution of Equation (4.37) can be found by solving the following linear assignment problem:*

$$min \sum_i \sum_j C_{i,j} * X_{i,j}$$

subject to:

$$\begin{aligned}\sum_i X_{i,j} &= 1 \quad \forall j \\ \sum_j X_{i,j} &= 1 \quad \forall i \\ X_{i,j} &\geq 0 \quad \forall i, j\end{aligned}$$

with a cost matrix:

$$C_{i,j} = \kappa_i \frac{2^{y[j]} - 1}{DCG(y, k, \pi_s) log_2(i+2)} - c_i f_j$$

The solution to the linear assignment problem is integral, as the constraint matrix X is totally unimodular. This means that $X_{i,j}$ is either 1 or 0 for all i, j. Thus, we can transform the solution X back into a permutation π by setting $\pi[i]$ = the j for which $X_{i,j}$ is 1. This π is the solution to Equation (4.37).

Proof. Recall

$$l(f, y, \pi) = \max_{\pi}[\Delta(\pi, y) + \langle c, f_\pi - f\rangle] \tag{4.40}$$

where $\Delta(\pi, y) = 1 - NDCG(y, k, \pi)$ for some arbitrary but fixed k. f is the current prediction and f_π is the current prediction permuted by π. Using this notation, the following derivations are possible:

$$\begin{aligned}
l(f, y, \pi) &= \max_{\pi}\,[1 - NDCG(y, k, \pi) + \langle c, f_\pi - f\rangle] \\
&= \max_{\pi}\,[\langle c, f_\pi - f\rangle - NDCG(y, k, \pi)] + 1 \\
&= \max_{\pi}\,[\langle c, f_\pi\rangle - NDCG(y, k, \pi)] + 1 - \langle c, f\rangle \\
&= \max_{\pi}\left[\langle c, f_\pi\rangle - \frac{DCG(y, k, \pi)}{DCG(y, k, \pi_s)}\right] + 1 - \langle c, f\rangle
\end{aligned}$$

The π which maximizes this equation also maximizes the following one, as $1 - \langle c, f\rangle$ are independent of π:

$$\begin{aligned}
\max_{\pi}\left[\langle c, f_\pi\rangle - \frac{DCG(y, k, \pi)}{DCG(y, k, \pi_s)}\right] &= \max_{\pi}\left[\langle c, f_\pi\rangle - \frac{1}{DCG(y, k, \pi_s)}\sum_{i=0}^{k-1}\frac{2^{y[\pi[i]]} - 1}{log_2(i+2)}\right] \\
&= \max_{\pi}\left[\langle c, f_\pi\rangle - \sum_{i=0}^{k-1}\frac{1}{DCG(y, k, \pi_s)}\frac{2^{y[\pi[i]]} - 1}{log_2(i+2)}\right] \\
&= \max_{\pi}\left[\langle c, f_\pi\rangle - \frac{2^{y[\pi[i]]} - 1}{DCG(y, k, \pi_s)log_2(i+2)}\right]
\end{aligned}$$

Defining

$$X_{i,j} = \begin{cases} 1 & \text{if } \pi[i] = j, \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad \kappa_i = \begin{cases} 1 & \text{if } i < k, \\ 0 & \text{otherwise} \end{cases}$$

allows us the following reformulation:

$$\begin{aligned}
& \max_{\pi} \left[\langle c, f_\pi \rangle - \sum_{i=0}^{k-1} \frac{2^{y[\pi[i]]} - 1}{DCG(y,k,\pi_s) log_2(i+2)} \right] \\
= & \max_{X} \left[\sum_i \sum_j c_i f_j X_{i,j} - \sum_j \sum_{i=0}^{k-1} X_{i,j} \frac{2^{y[j]} - 1}{DCG(y,k,\pi_s) log_2(i+2)} \right] \\
= & \max_{X} \left[\sum_i \sum_j c_i f_j X_{i,j} - \sum_i \sum_j \kappa_i \frac{X_{i,j}(2^{y[j]} - 1)}{DCG(y,k,\pi_s) log_2(i+2)} \right] \\
= & \max_{X} \left[\sum_i \sum_j c_i f_j X_{i,j} - \kappa_i \frac{X_{i,j}(2^{y[j]} - 1)}{DCG(y,k,\pi_s) log_2(i+2)} \right] \\
= & \max_{X} \left[\sum_i \sum_j X_{i,j}(c_i f_j - \kappa_i \frac{2^{y[j]} - 1}{DCG(y,k,\pi_s) log_2(i+2)} \right]
\end{aligned}$$

Thus, the problem can be solved using the following linear assignment problem formulation:

$$min \sum_i \sum_j C_{i,j} * X_{i,j}$$

subject to:

$$\begin{aligned}
\sum_i X_{i,j} &= 1 \quad \forall j \\
\sum_j X_{i,j} &= 1 \quad \forall i \\
X_{i,j} &\geq 0 \quad \forall i, j
\end{aligned}$$

with:

$$\begin{aligned}
C_{i,j} &= -\left(c_i f_j - \kappa_i \frac{2^{y[j]} - 1}{DCG(y,k,\pi_s) log_2(i+2)} \right) \\
&= \kappa_i \frac{2^{y[j]} - 1}{DCG(y,k,\pi_s) log_2(i+2)} - c_i f_j
\end{aligned}$$

The solution to this linear assignment problem is integral, $X_{i,j}$ is either 1 or 0 for all i, j. Thus, we can transform the solution X back into a permutation π by setting $\pi[i] =$ the j for which $X_{i,j}$ is 1. Using this permutationπ, the value of Equation (4.37) can be computed, which proofs the claim. □

4.4.5 Conclusion

In this section, we presented several choices regarding the loss function to be used. In addition to transferring some well-known element based loss functions from supervised machine learning to matrix factorization, we introduced the concept of row based loss functions. Two loss functions have been discussed in this context: We derived a faster algorithm for the computation of the ordinal regression loss function and developed a loss function which can be used to directly optimize the prediction for the NDCG evaluation measure.

We did not yet, however, present how to actually optimize the objective function (4.8). The next section presents an optimization procedure to do so.

4.5 Optimization

Now that suitable choices for the loss function have been discussed, the only part of a working matrix factorization algorithm which we did not present yet is the optimization procedure used to minimize the objective function (4.8).

This procedure is described in this section. As briefly mentioned when introducing the loss functions above, we assume the loss function to be convex and differentiable in f, or at least sub-differentiable. However, this assumption does not guarantee that the objective function (4.8) is jointly convex in C and R. In fact, the contrary is true: Equation (4.8) is not jointly convex in C and R:

Lemma 3. *Equation (4.8) is not guaranteed to be jointly convex in C and R.*

Proof. Let us assume a very simple case of Matrix Factorization: $C, R \in R^1$ and the least squares loss function. Then, the loss function becomes:

$$L(F,Y) = ((F) - Y)^2 = (RC^\top)^2 - 2RC^\top Y + Y^2$$

A function is convex if its hessian is positive semi-definite. The hessian of this loss can be computed based on the following partial derivatives:

$$\begin{aligned}
L_U(F,Y) &= 2RC^{\top 2} - 2C^\top Y \\
L_M(F,Y) &= 2R^2C^\top - 2RY \\
L_{UU}(F,Y) &= 2C^{\top 2} \\
L_{MM}(F,Y) &= 2R^2 \\
L_{UM}(F,Y) &= 4RC^\top - 2Y \\
L_{MU}(F,Y) &= 4RC^\top - 2Y
\end{aligned}$$

This leads to the following hessian:

$$H = 2\begin{pmatrix} C^{\top 2} & 2RC^\top - Y \\ 2RC^\top - Y & R^2 \end{pmatrix}$$

If H is were semi-definite, the function L was jointly convex in R and C. A matrix is positive semi-definite if its determinant is positive. However, the determinant of H is:

$$\begin{aligned} det(H) &= 2(C^{\top 2}R^2 - (2RC^\top - Y)^2) \\ &= 2(C^{\top 2}R^2 - 4R^2C^{\top 2} + 4RC^\top Y - Y^2) \\ &= 2(-3R^2C^{\top 2} + 4RC^\top Y - Y^2) \\ &= 8RC^\top Y - 6R^2C^{\top 2} - 2Y^2 \end{aligned}$$

It is easy to see that for every fixed Y, there will be R and C such that $6R^2C^{\top 2}$ will be bigger than $8RC^\top Y$ which renders the determinant negative. Thus, the objective function (4.8) is not guaranteed to be jointly convex in R and C for a simple special case, which proofs the claim. □

Efficient optimization of non-convex functions is challenging and multiple different approaches such as Stochastic Gradient Descent [Bot04], Semi-Definite Programming [SRJ05] and Subspace Gradient Descent [RS05] have been proposed to find the solution for factor models. Semi-Definite Programming as described in [RS05] is theoretically the most elegant solution, as it is guaranteed to find a globally optimal solution. However, the computational requirements of doing so a prohibitive for all but the smallest problems.

The Stochastic Gradient Descent methods are computationally elegant as they can be implemented in an online algorithm that optimizes in multiple linear sweeps over the data. However, they are based on the assumption of a per-element loss function which, for the reasons outlined above, is not desirable in the Machine Teaching and some important Recommender Systems scenarios.

Thus, an algorithm based on Subspace Gradient Descent is presented here. The key insight behind Subspace Gradient Descent is the fact that the objective function (4.8) is convex in C if R is kept fixed and convex in R if C is kept fixed. Thus, we can optimize by alternating between optimization steps over R and C as shown in Algorithm 3.

Algorithm 3 Alternate Subspace Descent for Matrix Factorization

```
Initialize $R$ and $C$ randomly
repeat
  For fixed $C$ minimize (4.8) with respect to $R$.
  For fixed $R$ minimize (4.8) with respect to $C$.
until no more progress is made or a maximum iteration count is reached.
```

Note that as the overall optimization problem is still non-convex this procedure cannot be guaranteed to converge to a global optimum. However, experimental evaluations show that an implementation of this procedure consistently finds solutions of equal quality, if not the same solution when minimizing the objective function on the same data set but different random initializations of R and C.

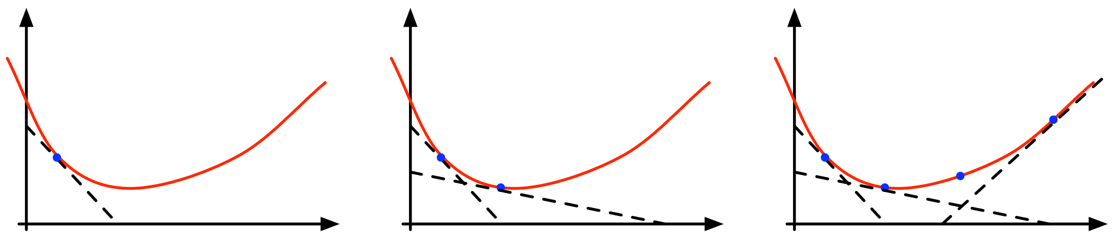


Figure 4.4: A convex function (solid) is bounded from below by Taylor approximations of first order (dashed). Adding more terms improves the bound.

Both the optimization steps in Algorithm 3 are convex and thus amenable to a wide range of optimization algorithms for convex problems such as the well known LBFGS algorithm[NW99, YVGS08]. However, these methods are guaranteed to converge within $\frac{1}{\epsilon^2}$ steps at best to a solution that is within ϵ of the minimum.

Recently, bundle methods have been introduced with promising results for optimizing regularized risk functions in supervised machine learning. Bundle Methods have been shown to converge within $\frac{1}{\epsilon}$ steps [SVL08]. This makes them especially suitable to the optimization problem at hand. Each step of the optimization algorithms requires a loss and gradient computation. As we have shown above, this can be a costly operation, especially for losses like NDCG and others following the structured estimation framework [TJHA05].

The key idea behind bundle methods is to compute successively improving linear lower bounds of an objective function through first order Taylor approximations as shown in Figure 4.4. Several lower bounds from previous iterations are bundled in order to gain more information on the global behavior of the function. The minimum of these lower bounds is then used as a new location where to compute the next approximation, which leads to increasingly tighter bounds and convergence.

Gradient Computation: The main computational cost in using any gradient based solver is the computation of the gradients with respect to R and C. Using the chain rule yields

$$\begin{aligned} \partial_C L(F,Y) &= \partial_F L(F,Y) R' && (4.41)\\ \partial_R L(F,Y) &= [\partial_F L(F,Y)]' C. && (4.42) \end{aligned}$$

By definition, the loss functions discussed in this thesis are decomposable per row of F. Thus, the computation of $\partial_F L(F,Y)$ decomposes per row, too. This observation facilitates a straight forward parallel implementation of this computation.

4.5.1 Optimization over the Row Matrix R

The same observation can be applied to the optimization of (4.8) over R for a fixed C. Not only does the loss function decompose per row, but so does the Frobenius norm used as the regularizer. Thus, the optimization over R decomposes into r convex optimization problems, one per row of Y and F. This observation is used in Algorithm 4 to form an efficient optimization procedure for the first step in Algorithm 3, the optimization over R. Note that in fact, each of the per-row optimization problems is a standard regularized risk minimization problem as discussed in the supervised machine learning literature. Example: When choosing the Hinge loss function, the problem is exactly that of a linear support vector machine.

Algorithm 4 Optimization over R with fixed C

```
  input Parameter matrices $R$ and $C$, data matrix $Y$
  output Matrix $R$
  for $i = 1$ to $r$ do
    Select idx as the nonzero set of $Y_{i*}$
    Initialize $w = R_{i*}$
    $R_{i,\mathtt{idx}} = \operatorname{argmin}_w l(wC_{\mathtt{idx},*}, Y_{i,\mathtt{idx}}) + \frac{\lambda_r}{2} \|w\|^2$
  end for
```

4.5.2 Optimization over the Row Matrix C

The optimization over C is not amenable to the same treatment, as the loss function, by design, does not decompose per column of Y and F. Thus, the complete gradient and loss over C needs to be computed. As introduced earlier, the computations of the loss value and its gradient with respect to F can still be decomposed per row. The main computational bottleneck then becomes the final multiplication to compute:

$$\partial_C L(F,Y) = R' \partial_F L(F,Y) \tag{4.43}$$

This formulation, if implemented naively, poses a serious challenge regarding the memory consumption. Recall that $\partial_F L(F,Y)$ is of the same size of F and that the latter is never constructed in memory, as it is too large in many Recommender Systems instances which have millions of rows (users) and thousands of columns (items). Therefore, the construction of $\partial_F L(F,Y)$ in memory should also be avoided.

However, special care can be taken when implementing this multiplication to reduce the memory consumption of this procedure by iteratively computing the entries of $\partial_C L(F,Y)$ in one pass over all rows of F and Y. Algorithm 5 is build upon this notion.

Algorithm 5 Efficient computation of $\partial_C L$

```
input Parameter matrices $R$ and $C$, data matrix $Y$
output Gradient of the loss function $L$ w.r.t $C$: $\partial_C L$
initialize with 0: $\partial_C L \leftarrow \mathbf{0}^{c \times d}$
for $i = 1$ to $r$ do
    Find index $idx$ where $Y_{i*} \neq 0$
    $D \leftarrow \partial_F L(R_{i*} C'_{idx,*}, Y_{i,idx})$
    $\partial_C L \leftarrow \partial_C L + R'D$
end for
return $\partial_C L$
```

4.5.3 New Row Optimization

In many scenarios, the system needs to react to new lines in Y. In Recommender Systems, these new lines correspond to new users joining the system. In Machine Teaching, new lines in Y are the default case: Each new line in Y corresponds to a new, partially finished artifact. The system is then asked to perform predictions on this partial artifact to support the learner.

For a large number of rows present in Y, one more is likely to have little influence on C. Additionally, the optimization with respect to R decomposes per row of Y. These observations enable minimizing the objective function (4.8) with respect to this one new row in Y for a fixed C in order to obtain a new row in R and thus a prediction for this new row.

Note that this is similar to the regularized risk minimization formulation of supervised machine learning. The main difference is that the known features of the supervised scenario, typically denoted by X, are replaced here with the *learned* features of the columns contained in C.

4.6 Extensions to the Regularized Matrix Factorization Model

Given the previous sections, a matrix factorization model can be constructed for many Machine Teaching and Recommender Systems scenarios. In this section, several extensions to this model are introduced that facilitate better prediction performance. It is also discussed how to integrate these extensions to the optimization procedure described in Section 4.5.

4.6.1 Row and Column Biases

Each row and each column of Y may carry an individual bias. In Recommender Systems, this bias may be the result of the individual user's rating habits for the rows. The columns may be biased by a universal quality associated with a certain item. For instance, 'Plan 9 from Outer Space' will probably not attract lots of high ratings while other movies may prove universally popular. Similar biases are conceivable in Machine

Teaching scenarios where a certain structure element value almost always is around a specific value. That value may then be used as the offset in the sense of this extension.

Biases like this can be introduced into the model by expanding the prediction function from Equation (4.1) with two more terms: The row bias r_i and the column bias c_j: This can be taken into account by means of an offset per movie. This can be incorporated via

$$F_{i,j} = \langle R_{i*}, C_{j*} \rangle + r_i + c_j. \tag{4.44}$$

Incorporation of these bias terms into the optimization procedure is done by extending R and C by two columns each: One for the bias vector, and one with a constant value of 1.

In this form no algorithmic modification for the R and C optimization is needed. The computational cost of this extension can be neglected, as the number of factors d is increased by only 2.

Note that this offset is different from a simple normalization of the input data and is not meant to replace pre-processing procedures altogether. The offset is learned for the loss function in use and for each row and column, while it would be tricky to find a normalization that does cater for both appropriately at the same time.

4.6.2 Adaptive Regularization

In many cases, the rows and columns of Y are not evenly filled. In a movie recommender scenario, some users rated thousands of movies while others only a few. Thus, a universal regularization parameter for all these rows is not advisable, as rows with many entries should be allowed a more 'complicated' model than those with only a few. The same argument holds for the columns in Y. For example in a Coding Machine Teaching System where some methods are universally popular and thus get called often.

Those issues can be dealt with by sample-size adaptive regularization for both columns and rows. Denote by D^r and D^c diagonal matrices corresponding to rows and columns of Y. Setting $D^r_{ii} = n_i^{-\alpha}$ and $D^r_{jj} = m_j^{-\alpha}$ where n_i denotes the number of entries in row i and m_j denotes the number of entries in column j, we obtain a sample size dependent regularizer as follows:

$$\min_{U,M} L(F,Y) + \frac{\lambda_c}{2} \operatorname{tr} C^\top D^c C + \frac{\lambda_r}{2} \operatorname{tr} R^\top D^r R.$$

In our experiments (reported in Chapter 5) we found that $\alpha = 0.5$ provides best generalization performance. This is equivalent to the regularization scales provided in a maximum a posteriori setting where the log-prior is fixed whereas the evidence scales linearly with the number of observations.

As the computation for this scaling can be done in advance, the computational cost of the adaptive regularizer is not significant when compared to the overall run time. The needed statistics can even be pre-computed and reused in many experiments.

4.6.3 Structure Exploitation with a Graph Kernel

In many instances, the sheer presence of an entry in the matrix Y carries important information in addition to the actual value stored at that point in Y, e. g. in the movie recommender domain. The fact that a user bothered to watch and subsequently rate a movie is an important piece of information that is genuinely distinct from the numerical value of the rating given. After all, renting, watching and labeling a movie represents a substantial effort, which the user presumably only invests into movies she is likely to like.

Example 10. *Knowing that a user* rated *'Die Hard', 'Die Hard 2', and 'Top Gun' makes it likely that this user is interested in action movies.*

One would expect that we should be able to take advantage of this structural information in addition to the actual scores. One possibility, proposed in [BH04] is to use the inner product between two rows of Y as a kernel for comparing two different rows. In this case they define the kernel between the rows i and i' to be $\langle S_{i*}, S_{i'*} \rangle$. It is well known that such a model is equivalent to using a linear model with row-features given by S.

The main drawback of this approach is that one needs to compute this kernel function between all users which scales quadratic in their number. Instead, one can use another formulation which uses the same insight, the importance of the binary aspects of Y in a computationally more efficient framework. To do so, we introduce an additional parameter matrix $A \in \mathbb{R}^{c \times d}$ to replace R with $R + SA$, which leads to the following prediction function:

$$F = (R + SA)\, C^\top = RC^\top + SAC^\top \tag{4.45}$$

In this formulation, A controls a mixture of column features to be assigned to each row. In the movie recommender example above, every user now partially assumes features of the movie she watched based on the mixing coefficients stored in A. Or, in other words, the proposed solution is based upon the assumption that "we are what we watch" up to a level encoded in A. We can optimize over A in a straight-forward manner by introducing a third step into the subspace descent described in Algorithm 3.

A similar approach has been proposed in [SM08], whereas a row-normalized version of S is used. In [WKS08c], it has been shown that all three approaches (the one presented here, the one in [SM08] and the one in [BH04]) are in fact equivalent.

4.6.4 Row and Column Features

Depending on the application scenario, features of both the rows and the columns may be available. In a Recommender System, demographic information is often known about the users and the items can be described by a set of features, too. In [ABEV06], the integration of features is proposed by defining a kernel between rows and columns that integrates features. Another way of introducing features is to use them as a prior for the factors, as studied in [AC09].

Here, we present a integration of features to our matrix factorization framework by adding several linear supervised learning models: Following common supervised machine learning notation, the rows of $X^R \in \mathbb{R}^{r \times d_R}$ shall contain the d_R feature vectors for the rows of Y. The rows of $X^C \in \mathbb{R}^{c \times d_C}$ contain those d_C features of the columns of Y.

Using these two new matrices, the prediction function from Equation (4.1) can be extended to use column features:

$$F_{i,j} = \langle R_{i*}, C_{j*} \rangle + \left\langle W^R_{i*}, X^C_{j*} \right\rangle \tag{4.46}$$

Here, $W^R \in \mathbb{R}^{r \times d_C}$ is a matrix whose rows are the weight vectors for the column features per row. The very same idea can be applied to row features as well:

$$F_{i,j} = \langle R_{i*}, C_{j*} \rangle + \left\langle W^R_{i*}, X^C_{j*} \right\rangle + \left\langle W^C_{i*}, X^R_{j*} \right\rangle \tag{4.47}$$

The matrix $W^M \in \mathbb{R}^{c \times d_U}$ encodes the weight vector for each column.

As with the offsets, the features can be integrated into the algorithmic procedure described above without any changes. To do so, one would extend C with the features from X^C, R with the features from X^R. These new entries are masked from optimization and regularization such that their value stays fixed.

To learn the parameters, R is extended by W^R and C by W^C. The optimization over R and C includes these new entries. This yields the parameter vectors W^R and W^C. Please note that the Frobenius norm decomposes per entry in R and C. Thus, the newly introduced parameters are regularized as if there would be a L_2 norm imposed on them. This essentially recovers a regularized risk minimization problem for these weight vectors.

4.7 Conclusion

In this chapter, a generalized matrix factorization model and algorithm has been presented. As shown at the beginning of this chapter (Section 4.1 on page 70), the model underlying this approach can be applied both to recommender systems and Machine Teaching applications. The presented approach is a *generalization* of the state-of-the-art in several respects.

Regularizer: While this thesis focuses on the Frobenius norm as the model regularizer, many more choices such as the L_1-norm are possible without leaving the framework presented here. The adaptive regularization can be applied to a wide range of regularizers, too.

Loss Functions: The model and algorithm here can be used with a wide range of loss functions known in the supervised machine learning community.

Especially notable is the possibility to use non-smooth loss functions due to the use of the bundle method solver. Even more so, the applicability of matrix factorization to per-row loss functions is a major contribution as it facilitates the optimization of ranking losses in recommender systems. To this end, one novel loss function to directly optimize for the NDCG score and a substantially faster version of the ordinal regression loss function have been presented. Per-row losses also are a crucial ingredient in making matrix factorization a viable method for a large class of Machine Teaching systems.

Hybrid Approach: The inclusion of features as described above extends matrix factorization to be what is known as a "hybrid recommender system": One that can use both the collaborative effect of users interacting on the same set of items as well as explicitly known features of the users and items. This is an important step in addressing the new user problem, which, as we have shown earlier, arises naturally in Machine Teaching.

The graph kernel and the adaptive regularization make it possible to use additional knowledge embodied in the rating matrix or the matrix representation of an artifact in the Machine Teaching setting.

The approach presented in this chapter provides a solid base for the construction of Machine Teaching systems for detailed feedback, as the prediction of suggestions for a new or changed artifact is fast and, as shall be shown below, very accurate. The wide range of possible loss functions allow the system to be used in a multitude of Machine Teaching scenarios and the possibility to use features of the artifact or the structural elements thereof can further increase this performance.

Next steps in this thesis: Before applying the algorithm in a Detailed Feedback Machine Teaching approach in Chapter 6, we first evaluate it on well established Recommender Systems data sets to compare its performance to the state-of-the-art in that field in Chapter 5.

5 Evaluation on Recommender Systems Data

Contents

5.1 Evaluation Setup

This chapter presents empirical evaluation results of the algorithm discussed so far in the Recommender Systems scenario before moving on to the application of the algorithm to a Detailed Feedback Machine Learning example in the next chapter. The reason for evaluating on Recommender Systems data lies in the fact that this field of research is well established:

- Data sets are readily available.
- The evaluation method including the evaluation measure is widely agreed upon in the field.
- Thus, it is possible to assess the performance of an algorithm in comparison to the related work.

Obviously, none of this is true for the Machine Teaching area. The purpose of the evaluation presented in this chapter therefore is to assure the general performance of the algorithm before depending upon both – the algorithm and its performance – in subsequent steps.

Similar to the evaluation presented in Chapter 3, the evaluation is discussed in two steps: First, the evaluation method including the used measures and the data and the pre-processing thereof are introduced. The remainder of this chapter then presents empirical results and their discussion for the two main questions to be evaluated:

1. Do the model extensions proposed in Section 4.6 add to the performance of the Recommender System?
2. Do the per-row losses improve the ranking performance of the system?

5.1.1 Evaluation Measures

In order to evaluate the predictive performance of the different variants of the model – rating and ranking prediction – we resort to two *evaluation measures*.

Rating Performance: Following the majority of the literature, the rating performance is evaluated with the root mean squared error (RMSE) measure:

$$RMSE(F,Y) = \frac{1}{2n} \sum_{i=0}^{r} \sum_{j=0}^{c} S_{i,j} (F_{i,j} - Y_{i,j})^2 \tag{5.1}$$

Where the binary matrix S indicates whether or not a user rated an item:

$$S_{i,j} = \begin{cases} 1 & \text{if user } i \text{ rated item } j \\ 0 & \text{otherwise} \end{cases}$$

and n is the number of those ratings in Y, e.g.:

$$n = \sum_{i=0}^{r} \sum_{j=0}^{c} S_{i,j}$$

Ranking Performance: In order to evaluate the ranking performance of the system, we resort to the *Normalized Discounted Cumulative Gain (NDCG)* measure as introduced in Section 4.4.4. The NDCG of a vector y, permutation π and cutoff k is defined as:

$$NDCG(y, k, \pi) = \frac{DCG(y, k, \pi)}{DCG(y, k, \pi_s)} \tag{5.2}$$

The permutation π is computed as the `argsort` of the predicted values: $\pi = argsort(f)$. The perfect permutation π_s is the `argsort` of the true ratings given by the user: $\pi_s = argsort(y)$. A NDCG of 1.0 indicates that the model sorts the items in the same order as the user. The parameter k is a cut-off beyond which the actual ranking does no longer matter. This follows the intuition that typical recommender systems can only present a limited amount of items to the user. In all our experiments, we evaluated using $k = 10$, which is commonly referred to as *NDCG@10*. We report the average NDCG@10 over all users below.

5.1.2 Evaluation Procedure

We distinguish two different evaluation scenarios: *strong* and *weak generalization*.

Weak generalization: This is the scenario commonly discussed in the literature: For each user, a number n of ratings is sampled from the known data. The system is then trained on these ratings and evaluated on the remaining items. We present results for $n = 10, 20, 50$. This evaluation resembles a system with an established user base that recommendations are generated for.

Strong generalization: As discussed earlier, the Machine Teaching scenario is dependent on fast and accurate solutions to what is referred to as the new user problem in Recommender Systems research: Given a fully trained model, predict for a user that was not part of the initial training phase of the system.

To simulate this scenario, we follow the procedure suggested in [YYTK06]: Movies with less than 50 ratings are discarded. The 100 users with the most rated movies are selected as the test set and the methods are trained on the remaining users. In evaluation, 10, 20 or 50 ratings are sampled from the test users. We then use these ratings to learn a new user feature matrix R_{strong} by performing a single iteration of the optimization over R as described in Section 4.5.3. The remaining ratings are used as the test set.

Note that sampling a small number of ratings for the training set mimics a real world recommender setting where ratings are very scarce compared to the total number of movies.

Data set	Users	Movies	Ratings	Rating Scale
MovieLens	983	1682	100000	1-5 Stars
EachMovie	61265	1623	2811717	1-6 Stars
Netflix	480189	17770	100480507	1-5 Stars

Table 5.1: Data set statistics

In all experiments, the regularization parameters λ_r and λ_c were fixed and not formally tuned. The dimension of R and C was set to $d = 10$ for the evaluation of the extensions and to $d = 100$ for the evaluation of the different losses. We did not observe significant performance fluctuations when comparing the performance of the system using $d = 30$ in [SM08]. This is an interesting observation in its own right: The preferences of a user for movies can be described by a relatively small number (in the order of 10) of factors.

All experiments were performed ten times with different random draws of the train and test set from the data sets. In total, we report results from more than 5000 experiments.

5.1.3 Data Sets

Evaluations have been conducted on three data sets that are well known in the recommender systems literature:

MovieLens: The MovieLens data stems from the research movie recommender of the same name[1] of the GroupLens research group[2] in the Department of Computer Science and Engineering at the University of Minnesota.

EachMovie: EachMovie was a recommender run by HP/Compaq Research (formerly DEC Research). The data set became available when the recommender was shut down and it has since been used in numerous recommender systems publications. However, the data set was retired in October 2004 and is since no longer available for download.

Netflix: The Netflix data set was collected and published by the online movie rental service Netflix[3] as part of their Netflix Prize challenge[4].

The data sets are of very different sizes, their descriptive statistics can be found in table 5.1.

[1]http://movielens.umn.edu/
[2]http://www.grouplens.org/
[3]http://netflix.com
[4]http://www.netflixprize.com/

	Method	N=10	N=20	N=50
EachMovie	Plain	0.625 ± 0.000	0.639 ± 0.000	0.641 ± 0.000
	Offset	**0.646 ± 0.000**	**0.653 ± 0.000**	**0.647 ± 0.000**
	GraphKernel	0.583 ± 0.000	0.585 ± 0.000	0.590 ± 0.001
	OffsetGK	0.576 ± 0.000	0.597 ± 0.000	0.580 ± 0.001
MovieLens	Plain	0.657 ± 0.000	0.658 ± 0.000	0.686 ± 0.000
	Offset	**0.678 ± 0.000**	0.680 ± 0.000	**0.701 ± 0.000**
	GraphKernel	0.624 ± 0.001	0.644 ± 0.000	0.682 ± 0.000
	OffsetGK	0.670 ± 0.001	**0.681 ± 0.000**	0.682 ± 0.000

Table 5.2: The NGDC@10 accuracy over ten runs and the standard deviation for the weak generalization evaluation.

	Method	N=10	N=20	N=50
EachMovie	Plain	**0.859 ± 0.000**	0.731 ± 0.000	0.627 ± 0.000
	Offset	**0.859 ± 0.000**	**0.734 ± 0.000**	0.631 ± 0.000
	GraphKernel	0.837 ± 0.000	0.693 ± 0.000	0.553 ± 0.001
	OffsetGK	0.832 ± 0.000	0.689 ± 0.000	0.587 ± 0.001
	All	0.836 ± 0.000	0.702 ± 0.000	0.585 ± 0.000
MovieLens	Plain	0.875 ± 0.000	0.750 ± 0.000	0.673 ± 0.000
	Offset	**0.886 ± 0.000**	0.764 ± 0.000	0.703 ± 0.001
	GraphKernel	0.845 ± 0.000	0.720 ± 0.001	0.667 ± 0.000
	OffsetGK	0.882 ± 0.000	**0.773 ± 0.000**	**0.703 ± 0.000**
	All	0.869 ± 0.000	0.730 ± 0.002	0.645 ± 0.005

Table 5.3: The NGDC@10 accuracy over ten runs and the standard deviation for the *inverted* weak generalization evaluation.

5.2 Results and Discussion

5.2.1 Model Extensions

To evaluate the model extensions, we performed experiments for each combination of the extensions as it cannot be guaranteed that the combination of all extensions performs best. As we are interested in the performance of the extensions, all experiments are done with a single loss function, namely the least squares loss.

Weak generalization: Table 5.2 contains the results of the weak generalization scenario experiments. We observe that adding the offset terms yields significant improvements in the performance of the model. On the other hand, enabling the Graph Kernel does not significantly improve performance. This can be attributed to the fact that the regularization factors were not tuned. Enabling the Graph Kernel adds a large set of additional parameters to the model which might lead to overfitting without the adjustment of the regularization parameters. Nonetheless, the Graph Kernel did add further improvements together with the offsets.

Inverted weak generalization: Note that due to the weak generalization procedure definition, the adaptive regularization extension cannot have an influence, as all experiments are done on $10, 20$ or 50 movies per user. Thus, we use the same data set split reversed for evaluating the adaptive regularizer: For each user, $10, 20$ or 50 movies are used as the test set, with all the remaining movies being the training set.

The results of this set of experiments can be found in Table 5.3. They again confirm our observations in the previous setting. The offset term combined with the graph kernel brings significant performance gains.

Strong generalization: For the strong generalization setting, the matrix factorization models were compared to Gaussian Process Ordinal Regression (GPOR) [CG05] Gaussian Process Regression (GPR), their collaborative extensions (CPR, CGPOR) as well as the original MMMF implementation [YYTK06]. Table 5.4 shows the results for the generalized MMMF models compared to the ones from [YYTK06].

For the Movielens data, the model with the offset and graph kernel extensions outperforms the other systems. Additionally, the system with both extensions performs consistently better than the ones with only one extension. On the EachMovie data, the model performs the best with the offset extension enabled. It appears that the Graph Kernel in the EachMovie data set does not improve the performance but again this could be attributed to a poor choice of the regularization parameters for this data set.

The matrix factorization model performance is particularly convincing in the strong evaluation setting. This is especially notable as the systems we compare to do use external features of the movies. These features can be obtained by crawling the Internet Movie Database (IMDB) for information on these movies.

The good performance of the matrix factorization model can be attributed to the fact that the system performs alternate convex optimization steps over item and user features. Once a “good” set of item features is obtained, there is reason to believe that it is a good representation of the items, even for new users. We believe that this is an important benefit of matrix factorization models in many applications, as it enables fast accurate predictions for new users without the need to retrain the whole system.

5.2.2 Ranking Losses

To compare the performance of the different loss functions, the system was trained without the extensions to facilitate the analysis of the loss function’s impact in isolation. Experiments were performed with the NDCG, Ordinal Regression and Least Squares

	Method	N=10	N=20	N=50
EachMovie	Plain	0.615 ± 0.000	0.633 ± 0.000	0.636 ± 0.000
	Offset	$\mathbf{0.641 \pm 0.000}$	$\mathbf{0.647 \pm 0.000}$	$\mathbf{0.644 \pm 0.000}$
	GraphKernel	0.574 ± 0.000	0.581 ± 0.000	0.596 ± 0.000
	OffsetGK	0.568 ± 0.000	0.594 ± 0.000	0.579 ± 0.000
	GPR	0.4558 ± 0.015	0.4849 ± 0.0066	0.5375 ± 0.0089
	CGPR	0.5734 ± 0.014	0.5989 ± 0.0118	0.6341 ± 0.0114
	GPOR	0.3692 ± 0.002	0.3678 ± 0.0030	0.3663 ± 0.0024
	CGPOR	0.3789 ± 0.011	0.3781 ± 0.0056	0.3774 ± 0.0041
	MMMF	0.4746 ± 0.034	0.4786 ± 0.0139	0.5478 ± 0.0211
MovieLens	Plain	0.587 ± 0.001	0.644 ± 0.001	0.630 ± 0.001
	Offset	0.583 ± 0.000	0.444 ± 0.000	0.690 ± 0.000
	GraphKernel	0.613 ± 0.000	0.634 ± 0.000	0.637 ± 0.001
	OffsetGK	$\mathbf{0.684 \pm 0.000}$	$\mathbf{0.691 \pm 0.000}$	$\mathbf{0.692 \pm 0.000}$
	GPR	0.4937 ± 0.0108	0.5020 ± 0.0089	0.5088 ± 0.0141
	CGPR	0.5101 ± 0.0081	0.5249 ± 0.0073	0.5438 ± 0.0063
	GPOR	0.4988 ± 0.0035	0.5004 ± 0.0046	0.5011 ± 0.0051
	CGPOR	0.5053 ± 0.0047	0.5089 ± 0.0044	0.5049 ± 0.0035
	MMMF	0.5521 ± 0.0183	0.6133 ± 0.0180	0.6651 ± 0.0190

Table 5.4: The NGDC@10 accuracy over ten runs and the standard deviation for the strong generalization evaluation.

Regression loss functions. In addition, we also report results obtained with the original MATLAB implementation of the MMMF model where possible[5].

Weak generalization: Table 5.5 contains the results of the experiments. The Ordinal Regression loss performs best overall with some exceptions where the Least Squares Regression Loss performs slightly better.

Strong generalization For the strong generalization setting, the NDCG scores are compared to those reported in [YYTK06] (table 5.6).

The model with the NDCG loss performs strongly compared to most of the other tested methods. Particularly in the strong generalization setting, it outperforms the existing methods in almost all of the settings. Again, note that all methods except MMMF use additional extracted features which are either provided with the data set or extracted from the `IMDB`.

[5]The implementation did not scale to the bigger data sets EachMovie and Netflix.

	Method	N=10	N=20	N=50
EachMovie	NDCG	0.6562 ± 0.0012	0.6644 ± 0.0024	0.6406 ± 0.0040
	Ordinal	$\mathbf{0.6727 \pm 0.0309}$	$\mathbf{0.7240 \pm 0.0018}$	$\mathbf{0.7214 \pm 0.0076}$
	Regression	0.6114 ± 0.0217	0.6400 ± 0.0354	0.5693 ± 0.0428
MovieLens	NDCG	0.6400 ± 0.0061	0.6307 ± 0.0062	0.6076 ± 0.0077
	Ordinal	0.6233 ± 0.0039	0.6686 ± 0.0058	$\mathbf{0.7169 \pm 0.0059}$
	Regression	$\mathbf{0.6420 \pm 0.0252}$	0.6509 ± 0.0190	0.6584 ± 0.0187
	MMMF	0.6061 ± 0.0037	$\mathbf{0.6937 \pm 0.0039}$	0.6989 ± 0.0051
Netflix	NDCG	0.6081	0.6204	
	Regression	0.6082	0.6287	

Table 5.5: Results for the weak generalization experiments. We report the NDCG@10 accuracy for various numbers of training ratings used per user. For most results, we report the mean over ten runs and the standard deviation. We also report the p-values for the best vs. second best score.

	Method	N=10	N=20	N=50
EachMovie	NDCG	$\mathbf{0.6367 \pm 0.001}$	$\mathbf{0.6619 \pm 0.0022}$	$\mathbf{0.6771 \pm 0.0019}$
	GPR	0.4558 ± 0.015	0.4849 ± 0.0066	0.5375 ± 0.0089
	CGPR	0.5734 ± 0.014	0.5989 ± 0.0118	0.6341 ± 0.0114
	GPOR	0.3692 ± 0.002	0.3678 ± 0.0030	0.3663 ± 0.0024
	CGPOR	0.3789 ± 0.011	0.3781 ± 0.0056	0.3774 ± 0.0041
	MMMF	0.4746 ± 0.034	0.4786 ± 0.0139	0.5478 ± 0.0211
MovieLens	NDCG	$\mathbf{0.6237 \pm 0.0241}$	$\mathbf{0.6711 \pm 0.0065}$	0.6455 ± 0.0103
	GPR	0.4937 ± 0.0108	0.5020 ± 0.0089	0.5088 ± 0.0141
	CGPR	0.5101 ± 0.0081	0.5249 ± 0.0073	0.5438 ± 0.0063
	GPOR	0.4988 ± 0.0035	0.5004 ± 0.0046	0.5011 ± 0.0051
	CGPOR	0.5053 ± 0.0047	0.5089 ± 0.0044	0.5049 ± 0.0035
	MMMF	0.5521 ± 0.0183	0.6133 ± 0.0180	$\mathbf{0.6651 \pm 0.0190}$

Table 5.6: The NGDC@10 accuracy over ten runs and the standard deviation for the strong generalization evaluation.

5.3 Conclusion

To evaluate the matrix factorization model and algorithm described in Chapter 4, it has been applied to well known data sets from the recommender systems domain. It has been shown that the algorithm outperforms the state-of-the-art in Recommender Systems in many instances. Especially noteworthy is the performance in the ranking task, where the presented algorithm is the first one able to directly optimize for that in the Recommender Systems area.

The model extensions have shown their capability to increase the performance of the system. And, as has been noted above, careful parameter optimization in the application domain probably would lead to even better performance with high probability. This is especially true when using the graph kernel which adds an additional set of parameters to be learned.

The variance over the ten runs on different data samples in all experiments is very low, especially given the fact that a non convex function is optimized. The same is true for the variance on the objective function. The low variance may mean that the same local minimum is always reached or that this minimum is indeed a global one. From a user's point of view, this means that the system presents consistent recommendations that are stable against small changes in the data set.

The strong performance when using the per-row loss functions is promising good results for the Machine Teaching scenario. This is even more true for the strong generalization setting. Good performance in this scenario is crucial for the application of any recommender system technology to be applied to the Machine Teaching scenario. The results for the matrix factorization model are an order of magnitude better than those reported for other systems in the literature which makes it especially suitable for the Machine Teaching task.

Next steps in this thesis: Given these promising results, the next chapter will build upon the same algorithm to assess the viability of a Detailed Feedback Machine Teaching system for software engineering.

6 Detailed Feedback Machine Teaching for Software Engineers

Contents

6.1 Introduction

In this chapter, we will apply the algorithm introduced in Chapter 4 to Detailed Feedback Machine Teaching. The goal of a Detailed Feedback Machine Teaching system as defined in Section 2.5 on page 47 is to provide the learner with detailed feedback regarding the artifacts she presents to the system. That feedback shall be based upon a machine learned model of the artifact structure that has been built based upon a database of artifacts created by experts.

The goal of this chapter is to evaluate whether the algorithm presented in Chapter 4 can be used as the core of such a system. This study is presented as follows: Section 6.1.1 introduces the chosen example domain, software engineering, and Section 6.2 presents the learning tools available in that area. In Section 6.3 we will show how a matrix factorization model can be used in this domain. Section 6.4 then presents the evaluation procedure used and the results obtained therewith to assess the suitability of the algorithm to this Detailed Feedback Machine Teaching task.

6.1.1 Application Domain: Programming with Frameworks

We chose to apply the Detailed Feedback Machine Teaching approach and the matrix factorization algorithm to the domain of software engineering with source code as the artifact. We did so for the following reasons.

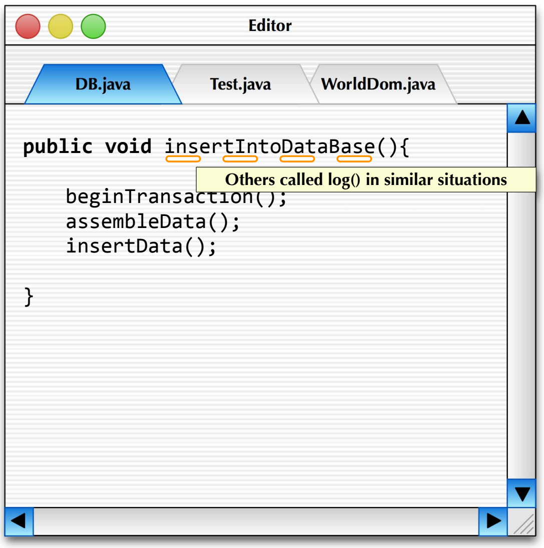
Source code is easy to mine: The source code of a program needs to be parsed by a machine in order to be run on a computer. This prerequisite of source code makes it conceptually easy to analyze it.

Source code contains mine-able structure: Additional, source code needs to be syntactically correct in order to be useful. Therefore, source code is highly structured and thus allows us to assume that its structure can be uncovered through machine learning.

Software Engineering knowledge is often not externalized: Despite the fact the source code is a very structured and formalized artifact, the knowledge that is needed to write it is often not externalized, is comprises not only text book knowledge, but also experience and skill. Nevertheless, this knowledge settles as sediment in the source code.

Within the broad area of software engineering, we investigate the use of software frameworks by programmers. Software frameworks such as the SWT user interface framework facilitate reuse not only of the functionality contained within them, but also of usage patterns for that functionality. Reuse is a major goal in software development. Properly implemented, it leads to lowered costs, faster time to market and increased quality. Software frameworks are a major facilitator for software reuse.

Thus, learning to use software frameworks properly is an important task for software engineers. And as the knowledge needed to use a framework is often not externalized, assisting programmers in learning to use them is a suitable application of Detailed



```
public void insertIntoDataBase(){

    beginTransaction();
    assembleData();
    insertData();

}
```

Figure 6.1: Mockup of the user interface of a Detailed Feedback Machine Teaching system for software engineers.

Feedback Machine Teaching. As described above, source code is well structured and therefore rather directly amenable to machine learning methods. Additionally, many examples of proper framework use exist that can be assumed to adhere to the inherent rules of the framework in question. In fact, developers frequently resort to code examples from other developers when learning.

The Machine Teaching instance described in this chapter focuses on a subset of the problem of learning to code with and withing software frameworks: The proper co-occurrence of calls within a context. Prominent examples include opened files or database transactions that need to be closed. Another example is code that creates user interfaces, as there are certain rules inherent to those as well, such as the need to link related check boxes and their label.

Figure 6.1 depicts the vision of such a tool. Given the source code currently present in a method, it recommends additional calls typically co-occurring with the ones already present.

More formally, we expect the Machine Teaching System to recommend additional method calls to a user when presented with a *context* in which the recommended calls shall be inserted. This context can be of different granularity, e. g. the method body surrounding the possible insertion of the call. Additionally, the system is provided with data about this context. At the very least, other calls present within the same context are available, while more detailed data is subject to an in-depth feature extraction process. As the aim of this work is broad applicability, the following focuses only on other method calls as available data.

6.2 Related Work

Today, programmers learning to use a software framework can resort to a number of tools and resources. This Section shall give an overview of these and discusses how they support the learning of software framework use.

Manual techniques

First and foremost, software developers rely on manually created resources in their work. Descriptive techniques such as cookbooks, tutorials, examples, and pattern languages document how to use a framework to accomplish a specific task [GM96, Joh92].

Other work is directed to the documentation of a framework's architecture, so that users understand the rationale behind the design [BJ94] while there also have been efforts to standardize the framework documentation [BKM00]. In [HHG90], techniques that focus on the documentation of interactions between specific sets of collaborating classes are introduced.

Finally, frameworks are usually shipped with minimal examples specially written to demonstrate specific functionality of a framework.

Conclusions for Machine Teaching: While all these techniques help in learning to use a software framework, much effort has to be spend to create and maintain corresponding documentation. Hence, often such documentation does not exist or is outdated. In contrast to these approaches, Machine Teaching promises to support the learner with automated feedback during her coding. The underlying models of this feedback can also be updated constantly to provide up-to-date help to the learner.

Example finding tools

Another category of tools enable users to find instantiation examples in code repositories. Several tools treat code like regular text documents and apply information retrieval technology to find code examples. Systems like this include [FN87] and Google Code Search[goo]. Software exploration tools like Sextant[SEHM06] or modern IDEs like Eclipse[Fou09] provide queries to search for code elements based on structural properties. However, such approaches require users to have a precise idea of their information need and to be familiar with the vocabulary used. CodeFinder[Hen91] addresses situations where this is not the case. It enables users to iteratively refine their queries based on the results of previous executions.

Some tools feature implicit queries, i.e., the user is no more required to write a query, but the latter is generated from the current context. For instance, Codebroker[YFR00] uses signature information and comments to actively present similar program elements in its repository. Strathcona[HM05] uses the structural context of the code under development, e. g. , the super type, method calls, or overridden methods, and finds examples with a high similarity to this context.

As shown in[SLB00], source code examples are useful for understanding a framework and the use of existing code bases imposes no additional effort. However, instead of providing the framework specific knowledge only, the developers also have to read irrelevant code, i.e., code specific to a single instantiation, which may complicate the understanding process.

Conclusions for Machine Teaching: Example finding tools can be an important augmentation for a Machine Teaching system. Given a suggestion by the Machine Teaching system, the learner may request examples to justify these suggestions.

Mining-based tools

We will now present related approaches where data mining and machine learning tools have been applied to source code before to analyze their limits.

One problem when using frameworks is to derive the exact sequence of calls needed to obtain an instance of a given framework type. This problem is addressed by tools like Prospector[MXBK05] which traverses an API call graph to find possible method sequences that return an instance of the framework type in question. For building the recommendations, Prospector uses different heuristics like shortest paths in the graph to rate the relevance of the sequences found.

Another problem is to find temporal constraints between method calls. Systems such as Jadet[WZL07] and MAPO[XP06] use sequence mining to find frequent method call sequences. This approach is well suited to mine usage patterns within one framework type. However, recommending framework usage that involve several framework types is unlikely to work well since temporal constraints between calls to these types typically exist in a few cases only.

A related area of research is automated bug detection. Tools like PR-Miner[LZ05a] and Dynamine[LZ05b] use association rule mining to find implicit programming rules in code and classify violations of these rules as potential bugs. FrUiT[BSM06] and CodeWeb[Mic00] use association rule mining to find programming rules in existing code.

Parallel to this thesis, the paper [BMM09] was published which also presents a machine learned approach to code recommendations. However, that approach focusses on recommending additional calls per object as opposed to per context as described here.

Conclusions for Machine Teaching: All of these approaches are limited, either to a specific type code such as object instantiation in the case of Prospector or to patterns that are bound to a single type as opposed to interactions between objects of different types.

These interactions between different types are, however, an important aspect of framework use. And as we shall see in the next section, patterns of those can be found by a matrix factorization model.

6.3 Matrix Factorization Modeling

This section describes how to transfer the task of Machine Teaching in the context of framework learning to a matrix factorization problem instance amenable to the treatment presented in Chapter 4. First, the mapping of calling relations in code into a matrix will be shown. The second and final step in building the matrix factorization problem is to choose an appropriate loss function.

Data Representation

The code used for training the system as well as the partial code written by a learner is transferred to a matrix Y by assigning one row of Y for each context.

Here, we distinguish between the method in which the call shall be placed (method-level) and the surrounding class (class-level). Other levels of granularity such as modules, code blocks and user defined code groups are conceivable.

Each column in the input matrix Y then represents a callable method. These calls are then recorded into the matrix Y where $Y_{i,j} = 1$ indicates that a call to method j was found in context i.

That representation is illustrated in Figure 6.2 for the following example:

Example 11. *`class A` is used as the context. In this class, four methods are called. Each of these calls is recorded in the row of Y that corresponds to `class A`. The matrix also shows a hypothetical `class B` that contains only two calls to methods. See Figure 6.2 for a visualization of this mapping.*

Loss Function

Each entry $Y_{i,j}$ encodes whether or not method j has been called in context i. Thus, the data is inherently binary. Candidates for the loss function in that case are the Hinge loss function as well as the logistic regression loss.

However, the positive and negative class are heavily imbalanced: Each class and method only calls a very small percentage of the methods present in the software framework. Thus, the rows of Y encode far fewer calls than “non-calls”. Hence, the weighted variants of these loss functions as introduced in Section 4.4.1 and Section 4.4.1 are used.

6.4 Evaluation Setup

This section provides details on the empirical evaluation conducted on real data from the software engineering domain. The evaluation focuses on the question of how much of the hidden and implicit knowledge embodied in source code a Matrix Factorization model can capture. This is a crucial step towards the actual integration into the software development work flow, e. g. as depicted in Figure 6.1.

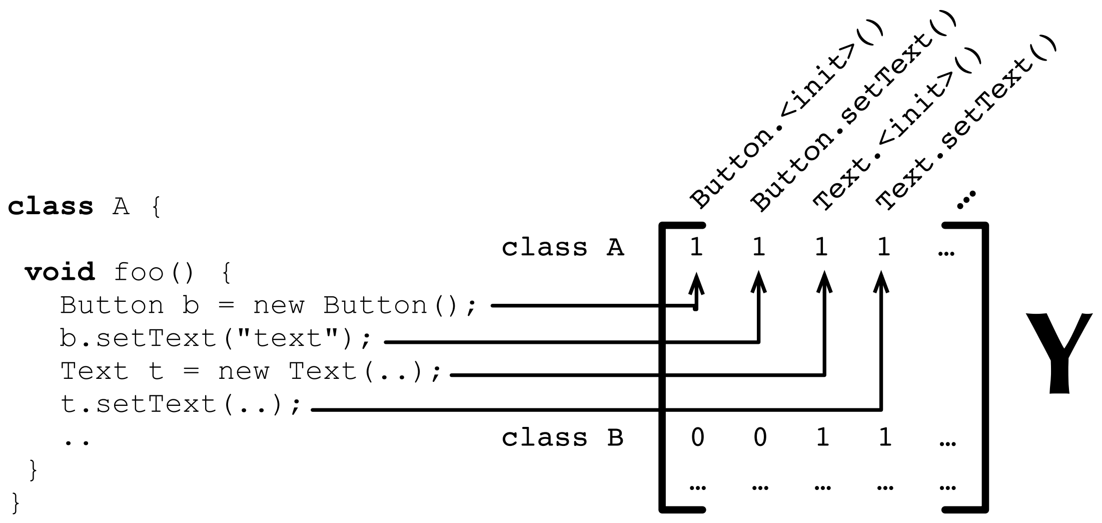


Figure 6.2: Representing call relations in source code as a sparse matrix when using classes as the context.

6.4.1 Method

Evaluation Procedure

The system was evaluated based on the *simulation* of a novice programmer: In each context, half the calls are presented to the system. Then, the system is asked to recommend additional calls. In order to build the initial model, the system is trained on 90 % of the contexts in the data. The resulting matrix C is then used according to Section 4.5.3 to predict the remaining calls for a user.

The remaining 10 % of the available data is then used for the actual simulation. The system is presented one context at a time that contains 50 % the actual calls in that context. Then, the objective function (4.8) is optimized for this one row in Y. The prediction F for this row is then evaluated against the known truth. We repeated this procedure 10 times for different random samplings of the train data.

Evaluation measure: The system can be thought of a special case of an information retrieval system: Given the calls already present in a context, it retrieves additional candidates for inclusion in the program code. These candidates may or may not be relevant in the context. Thus, the evaluation of the system can be done by drawing upon well established information retrieval performance measures, namely:

Definition 16. *The* ***precision*** *of the system is the fraction of the recommended calls that are relevant in the context:*

$$precision = \frac{|\{relevant\ calls\}| \cap |\{recommended\ calls\}|}{|\{recommended\ calls\}|}$$

Definition 17. *The **recall** of the system is the fraction of the relevant calls in the context the system was able to suggest:*

$$recall = \frac{|\{relevant\ calls\}| \cap |\{recommended\ calls\}|}{|\{relevant\ calls\}|}$$

It is commonly known in the information retrieval community that there is a trade-off between precision and recall: A system that recommends all possible calls will score a recall of 1.0 while it will not fare well in terms of precision. On the other hand, a system can achieve almost perfect precision by recommending only very few, but very likely calls. This problem is addressed by the so called F-measure:

Definition 18. *The **F-score** of a system is the weighted mean of its precision and recall:*

$$F_\beta = \frac{(1+\beta^2) * precision * recall}{\beta^2 * precision + recall}$$

Here, we focus $\beta = 1$ which is commonly referred to as the $F1$ measure. It is computed as the harmonic mean between precision and recall:

$$F1 = \frac{2 * precision * recall}{precision + recall}$$

6.4.2 Data Set

The system is evaluated on data gathered from the source code of the Eclipse Integrated Development Environment. To be more specific, the evaluation focuses on calls from the Eclipse Code to the Standard Widget Toolkit (SWT), the User Interface (UI) Framework used in Eclipse. This scenario was chosen for the following reasons:

- UI frameworks are amongst the most commonly used.
- Typically, the use of these frameworks involves several method calls on different framework objects. When creating graphical user interfaces, a large number of framework objects like text fields, buttons and labels collaborate with each other. This means that there are mine-able usage patterns for SWT.
- SWT is developed for Eclipse. Hence, we can assume that the calls from Eclipse to SWT follow the proper usage patterns for SWT.
- Eclipse is a sufficiently large code base to perform meaningful analysis on.
- SWT is used in many more projects and thus training a recommender on the Eclipse code base could be beneficial to a greater target audience of developers.

Data extraction:

As mentioned above, we perform experiments for two possible contexts of recommendation: Classes and Methods. To extract the calls from each method and class in Eclipse to SWT, the WALA toolkit[1] was used. WALA is an open source software package from IBM's T. J. Watson research center which provides functions for the static analysis of code running on the Java Virtual Machine.

Data characteristics:

We found 1, 557 methods in SWT that were called from Eclipse. We found 5, 642 methods in Eclipse that called at least one method in SWT and 2, 733 classes in Eclipse that do so. The method data contains 52, 895 calls, for the class data we recorded 41, 369 calls.

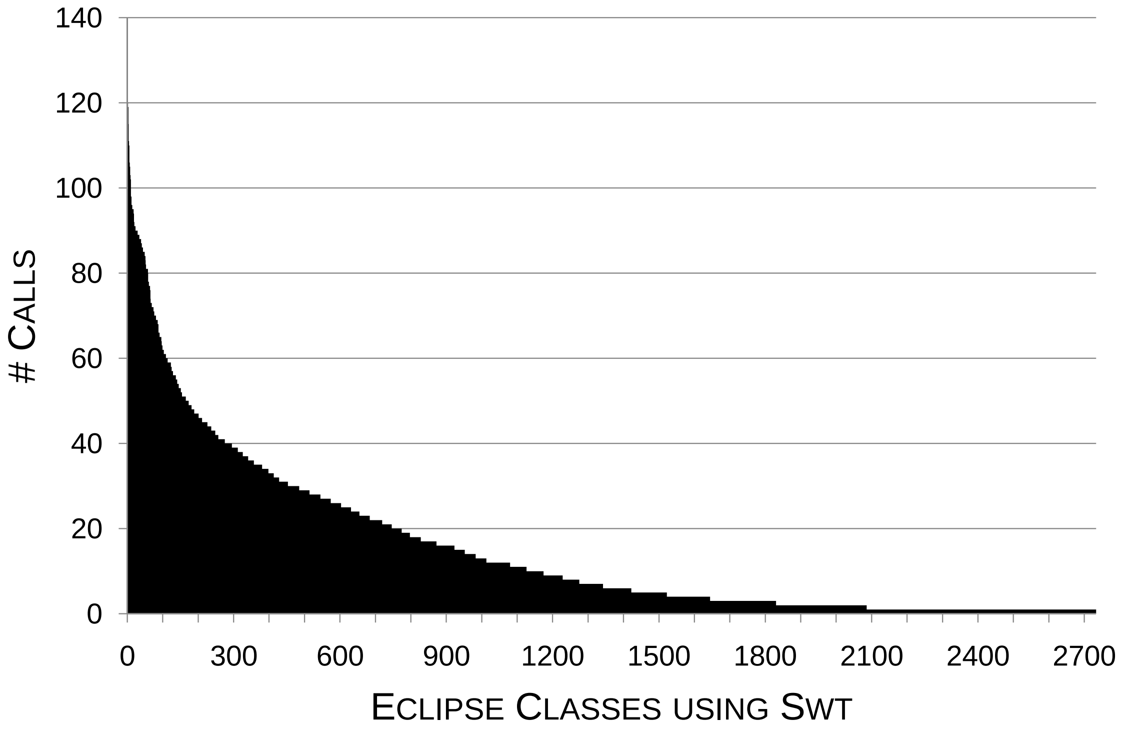


Figure 6.3: Histogram of the number of calls to SWT per class in Eclipse

Figure 6.3 shows the distribution of the number of calls per class. The data for the method context shows the same long tail distribution: We observe both a few classes with many calls to SWT and many classes with only a few calls to SWT. 24% of the classes in Eclipse call only a single SWT method and two thirds of the classes invoke

[1]http://wala.sf.net

less than 15 SWT methods. These properties can be attributed to the way modern object oriented software is constructed which favors small components with small responsibilities. However, there exist more complex classes in Eclipse that use up to 130 different methods from the SWT framework and unless their developers have a precise understanding of the framework their application is unlikely to function properly.

Figure 6.4 shows the call frequency distribution in this data set. The data shows a long tail of methods that have been called very infrequently. On the other hand, 2.5 % of the methods amount to 50 % of all the recorded calls. A qualitative analysis revealed that the more frequently called methods represent the core of the SWT framework such as the UI classes.

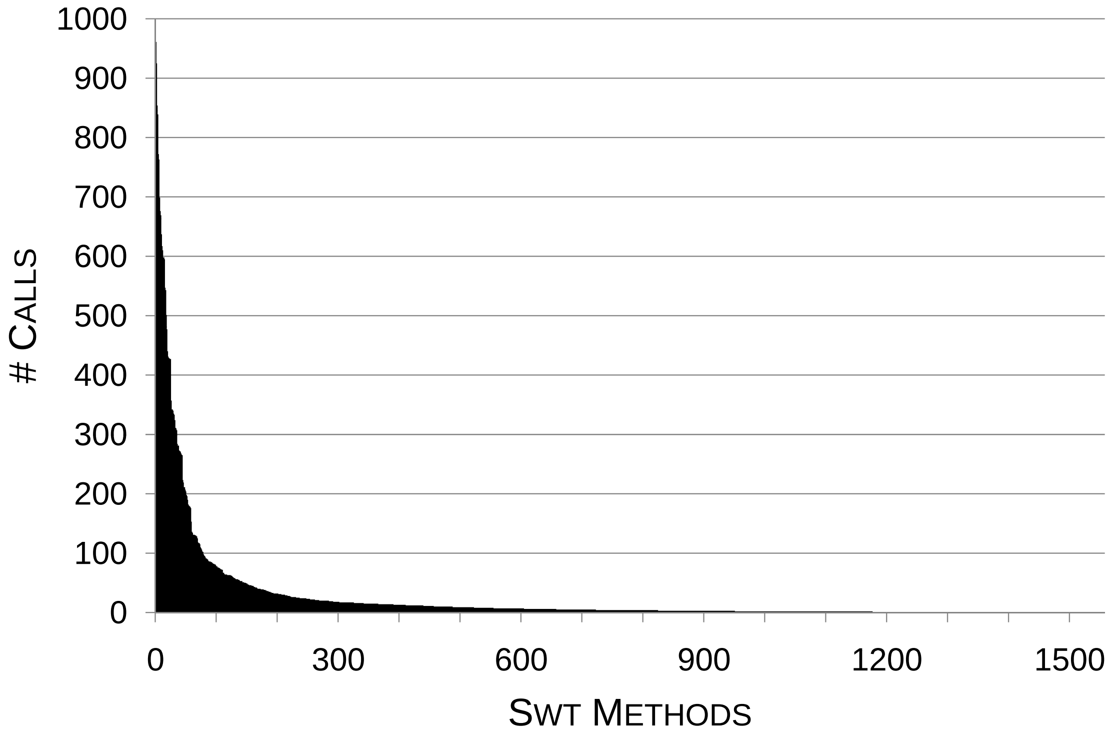


Figure 6.4: Histogram of the number of calls per SWT method

6.4.3 Baseline System

Now that the data set used for the evaluation has been presented, we now present the baseline system implemented. The baseline system serves as a lower bound for the performance to be expected by the matrix factorization algorithm.

Following the literature, a baseline recommender system based on the ideas presented in [BSM06, Mic00] was implemented to compare the matrix factorization approach with. These systems are based on association rule mining.

	Method	λ_r	λ_c	factors	g^+
Class context	Softmargin	10	1	45	10
	Logistic	20	20	30	10
Method context	Softmargin	10	1	30	10
	Logistic	10	10	30	10

Table 6.1: Parameter settings used in the software engineering experiments

The problem of mining association rules is to find all rules $A \rightarrow B$ that associate one set of method calls with another set. Hereby, A is referred to as the *antecedent* and B as the *consequent* of the rule. Two measures for the relevance of a rule exist. The *support* specifies how often the method calls appear together. The *confidence* measures how frequently they conditionally appear, in our case how many rows in the matrix that contain the method calls of the antecedent also contain all method calls of the consequent. To limit the result to most likely meaningful rules, a maximum antecedent size, a minimum support and a minimum confidence is specified for the mining process.

For mining the association rules, a freely available `Apriori` rule miner[2] is used. In order to obtain a reasonable baseline for our approach, we run a set of experiments with different thresholds for minimum confidence and minimum support, namely five minimum confidence values (0.01, 0.2, 0.5, 0.7 & 0.9) and three different absolute support thresholds (5, 20 & 40). Furthermore, we used the rule miner's built-in statistical significance pruning with standard settings. We restricted the antecedent size of the association rules to one. We selected these parameters based on results reported in [BSM06] for the FrUiT code recommender system. Experiments showed that an absolute support threshold of 5 and a minimum confidence threshold of 0.9 lead to the best results according to the $F1$ measure.

6.5 Evaluation Results and Discussion

For both the class and the method context, parameter tuning on one of the ten data splits for optimal $F1$ score was performed. Table 6.1 contains these best parameters found.

Figure 6.5 shows the performance for the two loss functions used in the evaluation as well as the results obtained with the baseline system. Table 6.2 contains the precise numerical results. Note that the unanswered cases, the cases for which the systems don't suggest any call, are generally fewer for the matrix factorization system. This allows the system to help the learning programmer in more cases, a generally desirable property.

[2] `http://www.borgelt.net/software.html`

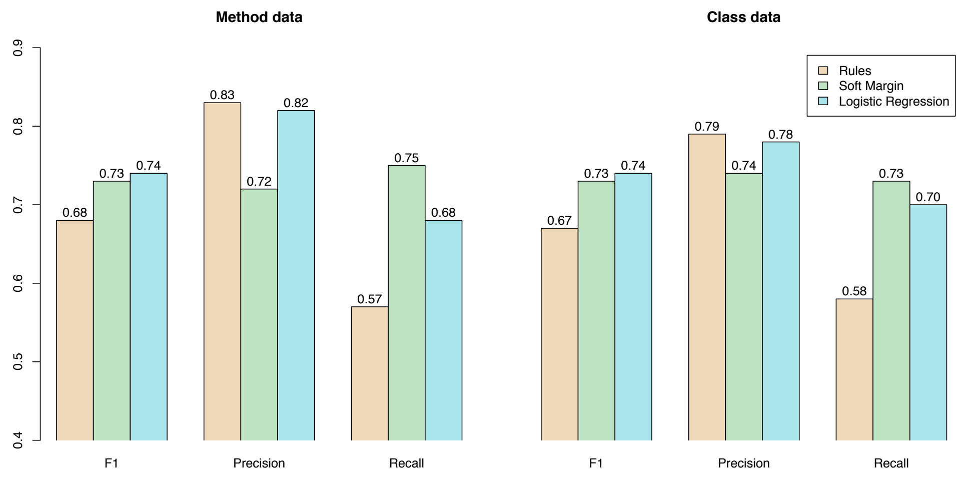


Figure 6.5: $F1$, precision and recall results for the Method and Class data for the rule based approach and matrix factorization using the soft margin and the regression loss functions.

Context	Algorithm	$F1$	Precision	Recall	Unanswered
Method	Rules	0.68 ± 0.01	$\mathbf{0.83 \pm 0.02}$	0.57 ± 0.01	144 ± 11
	Softmargin	$\mathbf{0.73 \pm 0.01}$	0.72 ± 0.02	$\mathbf{0.75 \pm 0.01}$	118 ± 11
	Logistic	$\mathbf{0.74 \pm 0.01}$	$\mathbf{0.82 \pm 0.02}$	0.68 ± 0.02	124 ± 8
Class	Rules	0.67 ± 0.02	$\mathbf{0.79 \pm 0.03}$	0.58 ± 0.02	56 ± 5
	Softmargin	$\mathbf{0.73 \pm 0.02}$	0.74 ± 0.02	$\mathbf{0.73 \pm 0.02}$	48 ± 10
	Logistic	$\mathbf{0.74 \pm 0.02}$	$\mathbf{0.78 \pm 0.02}$	0.70 ± 0.02	50 ± 8

Table 6.2: Averaged results in terms of $F1$, precision and recall over 10 runs. Note that the differences in $F1$ between softmargin and logistic loss as well as the difference in precision between rules and logistic loss are not statistical significant

We observe that the matrix factorization algorithm with a logistic regression loss performs best in terms of the $F1$ score, closely matched by the soft margin loss. Matrix factorization overall performs significantly better than the rule based approach in terms of the $F1$ score. However, both loss functions show very different performance characteristics: For the logistic regression loss, recall and precision differ significantly while the performance of the soft margin loss function is more balanced.

Influence of the weight

Note that precision and recall are very sensitive to the value of g^+, the positive weight. Thus, a desired balance between the two measures can be easily achieved by tweaking this value.

Figure 6.6 depicts the relation between the performance measures and the weight parameter for the matrix factorization system with a soft margin loss. As to be expected, the value of precision decreases as the positive weight increases. The recall behaves in an inverse manner and increases as the weight goes up. The optimal value of the $F1$ is reached for the weight value 10 (2.3 in the log plot). The weight parameter thus provides a convenient way of adjusting the performance of the system to the requirements of the particular situation.

Influence of the number of factors:

Another interesting parameter to study is the number of factors and its impact on system performance. The run-time performance of the system is almost only determined by its value.

Figure 6.7 depicts the values of $F1$, precision and recall for different numbers of factors for the soft margin loss function. We observe that for about 30 factors the system reaches an almost optimal performance while further increases do not affect the performance in terms of the $F1$ score in a significant way.

Runtime performance

Training the initial model takes about 7 minutes for the class context data and 25 minutes for the method context data using the logistic loss function. Based on this model, the system can predict calls in a few milliseconds in the current scenario. This is fast enough to implement the algorithm within an interactive environment.

The main memory needed is the one for C, as R can be discarded after initial training. C for 45 factors needs $1557 * 45 * 8\,\text{Bytes} = 547\,\text{kBytes}$ of memory. This also is easily available on a developer workstation.

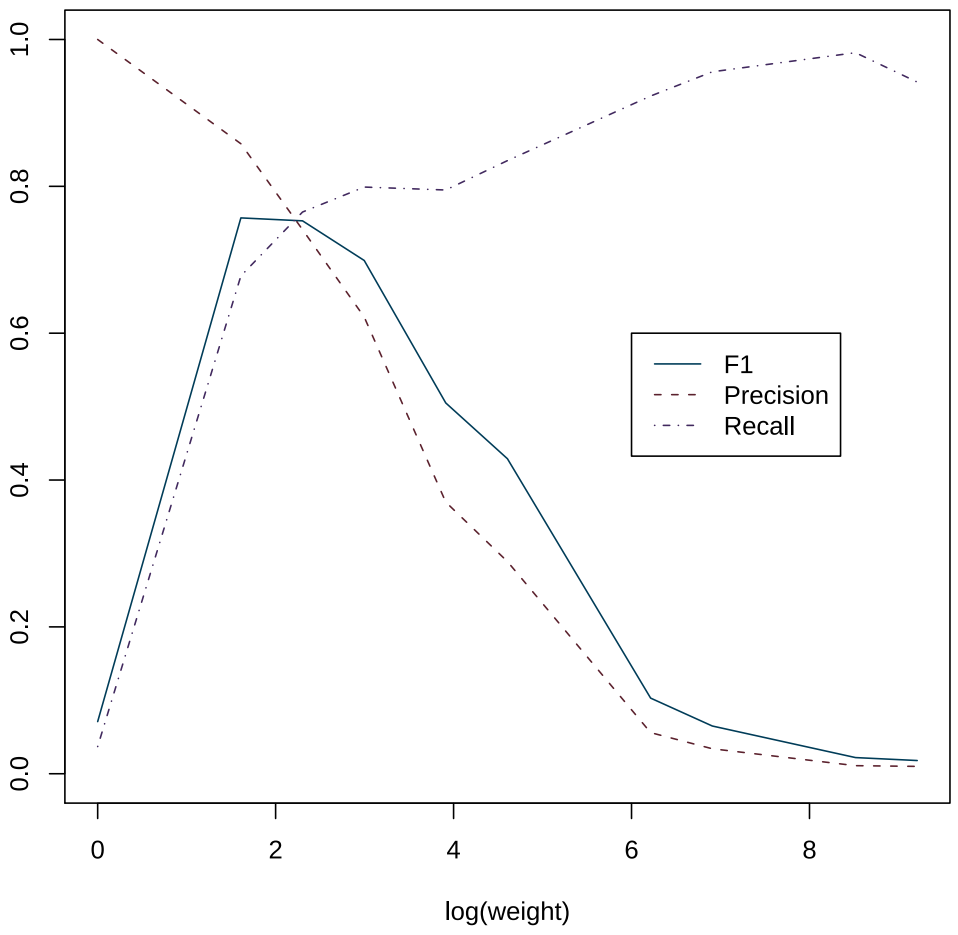


Figure 6.6: *F*1, precision and recall for the matrix factorization system with a soft margin loss for different value of the weight parameter (on a natural log scale)

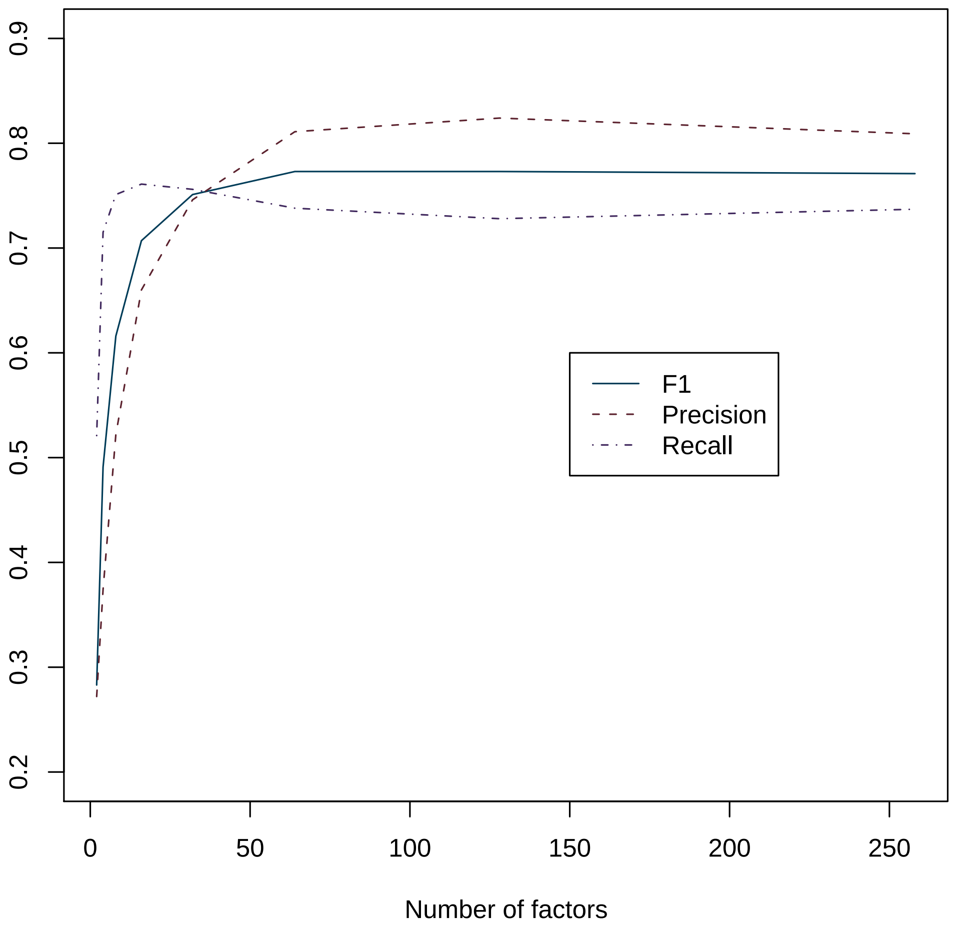


Figure 6.7: Results for $F1$, precision and recall obtained with the matrix factorization system with a soft margin loss and different values for the number of factors parameter

6.6 Conclusion

We presented an application of the Detailed Machine Teaching approach in this chapter. As before, the focus was on the question whether a machine learning model, namely the matrix factorization approach described in Chapter 4 can be applied to the task of generating detailed feedback for learners.

We chose the software engineering domain for this evaluation, as source code is naturally digital and well structured. Both of which render source code analysis suitable for this evaluation. In addition to this easy availability and interpret-ability of the artifacts, the knowledge needed to create them is usually not externalized. These observations make software engineering a suitable target for an evaluation of the Detailed Feedback Machine Teaching approach.

The results, which were obtained based upon a simulated programmer, indicate that the machine learning model is indeed capable of capturing important knowledge from the provided expert code. This is especially noteworthy, as we did not perform any adaptations of the model and algorithm as introduced in Chapter 4: The main design decision was to decide for the structural elements to operate on and for a loss function. Both decisions are inevitable and do not form a special case for the application in this chapter.

Thus, the argument from Section 2.5 on page 47 proofed itself to be true in this example: Even if one *could* design a machine learning model for each kind of artifact, it is not always *necessary* for sufficient performance. Future work should investigate the suitability of the algorithm presented in Chapter 4 to further substantiate that claim.

The results also show empirically that the algorithm meets the requirements set out for a recommender system to be used in Machine Teaching in Section 2.5 on page 49 in this application domain, namely:

R1: Accurate solutions for partial artifacts: As we have described, the system is capable of predicting well given 50 % of structural elements, in this case of the calls in a context.

R2: Interactive Performance: Prediction generally took only a few milliseconds. Obviously, this is fast enough for interactive use of the system, where a response times of up to 0.1 seconds are generally considered to be unnoticeable (see e. g. [Mil68] and [NL06], Chapter 5).

This performance, coupled with the low memory footprint, also facilitates further application of the approach to the multi-framework case where the number of potential callees rises significantly.

R3: Recommendation Coherence: This again follows from the evaluation results: The system achieved high precision scores in the order of 0.75 which indicate that it was indeed capable of providing coherent suggestions to the user. This is especially notable as the system showed a far higher recall than the baseline system.

R4: Easy Adaptation to Multiple Types of Prediction: Through choosing the weighted variants of the logistic and hinge loss functions, we adapted the system to the type or prediction necessary in this case.

Even though specialized models may outperform the application of the rather general matrix factorization framework done here, it is thus safe to conclude that the machine learning aspect of Detailed Feedback Machine Teaching can be provided by matrix factorization models, especially by the one introduced in Chapter 4 of this thesis. The predictions generated are accurate and quickly available, even in the case where changed artifacts occur frequently.

7 Conclusions

7.1 Summary

Problem Statement By its very design, current Technology Enhanced learning systems cannot convey knowledge that is not standardized, structured and most of all externalized. However, important knowledge such as intuition, experience or more generally speaking implicit knowledge does not meet these requirements. Additionally, the structured externalization of knowledge in the form of electronic learning material is often impossible for economic reasons, as its creation is costly.

In traditional education, the apprenticeship with its master-student relationship is used to convey this knowledge. In this setting, the master explains to, discusses with and corrects the learner in a situation centered around the learned activity. This approach is very effective. Yet, it by design requires the availability of a master in the field to the learner.

Approach MACHINE TEACHING is proposed in this thesis to overcome these limitations: We notice that knowledge, even if not standardized, structured and externalized, can still be *observed* through its application. We refer to this observable knowledge as PRACTICED KNOWLEDGE. Machine Teaching is build upon this notion: Machine Learning techniques are used to extract machine models of Practiced Knowledge from observational data. The models thus capture traces of the Practiced Knowledge that went into the artifacts and processes, be it implicit or explicit. These models are then applied in the learner's context for his support.

It is important to note that the people creating the artifacts or following the processes need not to be aware of their role in the system: They are not teachers, they are practitioners. Therefore, Machine Teaching focuses on the *practice*, not the *ideal*, just as the apprenticeship.

While both processes and artifacts can be the anchor in a Machine Teaching scenario, the automatic detection of processes still is an open field of research. As this is a precondition to a successful implementation of a Machine Teaching system, the thesis puts a focus on artifacts.

Levels of Feedback

We identified two important sub-classes of Machine Teaching: General and Detailed Feedback Machine Teaching.

General Feedback Machine Teaching General feedback in a way is like the grades given at school: Given an artifact, the Machine Teaching system determines its quality. This is a straight-forward application of techniques from the field of supervised machine learning and the main effort in building a specific Machine Teaching System lies in deriving appropriate feature extraction procedures. Thus, General Feedback Machine Teaching is readily applicable to a wide range of scenarios where the feature engineering task is either straight-forward or has been studied to great extend. This includes tasks related to text, images and of course artifacts which exhibit feature-vector structure themselves. The application presented in this thesis, learning to write for a specific community, shows that the approach is viable and that the predicted quality of the artifact can in fact be very close to the one perceived by the community.

Detailed Feedback Detailed Feedback Machine Teaching aims at providing feedback regarding the attributes of the artifact to the learner. To do so, the machine learning model needs to have a notion of these artifact and their relations. This, in principle, provokes the creation of a specific machine learning model for each detailed feedback application of Machine Teaching. However, it has been shown in this thesis that Detailed Feedback Machine Teaching often exhibits a striking similarity to the task of a recommender system. Where the latter suggests items to customers, the former "suggests" attributes to artifacts. Despite this obvious similarity, Machine Teaching poses additional requirements onto the underlying algorithm not usually occurring in Recommender Systems. These include a greater variety in the attributes of artifacts, the frequent prediction upon yet unseen artifacts and the real-time nature of a Machine Teaching system. Last but not least, Detailed Feedback Machine Teaching imposes consistency requirements for one artifact as a whole, while a Recommender System is usually focused on predicting the rating a user would associate with a single item.

Matrix Factorization Model and Method This thesis presented a novel model and algorithm based on factor models that substantially generalizes the state-of-the-art in this field to address these requirements uniquely found in Machine Teaching. The algorithm is a valuable contribution to the field of Recommender Systems in its own right as it is the first to be able to predict *rankings* as opposed to *ratings*. Additionally, it forms a hybrid Recommender System that can use known features of the user (artifacts) and items (structural elements) in addition to the collaborative effect present in such data. The base algorithm as well as the extensions thereof proposed in this thesis have shown favorable performance on standard Recommender Systems evaluation data sets available. It is important to note that the algorithm excels especially in what is called the new-user problem in Recommender Systems: Given a new, yet unseen user with some rated items and a trained model based on other users, the goal is to predict well for this new user. While this situation is the *exception* in Recommender Systems, it is the *norm* in Machine Teaching: every time the learner presents the system with an artifact, that artifact constitutes a "new user". Thus, excellent predictive performance on this task is crucial for Machine Teaching.

Machine Teaching for Software Engineering The domain of learning to program well within a software framework is modeled as a task for Detailed Feedback Machine Teaching by forming a caller-callee matrix of representative code. This matrix stores every call to the software framework from the code analyzed, where each row represents one method or class of the analyzed code and each column represents a possibly callable method of the framework. The goal of a Machine Teaching system when confronted with a partial row is to predict missing calls. As we have shown in the empirical evaluation, the model is capable of capturing the structure in this matrix and can therefore compute meaningful feedback for the learning programmer.

This application exemplifies the expressive power of the factor model presented in this thesis, as the model was applied to the problem in a straight-forward manner. So even while purpose-built models are likely to perform even better, the factor model can serve as the underlying model for a wide range of Machine Teaching applications. It therefore provides Detailed Feedback Machine Teaching with a foundation just as supervised machine learning forms the base of General Feedback Machine Teaching.

7.2 Future Work

In this thesis, it was successfully demonstrated that machine learning models can mine artifacts to compute feedback for learners. Future work can build upon this result to address additional challenges of a Machine Teaching System.

Open Questions in Machine Teaching

The User Interface: The feedback computed following the methodology introduced in this thesis needs to be presented to the learner in a useful way. For the software development scenario, we hypothesized that the presentation could be done similarly to that of compiler warning and errors as depicted in Figure 6.1 on page 113.

For many other domains, this question of the user interface is essentially open. Further studies should reveal if and how much the feedback user interface for Machine Teaching differs from traditional systems where the underlying model is designed as opposed to machine learned.

User Studies: Given a user interface, the next step would be a user study. Machine Teaching, especially when used without the distinction between Learners and Experts, may give rise to new or adapted use patterns. One question to study would be whether experts use a Machine Teaching system differently from novices and if both use cases lead to learning.

Feature Engineering for Detailed Feedback: As introduced above, the matrix factorization algorithm can make use of row and column features if they are available. While the software engineering system obtained good results without them, further improvements are possible. In the software engineering domain, these features can be extracted using static code analysis.

Matrix Factorization Improvements

Factor models are an active field of research and the model presented in Chapter 4 provides a solid framework to integrate current results from the field as well as upcoming research directions such as:

Use of Hierarchies: In many instances, the rows or columns of the input matrix form a hierarchy. In Recommender Systems, these hierarchies can be the result of e. g. movie sequels, editions or, for the users, the hierarchy induced by the addresses of the users. Hierarchies almost naturally occur in the software engineering application of the Machine Teaching approach:

- Statements are part of methods; methods are part of classes and they themselves are organized into packages, name spaces or modules.
- Programmers belong to a team which belongs to a department which is part of a larger institution. The coding style and conventions can be assumed to be increasingly detailed from the top to the base of this hierarchy.

Using these and other hierarchies can not only increase the predictive performance of a factor model, it can also allow it to fail in a more grateful manner by suggesting at least a movie from the right genre if failing to suggest the most likely movie. In Machine Teaching, a system could e. g. fail to suggest the right call, yet be able to predict that *a* call on a certain object is missing.

There are currently no systems that can make use of this information in the factor models realm. The topic is, however, in the focus of a track of research in supervised machine learning under the term "feature hierarchies". Further research could investigate the transfer, adaptation and augmentation of these results to factor models.

Fixed Memory Factor Models: Many Recommender Systems are faced with millions of items and users. Another large-scale problem, which Collaborative Filtering has been recently applied to, is advertisement placement, where one essentially deals with as many "users" as the number of websites as described e. g. in [ABC+07]. Storing the matrices R and C in main memory is infeasible for these scenarios.

There is a growing interest in what is called *hash models* in machine learning that compress the model by depending on the collisions of a hash function. Following [WDA+09], this idea can be transferred to factor models by defining a hash function on the rows and columns. Depending on the application, the collisions of this hash function may either be entirely random, or based upon features known about the rows and columns. One can then scale the memory requirement by fixing the hash function to output only a certain number of results to escape the linear scaling of factor models.

Note that this is not the same application of hash functions as in feature hashing as used e. g. by Vowpal Wabbit as described in [LLS07]. In these on-line learning approaches, the number of features in the input domain is reduced to a fixed size in order to gain computational efficiency. In our terms, that would be similar to reducing the number of factors d by means of hashing. In our approach, this is unnecessary as d is a parameter in our case, while it is fixed in normal classification or regression problems where it represents the number of features. In these types of problems that number can be huge, e. g. when dealing with text where the presence of every word is typically treated as a feature.

Online Learning: As introduced above, the factor model is trained in offline mode: Given the data matrix Y, the system finds the model matrices C and R. In many cases, an online training is more appropriate where the model is built from a constant stream of new entries in Y. As we have seen in Section 4.5.3 on page 96, the model can be adapted to new rows easily. The same idea cannot be extended to new columns under the assumptions of Chapter 4, as we explicitly support per-row loss functions.

If one assumes element-wise losses, it is well known that Factor Models can be built in an online fashion using algorithms such as the Stochastic Gradient Descent. Future research is necessary to investigate possible approximations or formulations of the model with per-row losses that facilitate online updates.

Factor Models with Context Aware Biases: Recently, a factor model was proposed for the field of Recommender Systems that explicitly models the temporal effect of a Recommender System by essentially making the row and column biases time dependent in [Kor09]. The reasoning for Recommender Systems is that the preferences of users change over time and that the overall preference changes over time, too, e. g. because of a successful product placement in a national TV show.

As hinted above, this proposal can be integrated into the framework of this thesis by having different column and row biases for different times.

In a broader view, this recent development hints at context-aware factor models. In the example above, time served as the context and as such linked the row and column biases. It would be interesting to extend this concept to features of the matrix elements. In Recommender Systems, that would be features of a rating and the premier example of such a feature is the time at which the rating is given or asked to be predicted. Other useful sources of context for e. g. the movie Recommender System would be the intended audience of the movie as the recommendations in e. g. a family should differ depending on whether a movie is sought for just the parents or the whole family.

In Machine Teaching, such a system opens additional avenues for tailoring the suggestions to the user by using the user features as context. Informally, the system would no longer form predictions like "In other instances, people also did A." to more personalized ones like "In other instances, people *like you* also did A." This raises subsequent questions in the application domain such as how to define a similarity measure on learners and under which circumstances a personalization helps or hinders the learning.

Sequence Prediction: In many Machine Teaching scenarios, ordered entries in the rows occur naturally, e. g. in the software engineering domain where a file needs to be closed *after* it has been opened. One naive way of supporting this is to encode in $Y_{i,j}$ the position at which call j was made in context i. Then, the ordinal regression loss function can be used to build a model that can predict these orderings. However, this approach is rather ad-hoc and is rather impractical for the following reasons:

- There is no principled way to deal with repetitions which frequently occur in real data.
- Such a model can only predict the ordering of the calls, but not the presence of them. This could be solved by yet another ad-hoc fix by building a second model e. g. based on the logistic regression loss.

Thus, a more principled solution to this task is needed. Markov Models probably can serve as an inspiration in finding one. A solution would likely be based on the following idea: The prediction rule of the model needs to be extended such that a prediction $F_{i,j}$ depends not only on R_i and C_j but also a number of those C_k which occurred prior to $Y_{i,j}$ in $Y_{i,*}$. It is plausible that a model around this notion can be build by extending the one presented here.

As we have shown, Machine Teaching opens a wide range of research questions in its application but also poses new challenges to the underlying machine learning models.

Wissenschaftlicher Werdegang des Autors

2007 und 2008: Forschungsaufenthalte in Australien. Mitarbeit in der Gruppe „Statistical Machine Learning" von NICTA, Canberra unter Leitung von Prof. Alex Smola.

2006, 2007 und 2008: Teilnahme an den Machine Learning Summer Schools in Canberra (Australien), Tübingen und Kioloa(Australien).

2005: Abschluss des Diplomstudiengangs Wirtschaftsinformatik an der TU Darmstadt mit Auszeichnung.

2005: Diplomarbeit „Genetic Mineplanning – a genetic algorithm approach to mine planning" bei Prof. Karsten Weihe.

2004: Studienaufenthalt am Centre for Complex Dynamic Systems and Control at The University Of Newcastle (Australien) zur Erstellung der Studienarbeit „Optimizing Flexibility – a new approach to cope with uncertainty in the mining industry and beyond". Betreuer: Prof. Karsten Weihe und Prof. Wolfgang Domschke.

2002: Studienarbeit „Entwicklung eines Mechatronik-Frameworks für das Projekt Läufer" bei Prof. Sorin Huss, TUD.

1999: Aufnahme des Diplomstudiengangs Wirtschaftsinformatik an der TU Darmstadt

1998: Abitur an der Fürst-Johann-Ludwig Schule in Hadamar in den Leistungskursen Mathematik und Physik.

Bibliography

[AB06] Yigal Attali and Jill Burstein. Automated essay scoring with e-rater v.2. *The Journal of Technology, Learning, and Assessment*, 4(3), February 2006.

[ABC$^+$07] Deepak Agarwal, Andrei Z. Broder, Deepayan Chakrabarti, Dejan Diklic, Vanja Josifovski, and Mayssam Sayyadian. Estimating rates of rare events at multiple resolutions. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM Press, 2007.

[ABEV06] Jacob Abernethy, Francis Bach, Theodoros Evgeniou, and Jean-Philippe Vert. Low-rank matrix factorization with attributes. *CoRR*, abs/cs/0611124, 2006.

[AC09] Deepak Agarwal and Bee-Chung Chen. Regression-based latent factor models. In *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 19–28, New York, NY, USA, 2009. ACM.

[AG01] John Robert Anderson and Kevin A. Gluck. *What role do cognitive architectures play in intelligent tutoring systems?*, pages 227–262. Erlbaum, 2001.

[AG09] IM-C AG. Lecturnity. `http://www.lecturnity.com`, August 2009.

[AT05] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, June 2005.

[BH04] Justin Basilico and Thomas Hofmann. Unifying collaborative and content-based filtering. In *Proceedings of the International Conference on Machine Learing*, pages 65–72, New York, NY, 2004. ACM Press.

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, 2006.

[BJ94] Kent Beck and Ralph E. Johnson. Patterns generate architectures. In *Proceedings of the European Conference on Object-Oriented Programming*, pages 139–149. Springer, 1994.

[BKM00] Greg Butler, Rudolf K. Keller, and Hafedh Mili. A framework for framework documentation. *ACM Computing Surveys*, 32(1):15–21, 2000.

[BKV07] Robert Bell, Yehuda Koren, and Chris Volinsky. The bellkor solution to the neflix prize. Technical report, AT&T Labs, 2007. http://www.netflixprize.com/assets/ProgressPrize2007_KorBell.pdf.

[Bla09] Blackboard. Blackboard. `http://www.blackboard.com`, August 2009.

[BMM09] Marcel Bruch, Martin Monperrus, and Mira Mezini. Learning from examples to improve code completion systems. In *Proceedings of 17th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, August 2009.

[BMRC09] Oliver Brdiczka, Jérôme Maisonnasse, Patrick Reignier, and James L. Crowley. Detecting small group activities from multimodal observations. *Applied Intelligence*, 30(1):47–57, 2009.

[Bot04] Léon Bottou. Stochastic learning. In Olivier Bousquet and Ulrike von Luxburg, editors, *Advanced Lectures on Machine Learning*, Lecture Notes in Artificial Intelligence, LNAI 3176, pages 146–168. Springer Verlag, Berlin, 2004.

[BSM06] Marcel Bruch, Thorsten Schäfer, and Mira Mezini. FrUiT: IDE support for framework understanding. In *Proceedings of the OOPSLA Workshop on Eclipse Technology Exchange*, pages 55–59. ACM Press, 2006.

[CB04] Martin Chodorow and Jill Burstein. Beyond essay length: Evaluating e-raters performance on toefl essays. Technical report, ETS, 2004.

[CG05] Wei Chu and Zoubin Ghahramani. Gaussian processes for ordinal regression. *Journal of Machine Learning Research*, 6:1019–1041, 2005.

[Com09a] The Moodle Community. Moodle. `http://moodle.org`, August 2009.

[Com09b] The Sakai Community. Sakai. `http://sakaiproject.org`, August 2009.

[CSS00] Michael Collins, Robert E. Schapire, and Yoram Singer. Logistic regression, AdaBoost and Bregman distances. In *Proc. 13th Annu. Conference on Comput. Learning Theory*, pages 158–169. Morgan Kaufmann, San Francisco, 2000.

[DBSS06] Paul De Bra, David Smits, and Natalia Stash. The design of aha! In *HYPERTEXT '06: Proceedings of the seventeenth conference on Hypertext and hypermedia*, pages 133–134, New York, NY, USA, 2006. ACM.

[DSSS05] Ofer Dekel, Shai Shalev-Shwartz, and Yoram Singer. Smooth ϵ-insensitive regression by loss symmetrization. *Journal of Machine Learning Research*, 6:711–741, 2005.

[EN08] Charles Elkan and Keith Noto. Learning classifiers from only positive and unlabeled data. In *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 2008.

[FN87] W.B. Frakes and B.A. Nejmeh. Software reuse through information retrieval. *SIGIR Forum*, 21(1-2):30–36, 1987.

[Fou09] The Eclipse Foundation. The eclipse ide. `http://www.eclipse.org`, August 2009.

[GM96] Dipayan Gangopadhyay and Subrata Mitra. Design by framework completion. *Automated Software Engineering*, 3(3/4):219–237, 1996.

[GMM$^+$07] Iryna Gurevych, Max Mühlhäuser, Christof Müller, Jürgen Steimle, Markus Weimer, and Torsten Zesch. Darmstadt Knowledge Processing Repository Based on UIMA. In *Proceedings of the First Workshop on Unstructured Information Management Architecture at Biannual Conference of the Society for Computational Linguistics and Language Technology*, Tübingen, Germany, April 2007.

[goo] Google code search. `http://www.google.com/codesearch`.

[Hen91] Scott Henninger. Retrieving software objects in an example-based programming environment. In *Proceedings of the SIGIR International Conference on Research and Development in Information Retrieval*, pages 251–260. ACM Press, 1991.

[HGO00] Ralf Herbrich, Thore Graepel, and Klaus Obermayer. Large margin rank boundaries for ordinal regression. In Alexander J. Smola, P. L. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 115–132, Cambridge, MA, 2000. MIT Press.

[HHG90] Richard Helm, Ian M. Holland, and Dipayan Gangopadhyay. Contracts: Specifying behavioral compositions in object-oriented systems. In *Proceedings of the Conference on Object-Oriented Programming, Systems, Languages, and Applications and the European Conference on Object-Oriented Programming*, pages 169–180. ACM Press, 1990.

[HKV08] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, 2008.

[HM05] Reid Holmes and Gail C. Murphy. Using structural context to recommend source code examples. In *Proceedings of the International Conference on Software Engineering*, pages 117–125. ACM Press, 2005.

[Hul94] David Hull. Improving text retrieval for the routing problem using latent semantic indexing. In *SIGIR '94: Proceedings of the 17th annual international*

ACM SIGIR conference on Research and development in information retrieval, pages 282–291, New York, NY, USA, 1994. Springer-Verlag New York, Inc.

[Joa06] Thorsten Joachims. Training linear SVMs in linear time. In *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 2006.

[Joh92] Ralph E. Johnson. Documenting frameworks using patterns. In *Proceedings of the Conference on Object-oriented Programming, Systems, Languages, and Applications*, pages 63–72. ACM Press, 1992.

[Kor09] Yehuda Koren. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2009.

[KPCP06] Soo-Min Kim, Patrick Pantel, Tim Chklovski, and Marco Penneacchiotti. Automatically assessing review helpfulness. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 423 – 430, Sydney, Australia, July 2006.

[KSF$^+$06] Jihie Kim, Erin Shaw, Donghui Feng, Carole Beal, and Eduard Hovy. Modeling and assessing student activities in on-line discussions. In *Proceedings of the Workshop on Educational Data Mining at the conference of the American Association of Artificial Intelligence (AAAI-06)*, Boston, MA, 2006.

[KSH09] Alexandros Karatzoglou, Alexander J. Smola, and Kurt Hornik. Cran package kernlab 0.9-8 - kernel-based machine learning, 2009.

[KSHZ04] Alexandros Karatzoglou, Alexander J. Smola, Kurt Hornik, and Achim Zeileis. kernlab – an S4 package for kernel methods in R. *Journal of Statistical Software*, 11(9):1–20, 2004.

[KW10] Alexandros Karatzoglou and Markus Weimer. *Collaborative Preference Learning*, chapter (to appear). Springer Verlag, 2010.

[Lea09] Advanced Distributed Learning. Scorm. `http://www.adlnet.gov/Technologies/scorm/`, August 2009.

[LL03] Wee Sun Lee and Bing Liu. Learning with positive and unlabeled examples using weighted logistic regression. In *Proceedings of the 20th International Conference on Machine Learning (ICML 2003)*. AAAI Press, 2003.

[LLS07] John Langford, Lihong Li, and Alex Strehl. Vowpal wabbit online learning project. Technical report, Yahoo! Research, 2007. `http://hunch.net/?p=309`.

[LMA87] Matthew W. Lewis, Robert Milson, and John R. Anderson. The teacher's apprentice: Designing an intelligent authoring system for high school mathematics. pages 269–301, 1987.

[LR04] Cliff Lampe and Paul Resnick. Slash(dot) and burn: Distributed moderation in a large online conversation space. In *Proceedings of ACM CHI 2004 Conference on Human Factors in Computing Systems, Vienna Austria*, pages 543–550, 2004.

[LZ05a] Zhenmin Li and Yuanyuan Zhou. PR-Miner: Automatically extracting implicit programming rules and detecting violations in large software code. In *Proceedings of the European Software Engineering Conference*, pages 306–315. ACM Press, 2005.

[LZ05b] Benjamin Livshits and Thomas Zimmermann. Dynamine: finding common error patterns by mining software revision histories. In *Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 296–305. ACM Press, 2005.

[MGNR06] Edward Meeds, Zoubin Ghahramani, Radford Neal, and Sam Roweis. Modeling dyadic data with binary latent factors. In *Advances in Neural Information Processing Systems 20*, Cambridge, MA, 2006. MIT Press.

[Mic00] Amir Michail. Data mining library reuse patterns using generalized association rules. In *Proceedings of the International Conference on Software Engineering*, pages 167–176. ACM Press, 2000.

[Mil68] R. B. Miller. Response time in man-computer conversational transactions. In *Proceedings of the AFIPS Fall Joint Computer Conference*, volume 33, 1968.

[MSM94] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1994.

[MWK$^+$06] Ingo Mierswa, Michael Wurst, Ralf Klinkenberg, Martin Scholz, and Timm Euler. YALE: Rapid prototyping for complex data mining tasks. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 935–940, New York, NY, USA, 2006. ACM Press.

[MXBK05] David Mandelin, Lin Xu, Rastislav Bodík, and Doug Kimelman. Jungloid mining: helping to navigate the api jungle. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 48–61. ACM Press, 2005.

[NL06] Jakob Nielsen and Hoa Loranger. *Prioritizing web usability*. New Riders, Berkeley, California, 2006.

[NW99] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer, 1999.

[PS09] Rong Pan and Martin Scholz. Mind the gaps: Weighting the unknown in large-scale one-class collaborative filtering. In *Proceedings of the 15th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2009.

[QABC09] LLC Quality Alliance Business Consultants. Power trainer. `http://www.powertrainer.net`, August 2009.

[RS05] Jasson D. M. Rennie and Nathan Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 713–719, New York, NY, USA, 2005. ACM.

[Sch95] Helmut Schmid. Probabilistic Part-of-Speech Tagging Using Decision Trees. In *International Conference on New Methods in Language Processing*, Manchester, UK, 1995.

[SEHM06] Thorsten Schäfer, Michael Eichberg, Michael Haupt, and Mira Mezini. The SEXTANT software exploration tool. *IEEE Transactions on Software Engineering*, 32(9):753–768, 2006.

[SG08] Ajit P. Singh and Geoff J. Gordon. A unified view of matrix factorization models. In *Machine Learning and Knowledge Discovery in Databases, European Conference (ECML/PKDD)*, 2008. ECML/PKDD-2008.

[SJ03] N. Srebro and T. Jaakkola. Weighted low-rank approximations. In *Proceedings of the 20th International Conference on Machine Learning (ICML 2003)*, pages 720 – 727. AAAI Press, 2003.

[SLB00] Forrest Shull, Filippo Lanubile, and Victor R. Basili. Investigating reading techniques for object-oriented framework learning. *IEEE Transactions on Software Engineering*, 26(11):1101–1118, 2000.

[SM08] Ruslan Salakhutdinov and Andriy Mnih. Probabilistic matrix factorization. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, volume 20, pages 1257–1264, Cambridge, MA, 2008. MIT Press.

[SRJ05] Nathan Srebro, Jason D. M. Rennie, and Tommi S. Jaakkola. Maximum-margin matrix factorization. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 1329–1336. MIT Press, Cambridge, MA, 2005.

[SS02] Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.

[SS05] Nathan Srebro and Adi Shraibman. Rank, trace-norm and max-norm. In P. Auer and R. Meir, editors, *Proceedings of the Annual Conference on Computational Learning Theory*, number 3559 in Lecture Notes in Artificial Intelligence, pages 545–560. Springer-Verlag, June 2005.

[SVL08] Alexander J. Smola, S.V.N. Vishwanathan, and Quoc Viet Le. Bundle methods for machine learning. In *Advances in Neural Information Processing Systems 20*, Cambridge, MA, 2008. MIT Press.

[TGK04] B. Taskar, C. Guestrin, and D. Koller. Max-margin Markov networks. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*, pages 25–32, Cambridge, MA, 2004. MIT Press.

[TJHA05] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *Journal Of Machine Learning Research*, 6:1453–1484, 2005.

[TPNT07] G. Takács, I. Pilászy, B. Németh, and D. Tikk. Major components of the gravity recommendation system. *SIGKDD Explorations Newsletter*, 9(2):80–83, 2007.

[Vap95] Vladimir Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, 1995.

[Vap98] Vladimir Vapnik. *Statistical Learning Theory*. John Wiley and Sons, New York, 1998.

[VGS97] Vladimir Vapnik, Steven E. Golowich, and Alexander J. Smola. Support vector method for function approximation, regression estimation, and signal processing. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 281–287, Cambridge, MA, 1997. MIT Press.

[VNC03] Salvatore Valenti, Francesca Neri, and Alessandro Cucchiarelli. An overview of current research on automated essay grading. *Journal of Information Technology Education*, 2:319–329, 2003.

[Voo01] E. Voorhees. Overview of the TRECT 2001 question answering track. In *TREC*, 2001.

[WDA$^+$09] Kilian Weinberger, Anirban Dasgupta, Josh Attenberg, John Langford, and Alexander J. Smola. Feature hashing for large scale multitask learning. In *Proceedings of International Conference on Machine Learning (ICML)*, 2009.

[WF05] Ian H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2 edition, 2005.

[WG07] Markus Weimer and Iryna Gurevych. Predicting the perceived quality of web forum posts. In *Proceedings of the Conference on Recent Advances in Natural Language Processing (RANLP)*, pages 643–648, 2007.

[WGM07] Markus Weimer, Iryna Gurevych, and Max Mühlhäuser. Automatically assessing the post quality in online discussions on software. In *Proceedings of* *the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 125–128, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[WKB09] Markus Weimer, Alexandros Karatzoglou, and Marcel Bruch. Maximum margin code recommendation. In *RecSys '09: Proceedings of the 2009 ACM conference on Recommender systems (to appear)*, 2009.

[WKLS08] Markus Weimer, Alexandros Karatzoglou, Quoc Viet Le, and Alexander J. Smola. Cofi rank - maximum margin matrix factorization for collaborative ranking. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1593–1600. MIT Press, Cambridge, MA, 2008.

[WKS08a] Markus Weimer, Alexandros Karatzoglou, and Alexander J. Smola. Adaptive collaborative filtering. In *RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems*, pages 275–282, New York, NY, USA, 2008. ACM.

[WKS08b] Markus Weimer, Alexandros Karatzoglou, and Alexander J. Smola. Improving maximum margin matrix factorization. In Walter Daelemans, Bart Goethals, and Katharina Morik, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 5211 of *LNAI*, pages 14–14. Springer, 2008.

[WKS08c] Markus Weimer, Alexandros Karatzoglou, and Alexander J. Smola. Improving maximum margin matrix factorization. *Machine Learning*, 72(3):263–276, 2008.

[WZL07] Andrzej Wasylkowski, Andreas Zeller, and Christian Lindig. Detecting object usage anomalies. In *Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 35–44. ACM Press, 2007.

[XP06] Tao Xie and Jian Pei. MAPO: mining api usages from open source repositories. In *Proceedings of the International Workshop on Mining Software Repositories*, pages 54–57. ACM Press, 2006.

[YC07] Kai Yu and Wei Chu. Gaussian process models for link analysis and transfer learning. In *Advances in Neural Information Processing Systems 20*, Cambridge, MA, 2007. MIT Press.

[YFR00] Yunwen Ye, Gerhard Fischer, and Brent Reeves. Integrating active information delivery and reuse repository systems. In *Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 60–68. ACM Press, 2000.

[YVGS08] Jin Yu, S. V. N. Vishwanathan, Simon Günter, and Nicol N. Schraudolph. A quasi-newton approach to non-smooth convex optimization. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, pages 1216–1223, New York, NY, USA, 2008. ACM.

[YYTK06] Shipeng Yu, Kai Yu, Volker Tresp, and Hans-Peter Kriegel. Collaborative ordinal regression. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 1089–1096, New York, NY, USA, 2006. ACM.

[ZDLZ07] Zhongyuan Zhang, C. Ding, Tao Li, and Xiangsun Zhang. Binary matrix factorization with applications. In *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*, pages 391–400, 2007.

[ZWS09] Andreas Zinnen, Christian Wojek, and Bernt Schiele. Multi activity recognition based on bodymodel-derived primitives. In *Proceedings of the 4th International Symposium on Location and Context Awareness (LoCA)*, May 2009.