Predictive Modeling of Relational Data

Dissertation submitted for the degree of

Doctor of Natural Sciences

Presented by

Immanuel Bayer

at the

Faculty of Sciences

Department of Computer and Information Science

Date of the oral examination: 13.09.2017

1. Referent/Referentin: Prof. Dr. Ulrik Brandes

2. Referent/Referentin: Dr. Steffen Rendle

Deutsche Zusammenfassung

In Unternehmen und anderen Organisationen fallen aufgrund der voranschreitenden Digitalisierung von Prozessen täglich grosse Datenmengen an, die durch die stetig fallenden Kosten für das Speichern von Daten vorgehalten werden können. Viele von diesen automatisch gesammelten Daten beschreiben die Interaktion zwischen Objekten, wie beispielsweise elektronische Briefe, die von einer Person zur Anderen gesendet werden oder einer Person, die auf eine Internet[1]-Seite klickt.

Graphen sind ein flexibler Weg diese Daten konzeptionell zu repräsentieren. Zum Beispiel kann der Verkehr von elektronischen Briefen einer Firma durch einen Graphen repräsentiert werden, in welchem Mitarbeiter als Knoten dargestellt sind, die verbunden werden wann immer ein elektronischer Brief gesendet wird. Die Verbindungsstruktur des Graphen zu kennen, erlaubt uns Eigenschaften einzelner Knoten zu schätzen. So kann anhand des elektronischen Briefverkehrs einer Firma die Rolle eines individuellen Mitarbeiters abgeleitet werden ohne Zugriff auf den Briefinhalt zu haben. Die Kenntnis darüber, welche Nachrichten ein Nutzer im Internet konsumiert, ermöglicht Rückschlüsse auf sein Geschlecht, Alter usw.

Merkmale wie Geschlecht oder Alter sind zuweilen jedoch nur teilweise bekannt. Darüber hinaus können auch die Interaktionen zwischen den Objekten unvollständig aufgezeichnet sein. In solchen Situationen können wir ein Modell konzipieren, das die elektronischen Briefaustausche schätzt, die zwar stattfanden, aber nicht in den uns zur Verfügung stehenden Daten registriert sind.

Die oben aufgeführten Beispiele haben dargestellt wie Beziehungen benutzt werden können, um unbeobachtbare oder teilweise beobachtbare Information abzuleiten. Eine Fallstudie hat gezeigt, dass Patentdokumente besser anhand

[1] Ein globaler Verbund von Computern und Computernetzwerken.

ihrer Verweise auf andere Patente kategorisiert werden können, als auf Grundlage ihrer textuellen Beschreibung. Diese und andere vielversprechende Ergebnisse haben zur Entwicklung von Statistisch Relationalem Lernen, einer Unterdisziplin von Künstlicher Intelligenz und Maschinellem Lernen geführt. Eine der fundamentalen Herausforderungen von diesem Feld ist, dass viele Ansätze entweder sehr problemspezifisch oder rechentechnisch zu aufwendig sind, um sie universell anzuwenden.

Modelle, die auf Grundlage von vergangenen Beobachtungen neue Interaktionen vorhersagen, werden heutzutage großflächig eingesetzt, um beispielsweise die Ausspielung von Werbung im Internet festzulegen oder um Buchempfehlungen (Amazon), Musikempfehlungen (Spotify) und Filmempfehlungen (Youtube) anzubieten. Was diese Art von Vorhersageproblemen anspruchsvoll macht, ist die sehr große Anzahl von möglichen Interaktionen. Dies bedeutet, dass auch wenn jedes Paar von Knoten eine valide Interaktion darstellt, nicht jedes Paar explizit bei der Berechnung berücksichtigt werden kann.

Diese Arbeit untersucht wie teilweise beobachtbare Knoten, Merkmale und Interaktionen mit hoher Genauigkeit vervollständigt können - auf rechentechnisch effiziente Art und Weise. Anstatt auf Techniken zurückzugreifen, die auf gemeinsamen Wahrscheinlichkeitsverteilungen oder Annäherungen davon beruhen, wird angenommen, dass Knoten unabhängig sind, wenn Informationen zur Verfügung stehen, die die Graphstruktur beschreiben. Diese Arbeit zeigt, wie man die richtigen Merkmale konstruiert oder automatisch lernt. Diese Merkmale erlauben es uns jede auf die höchste Wahrscheinlichkeit basierende Lernmethode zu nutzen, um unvollständig beobachtete Knoteneigenschaften vorherzusagen.

Um eine flexible Modellierung von Interaktionen zu ermöglichen, wird ein Merkmal-basiertes Modell entwickelt. Die Parameter dieses Modells können in einer Zeitspanne proportional zu der Anzahl der beobachteten Interaktionen gelernt werden. Dies wird durch einen rechentechnischen Trick ermöglicht, der es erlaubt jede mögliche Interaktion exakt zu berücksichtigen ohne sie im einzelnen zu berechnen.

Das Endergebnis ist ein Vorhersagemodell, das effizient zu berechnen ist und eine hohe Vorhersagegenauigkeit beim Schätzen von unbeobachtete Interaktionen erreicht. Darüber hinaus zeigt diese Arbeit wie dieses Modell zur Vorhersage von Knoteneigenschaften verwendet werden kann.

Contents

Part I

Overview

1. Introduction

Data sets collected by businesses and governments grow rapidly — in part because of the declining cost for storing and collecting data. Many of the automatically collected data record interactions between entities such as an e-mail *send* from one person to another or a person interacting with a web site by *clicking* on it. Graphs are a flexible way to represent this type of data conceptually. As an example, the e-mail flow in a company can be represented by a graph with individual employees as nodes that are linked whenever an e-mail is exchanged. Knowing the link structure of the graph allows us to infer the properties of individual nodes. For instance, the e-mail flow in a company allows us to guess the role of individual employees without having access to the e-mail content. Knowing which news articles a user read on the web could allow us to estimate his gender, age, and so on.

Attributes such as gender or age are sometimes only partially observable, and records of interactions can be incomplete. In such situations we can, for example, build a model to predict e-mail exchanges that have occurred, but are not included in the snapshot available to us.

The examples above have illustrated how relations can be used to estimate unobservable or partially observable information. A case study (Chakrabarti, Dom, and Indyk, 1998) showed that a patent document could be better categorized based on the hyperlinks linking it with other patents than by its textual representation. This and other promising results led to the development of Statistical Relational Learning, a sub-discipline of artificial intelligence and machine learning. One of the key challenges of this field is that many models are either very problem specific or too computationally expensive to apply them widely.

Models to predict new interactions based on past observations are widely deployed to position advertisements on the web and to provide book (Amazon),

music (Spotify) and video (YouTube) recommendations. What makes this problem challenging is the very large number of possible interactions. This number is especially large compared to the number of interactions that are actually observed (the interactions form a sparse graph). This means that even though each node pair is a valid interaction, explicitly considering each pair when estimating the model parameter is computationally not possible.

This thesis investigates how partially observed node attributes and interactions can be completed with high predictive accuracy in a computationally efficient way.

Instead of resorting to expensive joint probability estimation techniques or an approximation thereof, nodes are assumed to be conditionally independent given features that describe the graph structure. This thesis demonstrates how to construct or automatically learn the right features, features that allow us to use every maximum-likelihood based machine learning method to predict partially observed node attributes.

In order to allow flexible modeling of interactions, we have developed a new feature-based model. The parameter of this model can be learned in a time proportional to the number of observed interactions. This is possible through a computational trick that allows each possible interaction to be modelled exactly, without having to compute them individually.

The final result is a predictive model that is efficient to compute and achieves high predictive accuracy in estimating unobserved interactions. Moreover, this thesis shows how this model can be used to predict node attributes as well.

1.1. Overview

The following section provides a high-level overview of the organization and main results of this thesis.

Chapter 2: Related Work

We will start with a summary of the general concepts and ideas that are needed as background for later chapters. This includes a literature review with an

emphasis on the field of Statistical Relational Learning and Recommender Systems.

Chapter 3: Problem Statement

The problem of predicting node attributes and interactions is formally introduced as *link prediction* and *relational classification*. These two problems will guide the development and evaluation of the models presented in parts II and III.

Part II: Link Prediction

In this part, we first develop a sequential link prediction model which is trained with a pair-wise loss. Based on these results, the Implicit Factorization Machine model is developed, and we derive a new coordinate descent solver to train this model. The model is easier to train than the pair-wise approach for complicated modeling scenarios such as link prediction for anonymous nodes.

Chapter 4: A Factor Model for Sequential Link Prediction

In this chapter we investigate how to include time-dependence in link prediction models. Our evaluation shows that including time can be very important for predictive accuracy. We sub-sample all possible interaction pairs (Section 2.3.2) while learning the model. We use a Factorization Machine (Section 2.4) to model a factorized Markov Chain. The model predictions have been submitted to a public machine learning competition (ECML PKDD Discovery Challenge 2013). Our sequential model achieved top results on which we will build in the following chapter. The content of this chapter is based on a previous publication (Bayer and Rendle, 2013).

Chapter 5: Implicit Factorization Machines

In this chapter we describe the development of the Implicit Factorization Machine (IFM) model and a coordinate descent algorithm to optimize it

efficiently. This coordinate descent solver allows us to model each possible interaction exactly, without having to compute them individually. This means that we don't have to use random sampling of node pairs as described in the previous chapter. The content of this chapter is based on a previous publication (Bayer, He, et al., 2017).

Chapter 6: Link Prediction using Node Attributes

The IFM model allows us to deal with challenges such as predicting links for previously unseen nodes without retraining the model. This is not possible for matrix factorization models that have been suggested for link prediction before. We evaluate this capability with experiments on a sample of the YouTube view graph in the next chapter. The content of this chapter is based on a previous publication (Bayer, He, et al., 2017).

Part III: Relational Classification

This part shows how to construct relational features that can be used with every maximum likelihood based machine learning model to achieve state-of-the-art performance for relational classification problems. We then show how relational features can be learned automatically with the IFM model.

Chapter 7: Graph Based Relational Features

We introduce new *relational features* for standard machine learning methods by extracting information from *direct* and *indirect relations*. Using three standard benchmark data sets, we show empirically that our relational features yield relational classification results comparable to state-of-the-art Structural Relational Learning methods. Finally, we show that our proposal outperforms these methods when additional information is available. The content of this chapter is based on a previous publication (Bayer, Nagel, and Rendle, 2015).

Chapter 8: Learning Latent Relational Features

This chapter introduces a new relational classification approach that is based on the IFM model. The approach can be easily combined with the graph based relational features that were introduced in Chapter 7. We further introduce a random search-based algorithm to automate the hyper-parameter optimization. The performance of the latent features is investigated in detail and compared against the manually constructed features.

Part IV: *fast*FM: a Factorization Machine Library

This chapter describe a Factorization Machine library that has been developed for this thesis and has been released as an open source project. The library (*fast*FM) provides easy access to many solvers and supports regression, classification, and ranking tasks. This implementation simplifies the use of FM for a wide range of applications and allows the understanding of the FM model by interactive investigations to be improved. The analysis of Bayesian Factorization Machines in particular is simplified by access to the sampling process from a high-level language. The content of this chapter is based on a previous publication (Bayer, 2016).

Part V: Contributions

This chapter summarizes the solutions developed in this thesis for the link prediction and relational classification problem as defined in Section 3. We further point out how the contributions can be used in a more general context where appropriate. We conclude with suggestions for research questions that can be investigated based on the contributions developed in this thesis.

1.2. Published Work

The following peer reviewed articles have been published and form the basis of this thesis.

- Immanuel Bayer and Steffen Rendle (2013). "Factor models for recommending given names." In: *Proceedings of the ECML-PKDD Discovery Challenge Workshop 2013*, pp. 81–89

- Immanuel Bayer, Uwe Nagel, and Steffen Rendle (2015). "Graph Based Relational Features for Collective Classification." In: *Proceedings of the 19th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2015)*. Springer, pp. 447–458

- Immanuel Bayer (2016). "fastFM: A Library for Factorization Machines." In: *Journal of Machine Learning Research* 17.184, pp. 1–5

- Immanuel Bayer, Xiangnan He, et al. (2017). "A Generic Coordinate Descent Framework for Learning from Implicit Feedback." In: *WWW '17: Proceedings of the 26th international conference on World Wide Web*. Perth, Australia: ACM, pp. 811–820

2. Related Work

This chapter provides an overview of research topics related to this thesis. First, three research communities are introduced that are focused on the statistical modeling of relational data (Chapter 2.1).

This is followed by a review of two applications: relational classification and link prediction applications; they are well suited to analyze the fundamental challenges and properties that relational data provide for predictive modeling (Section 2.2).

A more detailed overview of common recommender system approaches is then presented in Section 2.3 with a focus on matrix factorization and the similarities between link prediction (Section 2.2.1) and implicit feedback (Section 2.3.2).

The final Section 2.4 is devoted to the Factorization Machine Model. This model generalizes from many latent factor models and is the basis for the relational model introduced in Chapter 5.

2.1. Relational Modeling

This section introduces three different research communities that focus on modeling relational data and provides a brief summary of each community.

2.1.1. Statistical Relational Learning

Statistical Relational Learning (SRL) (Getoor and Taskar, 2007) is a sub-discipline of artificial intelligence and machine learning that deals with data that includes relational structures. The focus is on method development for parameter inference, reasoning, and knowledge representation.

SRL focuses on a wide range of problems that all include some relational information between objects. The main problems it looks at include the following:

- Relational Classification
- Link Prediction
- Link-based Clustering
- Entity Resolution

The range of approaches to these problems ranges from Collective Classification (Sen et al., 2008) for relational classification to Markov Logic (Richardson and Domingos, 2006), which is a very generic model that aims to provide a unified solution for all canonical problems. More information about SRL can be found in Getoor and Taskar, 2007; Džeroski, 2009; De Raedt, 2008.

2.1.2. Network Science

Network Science is a research community with broad interests including the "study of the collection, management, analysis, interpretation and presentation of relational data" (Brandes et al., 2013, p. 2). Models are often developed based on graph theory, statistics, and social structure. Popular topics include the study of complex networks, the understanding of network formation processes, and network visualization. Historically, network science has been strongly influenced by research questions from social psychology, sociology, and anthropology.

2.1.3. Recommender Systems

Recommender Systems is a field with a strong focus on applied research and is closely related to the data mining and machine learning communities. The standard task is to recommend items to users based on their individual and collective past behaviors. Many problems are directly motivated by industrial applications such as movie, book, hotel, music recommendations that are deployed as web- based services (Linden, Smith, and York, 2003; Baluja et al., 2008; Davidson et al., 2010). These applications require highly accurate predictions, as predictive accuracy often directly impacts revenue. Efficient

algorithms are required to deal with what is often a very large number of users and products.

Many recommender systems can be characterized as either *collaborative* or *content-based*, or they can be a combination of the two. Collaborative filtering approaches (Koren, Bell, and Volinsky, 2009; Koren and Bell, 2011) use past user behavior as the basis for future recommendations, while pure content-based approaches (Lops, De Gemmis, and Semeraro, 2011) rely only on known static attributes of user and item. Context aware collaborative filtering is a popular way to combine the two modeling approaches. What follows is an overview of popular collaborative filtering methods (Section 2.3), followed by a detailed introduction to Factorization Machines, a general model for context-aware collaborative filtering (Section 2.4).

Recommender Systems are closely related to the link prediction problem (Section 2.2.1). Predicting future user-behavior based on past behavior can be seen as predicting links in a bipartite user-item interaction graph. This connection has been used in the literature to utilize link prediction approaches (Palmisano, Tuzhilin, and Gorgoglione, 2008) in recommendation settings and vice versa (Menon and Elkan, 2011).

2.2. Applications

This section provides an overview of the state-of-the-art approaches for two relational problems: link prediction and relational classification. These problems can be seen as the relational equivalent of the standard machine learning problems: classification and (ordinal) regression.

2.2.1. Link Prediction

Link prediction is the problem of predicting whether two nodes in a graph are connected or will connect in the future. The following sections introduce three different research directions addressing this problem.

Similarity Indices

Link prediction approaches can be divided into *unsupervised* and *supervised* methods. Unsupervised methods calculate a *similarity score* $S_{u,v}$ between nodes u and v based on the network structure and can be classified into *local* indices, *global* indices and *quasi-local* indices (Lü and T. Zhou, 2011). The local indices include only adjacent nodes u and v in the calculation, while the global indices score can require paths or walks over the whole graph. Quasi-local indices are extensions of local indices that include not only adjacent nodes, but also non-adjacent nodes. A wide range of similarity indices have been proposed. (An extensive comparisons of local indices can be found in Liben-Nowell and Kleinberg, 2003 and in, T. Zhou, Lü, and Y.-C. Zhang, 2009).

Graph Feature Models

Link prediction can also be perceived as a binary supervised classification problem in which each node pair is either present or not (Al Hasan et al., 2006; Lichtenwalter, Lussier, and Chawla, 2010; Menon and Elkan, 2011). Each node pair is represented by a set of features that are derived from the graph structure and can include similarity indices. Lichtenwalter, Lussier, and Chawla, 2010 have compared the prediction accuracy of similarity indices and a graph-features-based supervised model. They conclude that *"in general, the application of unsupervised methods, at least without due study and consideration, is highly sub-optimal"(emphasis mine, p. 9)*. Their conclusion is based on the observation that supervised methods are better able to adjust to varying characteristics in distinct parts of a graph or across time. However, supervised methods have a major drawback: The number of training samples is quadratic in terms of the number of nodes in the graph. A straight-forward application of binary classification is therefore only possible for graphs with a small number of nodes and leads to a very imbalanced problem for sparse graphs as, for example, when very few node pairs form a link and the vast majority of pairs belong to the class of non-connected pairs. The high-class imbalance has been approached by sub-sampling non-connected node pairs (Al Hasan et al., 2006), by restricting maximal distance between non-connected node pairs (Lichtenwalter, Lussier, and Chawla, 2010), and by optimizing a pairwise loss (Menon and Elkan, 2011).

Latent Feature Models

A basic latent feature model for relational data is the *stochastic block model* (Nowicki and Snijders, 2001). This is a generative model that groups nodes into blocks that connect to the same blocks. Note that nodes in the same group do not need to be connected. The *mixed membership stochastic block model* (Airoldi et al., 2008) extends this model so that a node can belong to multiple blocks. Matrix factorization methods are another class of latent feature models that are well suited for link prediction. The close connection between matrix factorization for collaborative filtering and link prediction is discussed in Menon and Elkan, 2011. The next section provides a review of common collaborative filtering techniques so that later chapters can build upon this connection.

2.2.2. Relational Classification

Relational Classification is similar to standard classification, except that the observations are related. The following sections describe two two main research directions that address the challenge that related observations provide for classification. The first, treats collective classification as a joint inference problem.

Collective Inference

Full relational models such as Markov Logic Networks (MLN Richardson and Domingos, 2006) or the Probabilistic Relational Model (Taskar, Segal, and Koller, 2001) can be used for relational classification (Crane and McDowell, 2012). (See to Sen et al., 2008 for a comprehensive overview of collective inference-based relational classification algorithms.) Their strength in high-label auto-correlation settings and the problem of error propagation has been examined by Jensen, Neville, and Gallagher, 2004 (see also Xiang and Neville, 2011), and improved inference schemes have been proposed (McDowell, Gupta, and Aha, 2009). Recently, stacking of non-relational models has been introduced by Kou and Cohen, 2007 as a fast approximation of Relational Dependency Networks (Neville and Jensen, 2003).

Relational Features

Relational features can be combined with collective inference or directly used with standard ML methods. Models such as Relational Probabilistic Trees (Neville, Jensen, Friedland, et al., 2003), Relational Bayes Classifier (Neville, Jensen, and Gallagher, 2003) and the Link Based Classifier (LBC, (Lu and Getoor, 2003)) concentrate primarily on the aggregation of attributes of connected samples. Others use rows from the (weighted) adjacency matrix as basis for feature construction (Bernstein, Clearwater, and Provost, 2003; Perlich and Provost, 2006; Perlich and Provost, 2003).

2.3. Recommender Systems

This section provides an overview of the recommender systems terminology and introduces popular models that are fundamental to later discussions. Recommender Systems are commonly evaluated based on three criteria that we will refer to throughout the following sections. The three criteria are the following:

- The Quality of the Recommendations.
- Speed and Scalability of the Model.
- Degree of Difficulty to Update the Model with New Data.

The recommender systems in this section can be formulated as a score function $\hat{y}(u, i) : U \times I \rightarrow R$ that assigns a score to each user u and item i pair. The top-N recommendations for user u can be computed by evaluating and sorting the score function $\hat{y}(u, i)$ for all available items $i \in I$. The parameterization of the score function is estimated based on a rating matrix R; the rating of user u for item i is denoted $R_{i,j}$. The set of known ratings I_u for each user u is usually small compared to the number of available items. The rating matrix is therefore only sparsely filled with known values. The set of users U is usually much larger than the set of items I.

Memory-based Recommendations Memory-based models use the rating matrix in order to

1. find similar users
2. compile a list of recommendations based on them.

The two main challenges are therefore to find a good similarity measure $\text{sim}(u, j)$ that quantifies how similar two users are and to find an aggregation method that combines the individual item sets of similar users. The following model has been shown to produce good recommendations and allows the system to include new information easily (Breese, Heckerman, and Kadie, 1998):

$$\hat{y}(u, i) = \bar{r}_u + \kappa \sum_{j=0}^{n} \text{sim}(u, j)(R_{j,i} - \bar{r}_j) \tag{2.1}$$

The mean rating (Equation 2.2) of a user is used twice in the model to reduce user-specific rating value biases.

$$\bar{r}_u = \frac{1}{|I_u|} \sum_{j \in I_u} R_{u,j} \tag{2.2}$$

A disadvantage of this model is that searching for similar users at prediction time is difficult to scale for large user bases.

Model-based Recommendations Model-based approaches use the full data (rating matrix) only to train the model. The recommendations can therefore be performed efficiently, as the model is usually much smaller than the training data. A probabilistic formulation of the memory-based recommendations problem is (Equation 2.3). The ratings range from zero to m.

$$\hat{y}(u, i) = E(R_{u,i}) = \sum_{j=0}^{m} \Pr(R_{u,i} = j | R_{u,k}, k \in I_u) j \tag{2.3}$$

Most machine learning algorithms can be used to model the conditional probability in Equation 2.3. It is therefore not surprising that a wide range of models has been proposed for this task. For example, Breese, Heckerman, and Kadie, 1998 use Bayesian Classifier and Bayesian Network Models.

Item-based Recommendations

Item-based recommender is similar to the previous models except that the user similarity is replaced with item similarity. The typical model consists of two steps:

1. compute the similarity of items,
2. combine item similarities for the basket of items of the active user for new recommendations.

Item-based recommendations scale much better than user-based recommendations for the common situations when the number of items is much lower than the number of users (Deshpande and Karypis, 2004). The performance is often comparable or better than user-based recommendations (Sarwar et al., 2001).

2.3.1. Matrix Factorization

Matrix Factorization (Collaborative Filtering) based recommender systems became very popular after their impressive performance in the Netflix competition (Bennett and Lanning, 2007; Koren, 2009b). The model learns a latent representation, of dimension k, for each user $U_u \in R^{|U| \times k}$ and a latent representation for each item $I_i \in R^{|I| \times k}$. The score for user u and item i is then defined as the scalar product of the corresponding latent representations.

$$\hat{y}(u,i) = \sum_f U_{u,f} V_{i,f} = U_u^T V_j \tag{2.4}$$

The latent user matrices and item matrices are learned by optimizing the following regularized loss function.

$$\underset{U,V}{\operatorname{argmin}} \sum_{u \in U} \sum_{i \in I} \left(R_{ij} - U_i V_j^T \right)^2 + \lambda (\|U_F^2\| + \|V\|_F^2) \tag{2.5}$$

A probabilistic formulation of the model that avoids the need to find suitable weights for the regularization term λ has been introduced by Salakhutdinov and Mnih, 2008.

$$p(R|U,V,\sigma) = \prod_{i=1}^{N} \prod_{j=1}^{M} \left[N(R_{ij}|U_i^T V_j, \sigma^2)\right]_{ij}^{I} \tag{2.6}$$

Context-Aware

The models we have discussed so far rely only on the rating matrix as input and completely ignore the contexts in which the ratings have been collected. Giving an exact definition of context is rather difficult; an extensive attempt can be found in Bazire and Brézillon, 2005. Two important factors influencing context — time and location — are, however, easy to illustrate. Users can, for example, have drastically different reading interests during the workday as opposed to during their free time in the evening (Koren, 2009a). Location information is very important, for example, for restaurant recommendations; restaurants too far away from the current user position are not interesting (M. Ye et al., 2011). Context-aware recommender systems can be classified as

- Contextual prefiltering,
- Contextual postfiltering,
- and Contextual modeling (Adomavicius and Tuzhilin, 2008).

Prefiltering is usually implemented by using only the training data that corresponds to the current context for prediction. *Postfiltering* can be used to conceptualize recommendations from context invariant recommender systems. Pre- and post-filtering approaches are compared in Panniello et al., 2009. Factorization Machines are a general factorization model that can be used for *contextual modeling* (Rendle et al., 2011a). (The Factorization Machine model is introduced in Section 2.4.)

2.3.2. Implicit Feedback

Most recommender systems that we have seen so far require user-submitted ratings. These ratings are *explicit* feedback that is difficult to obtain, as it

requires effort from the user. In contrast *implicit* feedback, as for example observations of user behavior, is often routinely collected in vast quantities, for instance, in server log files. (For an in-depth analysis of the costs and benefits, as well as typical sources for implicit feedback, see Nichols, 1998). Sources for implicit feedback can be grouped into categories (Table 2.1).

A good example for the close relationship between implicit-feedback-based recommendations and link prediction (Section 2.2.1) are citations between documents, listed under the reference category (Table 2.1).

Implicit feedback based recommender systems consequently face the same challenges as Graph Feature Models: the number of training examples is quadratic in the number of nodes ($|U| \cdot |I|$).

Category	**Observable Behavior**
Examination	Selection Duration Edit wear Repetition Purchase (object or subscription)
Retention	Save a reference or save an object (with or without annotation) (with or without organization) Print Delete
Reference	Object→Object (forward, reply, post follow up) Portion→Object (hypertext link, citation) Object→Portion (cut & paste, quotation)

Table 2.1.: Categories for implicit feedback data sources as compiled by Oard and Kim, 1998.

Learning to Rank

The implicit recommendation problem is fundamental to information retrieval and is known as the learning to rank problem. Approaches to optimize recommender systems for a ranking loss can be classified into

- point-wise,
- pair-wise or
- list-wise

approaches according to H. Li, T.-Y. Liu, and Zhai, 2009.

Point-Wise

The point-wise approaches learn a scoring function by considering one sample at a time. Regression, classification, or ordinal regression models can be directly used for this task. A popular weighted alternating least squares (wALS) learning algorithm has been proposed to optimize a point-wise regression model by R. Pan, Y. Zhou, et al., 2008.

$$\underset{U,V}{\operatorname{argmin}} \sum_{ij} W_{ij} \left(R_{ij} - U_i V_j^T \right)^2 + \lambda(\|U_F^2\| + \|V\|_F^2) \tag{2.7}$$

This approach introduces a weight W_{ij} for each example of implicit feedback R_{ij}. Properly choosing the weighting improves performance. Only a sub-sample of negative feedback is used for training as the sum $\sum_{ij}$ would otherwise lead to a cost of $O(|U| \cdot |I|)$ per iteration. Using slightly less flexible weighting schemes, the learning complexity can be reduced to be proportional to the total number of examples of positive feedback $|S^+|$ leading to a complexity of $O(k^2 \cdot |S^+| + k^3 \cdot |U|)$ per iteration (Hu, Koren, and Volinsky, 2008). This allows the use of the full implicit rating matrix, without sub-sampling the negative feedback. Based on these improvements, other weighting schemes have been proposed (R. Pan and Scholz, 2009).

Pair-Wise

A pair-wise loss operates on a set of positive and negative feedback pairs. For each row (user) in the implicit feedback matrix all pairs of observed (nonzero entries) and unobserved items (zero entries) are constructed; together they

form the set of training pairs D_s. The pair-wise loss is then optimized from this set. The following generic optimization criterion

$$\text{BPR} - \text{OPT} := \sum_{(u,i,j) \in D_s} \ln \sigma \left(y(u,i) - y(u,j) \right) + \lambda_\Theta \|\Theta\|^2 \tag{2.8}$$

has been proposed by Rendle, Freudenthaler, Gantner, et al., 2009 for this task. Instead of explicitly constructing the potentially very large set D_s, the pairs are constructed online and optimized with a stochastic gradient descent algorithm. Later, the convergence of these algorithms was improved by using better-informed sampling schemes for constructing the training pairs (Rendle and Freudenthaler, 2014). (For further information on this type of ranking loss, see, for example, Joachims, 2002; H. Li, T.-Y. Liu, and Zhai, 2009).

2.4. Factorization Machines

This section introduces the Factorization Machine (FM) model with an emphasis on two properties: modeling interactions between variables of high cardinality and efficient parameter estimation with coordinate descent.

Standard FM Model

A second-order FM for a feature vector $\boldsymbol{x}$ is defined (Rendle, 2012a) as

$$\hat{y}(\boldsymbol{x}) = w_0 + \sum_{j=1}^{p} w_j x_j + \sum_{j=1}^{p} \sum_{j'=j+1}^{p} x_j \, x_{j'} \langle \boldsymbol{v}_j, \boldsymbol{v}_{j'} \rangle \tag{2.9}$$

with model parameters $\Theta = \{w_0, w_1, \ldots, w_p, v_{1,1}, \ldots, v_{p,k}\}$. The parameters can be grouped into zero-order (offset) $w_0 \in \mathbb{R}$, first-order (linear) $w \in \mathbb{R}^p$, and second-order (pair-wise) interactions $V \in \mathbb{R}^{k \times p}$. The model can be extended to higher order interactions, but here we will focus on the second-order formulation.

2.4.1. Modeling Variable Interaction

The interaction between two variables (i, j) is often modeled by adding an *interaction variable* $x_i x_j$ to the model. This is a viable approach as long as only a few such interactions variables are used. Including more interactions quickly becomes infeasible, as this is equivalent to parameterizing a quadratic form q_A in n variables.

$$q_A(x_1, \ldots, x_n) = \sum_{i=1}^{n} \sum_{i=1}^{n} a_{ij} x_i x_j = a^T A x.$$

The number of parameters $a_{i,j}$ grows quadratically with the number of variables. The number of data points required to fit this form becomes quickly impossible to obtain and process, especially for variables of high carnality. Factorization Machines can express such pair-wise relationships by using a low-rank approximation $V \in \mathbb{R}^{n \times k}$ for the coefficient matrix A. (FMs further exclude the diagonal terms and represent A with a symmetric matrix.)

2.4.2. Efficient Computation

The model formulation (Equation 2.9) can be rewritten without nested sums:

$$\hat{y}(\boldsymbol{x}) = w_0 + \sum_{i=1}^{p} w_i x_i + \frac{1}{2} \sum_{f=1}^{k} \left[\left(\sum_{j=1}^{p} v_{j,f} x_j \right)^2 - \sum_{j=1}^{p} v_{j,f}^2 x_j^2 \right]$$

This formulation shows that FMs can be computed efficiently in $\mathcal{O}(kN_z(\boldsymbol{x}))$, where $N_z(\boldsymbol{x})$ is the number of non-zero entries in $\boldsymbol{x}$. Caching the inner sum is crucial for efficient learning algorithms, as we will see in the next section. FMs are multi-linear in each model parameter $\theta \in \Theta$:

$$\hat{y}(\boldsymbol{x}) = g_\theta(\boldsymbol{x}) + \theta h_\theta(\boldsymbol{x}), \quad \forall \theta \in \Theta \tag{2.10}$$

where g and h are independent of the value of θ. The partial derivatives with respect to the different parameter groups can therefore be written as

$$h_\theta(\boldsymbol{x}) = \frac{\partial \hat{y}(\boldsymbol{x})}{\partial \theta} = \begin{cases} 1, & \text{if } \theta \text{ is } w_0 \\ x_l, & \text{if } \theta \text{ is } w_l \\ x_l \sum_{j \neq l} v_{j,f} x_j, & \text{if } \theta \text{ is } v_{l,f} \end{cases} \tag{2.11}$$

The definition of g_θ is skipped, as we will always use the substitution $g_\theta(\boldsymbol{x}) = \hat{y}(\boldsymbol{x}) - \theta h_\theta(\boldsymbol{x})$ instead.

2.4.3. Coordinate Descent

Coordinate Descent (CD) is a popular algorithm to learn the model parameters Θ of an FM given a set $S = \{(x_i, y_i)\}_{i=1}^N$ of labeled training samples — or $X, \boldsymbol{y}$ in matrix notation. CD iterates over the model parameters Θ and updates one model parameter $\theta \in \Theta$ at a time with

$$\theta^* = \frac{\sum_{(\boldsymbol{x},y) \in S} \left[h_\theta(\boldsymbol{x})(y - \hat{y}(\boldsymbol{x})) + \theta h_\theta(\boldsymbol{x})^2 \right]}{\sum_{(\boldsymbol{x},y) \in S} h_\theta(\boldsymbol{x})^2 + \lambda_\theta} \tag{2.12}$$

Clearly, the update costs are dominated by the two sufficient statistics:

$$\sum_{(\boldsymbol{x},y) \in S} h_\theta(\boldsymbol{x})^2, \qquad \sum_{(\boldsymbol{x},y) \in S} h_\theta(\boldsymbol{x})(y - \hat{y}(\boldsymbol{x})). \tag{2.13}$$

A CD iteration over all training examples and model parameters can be performed efficiently in $\mathcal{O}(k\, N_z(X))$ (Rendle, 2012a). This requires caching predictions $\hat{\boldsymbol{y}} \in \mathbb{R}^{|S|}$ and factor sums $Q \in \mathbb{R}^{|S| \times k}$ for efficient computation of gradients.

$$q_{i,f} := \sum_{j=1}^{p} v_{j,f}\, x_{i,j}. \tag{2.14}$$

Reducing the cost to compute the two sufficient statistics (eq. 2.13) is an active research field. This thesis (Chapter 5) and the work summarized in

Section 2.4.3 demonstrate how assumptions about the input data allow the computation of these statistics to be sped up. Additional caches (eq. 2.14) are often required and will be discussed in detail later. A detailed analysis on how to use CD efficiently for a multi-linear model can be found in Friedman, Hastie, and Tibshirani, 2010. What makes CD especially appealing is that near-linear speedup can be achieved in distributed environments (Rendle, Fetterly, et al., 2016).

Mathematical Properties

A range of publications have examined the mathematical properties of the FM model. FMs of arbitrary order are multi-convex and can therefore be optimized by solving a sequence of convex problems (Blondel, Fujino, Ueda, and Ishihata, 2016). Changing the model equation to

$$\hat{y}(x|w, Z) := w^T x + \sum_{j=1}^{d} \sum_{j'=1}^{d} z_{jj'} x_j x_{j'} = w^T x + \langle Z, xx^T \rangle \tag{2.15}$$

combined with nuclear norm regularization has been shown to be a convex problem (Blondel, Fujino, and Ueda, 2015). A block coordinate descent algorithm has been developed to train FMs with a logistic classification loss (Chin et al., n.d.) for this formulation. A linear time dynamic programming algorithm for arbitrary high order FMs has been developed by Blondel, Fujino, Ueda, and Ishihata, 2016.

Including More Information

FM modifications have been developed in order to include information in the model that can't be easily expressed by feature engineering. Field-aware FMs have achieved top results in machine learning competitions (Juan et al., n.d.). This modification can learn multiple embeddings for a feature. Which embeddings are used to model a specific parameter interaction is an explicit modeling decision. This flexibility comes at the cost of higher run-time complexity $O(N_z(x)^2 \hat{k})$. Using multiple embeddings for the same feature can be seen as a regularization of the standard FM model. Combining FMs for two

different tasks that share the same features has been investigated by Hong, Doumith, and Davison, 2013. An extension of Bayesian FMs (Freudenthaler, Schmidt–Thieme, and Rendle, 2011) has been proposed that uses multiple levels of feature groups (Oentaryo et al., 2014) for regularization. The hierarchy shrinks parameters that belong to the same groups towards each other.

Memory and Runtime Improvements

Choosing the right regularization values for a FM model can be very important for prediction accuracy, but computationally expensive. An algorithm that learns the regularization values together with the model parameter (based on the validation error) has been proposed (Rendle, 2012b). Including infrequent features in a FM model can improve prediction accuracy, but might increase the memory cost dramatically. Coupling the latent dimension of a feature on its frequency allows the memory cost to be reduced without sacrificing prediction accuracy (M. Li et al., 2016). Design matrices that describe relational data often contain sub-matrices that occur repeatedly. A clever algorithm to take advantage of this pattern has been developed for FMs (Rendle, 2013). Reducing the complexity from $O(N_z(B))$ to $N_z(\mathcal{B}) := |\mathcal{B}|n + \sum_{B \in \mathcal{B}} N_z(X^B)$ makes otherwise infeasible feature-based modeling techniques possible for relational data.

3. Problem Statement

This chapter describes the research objective motivating this thesis. The problem is formally introduced in Section 3.1; then the conceptual and experimental evaluation is discussed in Section 7.3.1.

3.1. A Computationally Efficient Prediction Model for Relational Data

This thesis advances scientific knowledge by developing a **computationally efficient prediction model for relational data.** Relational data consists of observations that are connected by some relation. The model combines both the information inherent in the observational attributes and the relation, in order to predict partially observed attributes or relations. In order to ensure that the model is applicable to a wide range of problems, the following requirements have to be satisfied:

Requirement 3.1. The model needs to *scale well*. This means that the runtime is allowed to increase at most linearly with the data size,

- i.e., linearly in the number of entities $|V|$
- and linearly in the number of observed edges $|E|$.

Requirement 3.2. The model should achieve state-of-the-art *predictive performance* under competitive circumstances.

Note that the best performing models are often difficult to interpret. We will therefore, if necessary, trade ability to interpret the model for higher predictive accuracy. A computational model should be applicable to many different problems without requiring fundamental changes to the underlying system. This leads to the next requirement.

Requirement 3.3. Address two canonical relational modeling problems:

- Relational Classification
- Link Prediction

Requirement 3.4. The model needs to allow flexible feature-based adaptation to a wide range of applications. This includes the combination of relational information and object attributes.

3.2. Evaluation on Two Applications

Two fundamental and well-studied relational modeling problems are used to evaluate the predictive performance and modeling flexibility of the model . The next section establishes the notation necessary for the formulation of the two problems. Then, both the relational classification and the link prediction are specified and illustrated by examples. For each problem an evaluation procedure is introduced.

3.2.1. Notation

Definition 3.5. Consider a *directed graph* $G = (V, E, x(t), d)$ with a set of *vertices* or *nodes* V and a set of *edges*, or *links* E where each edge is an ordered pair (u, v) of vertices with the edge pointing from u to v. We further define node *attributes* or *features* $x(t) \colon V \to \mathbb{R}^p$ (we omit the t in $x(t)$ if all features are *time* independent) and *dyad weights* $d \colon V \times V \to \mathbb{R}$. We use D for the set of all possible node pairs or *dyads*. We define E as the set of dyads with non-zero weight $E = D \setminus \{d(u, v) = 0\}$.

3.2.2. The Link Prediction Problem

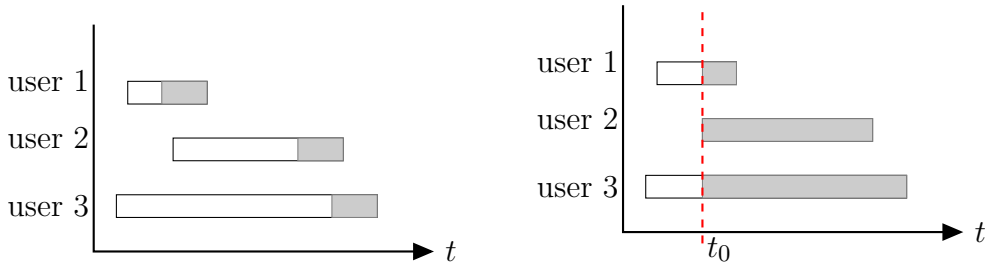
Definition 3.6. Given a graph $G = (V, E, x(t), d)$ we formalize the link prediction problem as learning a scoring function $\hat{y} \colon C \times I \times T \to \mathbb{R}$ which estimates a preference score $\hat{y}(c, i, t)$ for every node pair c and i at time t. The

scoring function can be used to rank nodes i for a node c by their preference, i.e. for node c, node i is preferred over node j iff $\hat{y}(c, i, t) > \hat{y}(c, j, t)$.

Example 3.7. We have log data from a website that helps parents to find a given name for their upcoming baby. We want to support the parents with name recommendations during their search. The log file contains time-stamped interaction data for each user, *e.g.,* entering name into a search field, saving names or requesting background information. We can create a graph representation of the user behavior in the form $G = (V, E, x(t), d)$.

- Nodes V: The set of nodes V represents user and given names (items).
- Edges E: An edge $(u, i) \in E$ between nodes v_u, v_i exists if user v_u clicks on item v_i.
- Node Attributes x: The set of attributes at a node might be different for user and item nodes. The set of user attributes could contain his location, gender, and time dependent information like which item he clicked on last. The set of item attributes could be length of the name and its gender.
- Edge Weight d: The weight $d_{u,v}$ on an edge between the nodes v_u, v_i could depend on how often a user v_u clicked on an item v_i.

Evaluation Protocol Time is a very important aspect for the link prediction problems we consider. We therefore use two different time-based evaluation protocols for this problem. The *offline* protocol (Figure 3.1a) removes the last n links for each node u_i and evaluates how well the model recovers them. The *online* protocol (Figure 3.1b) uses a global time point t_0 to hide all links established afterwards as $t > t_0$ for evaluation.



(a) Offline Evaluation (b) Online Evaluation

3.2.3. The Relational Classification Problem

Definition 3.8. Given a graph $G = (V, E, x, d)$ with a subset of *labeled nodes* $V_l \subset V$, a set of *unlabeled nodes* $V_u = V \setminus V_l$ and a set of m possible labels Y, we define the node classification problem as the task to assign labels $\tilde{y} : V \to Y$ to all unlabeled nodes V_u of the graph.

For clarity, we provide two examples of collective classification problems that we will revisit in later chapters.

Example 3.9. The task is to predict if a movie is going to be successful. (We consider a movie successful if the box-office receipts exceed $2 million on its opening weekend). The data contains labeled movies released in the United States between 1996 and 2001. The data contains an edge list linking movies if they share a production company, producer, director, or actor. A graph representation of the movie data of the form $G = (V, E, x, d)$ can be created in which

- Nodes V: The set of nodes V represents movies.
- Edges E: An edge $(u, i) \in E$ between nodes v_u, v_i tells us that they share a production company, producer, director, or actor.
- Node Attributes x: The set of attributes at a node may include the movie's title, length, genre, and so on.
- Edge Weight d: The weight $d_{u,v}$ on an edge between the nodes v_u, v_i could indicate how many characteristics are shared.

Example 3.10. Assume we have a collection of scientific papers which are labeled by topic. The labels are, however, missing for some of the papers and should be estimated. The collection includes the content for each paper and an edge list linking two papers if one cites the other. The problem can be approached from three fundamentally different directions, each with a different graph representation $G = (V, E, x, d)$. These are

1. Only use the paper content and ignore the citation information. This way the papers are effectively treated as *iid samples*.
    - Nodes V: The set of nodes V represents scientific papers.
    - Edges E: The set of edges $E = \phi$ is empty if we ignore the citations.
    - Node Attributes x: The set of attributes at a node could be the word frequency of all the words in the paper.

- Edge Weight d: The weight $d_{u,v}$ on an edge between the nodes v_u, v_i could always be set to the same constant, as citing a paper is a binary operation.

2. Only use the citation graph and ignore the paper content (*network only*).
   - Nodes V: The set of nodes V represents scientific papers.
   - Edges E: An edge $(u, i) \in E$ between nodes v_u, v_i exists if node v_u cites v_i.
   - Node Attributes x: The set of attributes at a node is restricted to its unique identifier.
   - Edge Weight d: The weight $d_{u,v}$ on an edge between the nodes v_u, v_i could always be set to the same constant, as citing a paper is a binary operation.
3. Combine citation and content data in a *complete model*.
   - Nodes V: The set of nodes V represents scientific papers.
   - Edges E: An edge $(u, i) \in E$ between nodes v_u, v_i exists if node v_u cites v_i.
   - Node Attributes x: The set of attributes at a node could be the word frequency of all of the words in the paper.
   - Edge Weight d: The weight $d_{u,v}$ on an edge between the nodes v_u, v_i could always be set to the same constant, as citing a paper is a binary operation.

Evaluation Protocol The evaluation protocol addresses two relational classification specific issues that can greatly influence the result: the percentage of labeled nodes and the specific nodes which are labeled. We therefore repeat the evaluation by hiding $10, 20, \ldots, 90$ percentage of the node labels in order to investigate the influence of label sparsity in the graph on the model's performance. For each percentage of hidden node labels, the experiment is further repeated ten times to account for the fact that the node-based sampling is ignorant of the edge distribution. The tenfold repetition helps to better estimate the generalization error by reducing the influence of very difficult or easy label constellations.

Algorithm 1 Estimate Relational Classification Performance

```
1:  procedure EVALUATE(G, model)
2:      for p ∈ {10, 20, ..., 90} do
3:          while not repeated 10 times do
4:              V_u ← sample p % of V
5:              G_train ← delete node labels for nodes V_u from G
6:              model.train(G_train)
7:              ŷ_u ← model.predict(G_train)
8:              score ← evaluate(ŷ_u, y_u)
9:              save score
10:         end while
11:     end for
12: end procedure
```

Part II

Link Prediction

We start the development of a new *computationally efficient relational prediction model* by investigating the link prediction as defined in Section 3.2.2 with respect to two challenges, high predictive performance and computational efficiency.

It is paramount for predictive performance that the model is able to express the fundamental properties of the problem domain. We identify time dependence and regular equivalence as the two properties that a general link prediction model need to address.

Time becomes relevant as soon as we can't consider the properties of a node as stationary. For movie recommendation, time dependence could be introduced by people preferring to watch movies (series) in a certain order or when their tastes change over time. We use a recommendation task in Chapter 4 to show that a Factorization Machine can model time dependency through feature engineering.

Two nodes are regular equivalent if they share the same kind of connections to other nodes. This means that modeling regular equivalence is less strict than modeling structural equivalence which requires both nodes to connect to the same nodes. Patterns based on regular equivalence therefore have a better chance to generalize to unseen data. In Chapter 5, we discuss how Implicit Factorization Machines can predict new links by modeling nodes based only on their attributes and not their exact connection to nodes.

In order to develop a model that scales to large data sets, we need to avoid the quadratic complexity that naturally surfaces when we model link prediction as pairs of nodes. Collaborative filtering methods (Section 2.3) achieve this by only considering the actually observed node pairs (edges). However, this approach is not possible for general graphs (or implicit feedback, section 2.3.2), as a simple constant function would solve this optimization problem perfectly if all edge weights were equal.

We approach this challenge first by sub-sampling the non-connected pairs in Chapter 4, an approach which produces good results, but is sensitive to the sampling strategy. We therefore develop the Implicit Factorization Machine model that doesn't require any sampling in Chapter 5.

4. A Factor Model for Sequential Link Prediction

In this chapter, we develop a sequential link prediction model which is trained with a pair-wise loss (Section 2.3.2). We include the sequential information in a Factorization Machine model via feature engineering and use a pair-wise loss to deal with the large number of un-connected node pairs (sparse graph).

We evaluate this model on a link prediction task that requires us to model interactions between visitors of a web-site and its content. The web-site helps new parents find a name for their child by providing background information about given names. The link prediction task is to predict new links between the user and a large set of given names. We have evaluated our model on this task by participating in the ECML PKDD Discovery Challenge 2013 that provided the data and made it possible to compare it objectively against other approaches.

The following discussion describes our contribution to the ECML PKDD Discovery Challenge 2013 (Offline Track) in more detail. The task was framed as recommending given names to a user of the name search engine, "Nameling"(Mitzlaff and Stumme, 2012) based on the user's historical search and clicking behavior. We interpret the problem as a bipartite link prediction problem and use a purely statistical approach, one based on the user history. We didn't use the available meta-data such as word similarity lists or geographic information that have been provided to the users.
We use a factorized personalized Markov Chain model (FPMC) (Rendle, Freudenthaler, and Schmidt-Thieme, 2010) in order to capture the user-specific name preferences. The Bayesian Personalized Ranking (BPR) Optimization Criterion (Rendle, Freudenthaler, Gantner, et al., 2009) is used to learn the latent variables of this model. We complement this factor model with syntactical similarity information by applying prefix smoothing to the name ranking.

4.1. Terminology and Formalization

Let $U = \{u_1, u_2, \ldots\}$ be the set of all users and $N = \{n_1, n_2, \ldots\}$ the set of all names. We further use $T = \mathbb{N}$ to identify user action over time. The problem can be represented by a bipartite graph $G = (V, E, x(t), d)$ with the vertices $V = U \cup V$.

The scoring function

$$\hat{y} \colon U \times T \times N \to \mathbb{R} \tag{4.1}$$

returns the estimated preference of a user U at time T for a name N. To obtain the user specific ranking for every user $u \in U$, each name is scored — i.e. $\hat{y}(u, t, n_1)$, $\hat{y}(u, t, n_2)$, etc. and the names are sorted by their scores. We further define $D \subseteq U \times T \times N$ as the training set of available names which have been observed in the logs.

4.2. Sequential Factor Model

In order to extract training samples from the user activities, we first defined four indicator functions. These functions are then used to encode the training samples as sparse, real-valued features that can be used in a factorization machine. Finally, we give a formal definition of our prefix smoothing approach.

4.2.1. Indicators

Our model assumes that the preference of a user u at time t for a name n can be explained by:

1. The ID of the name: n.
2. The ID of the user: u.
3. The **l**ast name selected by the user: $l \colon U \times T \to N$.
4. The **h**istory of all names selected by the user up to time t: $h \colon U \times T \to \mathcal{P}(N)$.

The indicator functions 2–4 can be used for are a time-dependent user-node description. The first indicator describes the name nodes.

Besides these four indicators, the model should also take into account all interactions between indicators. For example, the interaction between name n and last name $l(u,t)$ would model the effect for choosing name n if the name $l(u,t)$ has been selected before. In total, the first three indicators correspond to a personalized Markov chain (Rendle, Freudenthaler, and Schmidt-Thieme, 2010). The fourth indicator can be seen as a Markov chain with long memory in which all of the history is aggregated into a single set.

4.2.2. Factorization Machine

The number of pairwise interactions between variables is high and cannot be estimated reliably with standard parametrization. Thus, we use a factorized parametrization (Rendle, Freudenthaler, and Schmidt-Thieme, 2010), which allows the model to estimate parameters even in highly sparse data.

The ideas described so far can be realized with a factorization machine (Rendle, 2012a). For this purpose, the four indicator variables are translated into a sparse real-valued feature vector $\mathbf{x} := [x^U, x^I] \in \mathbb{R}^p$ with $p = |N| + |U| + |N| + |N|$ with many predictor variables. The feature vector is a concatenation of the sparse user x^U and item x^I feature representation for which the standard encoding described in (Rendle, 2012a) is used.

For example, for an edge E (u,t,n) let the values of the four indicators be:

1. name to rank: *Anna*,
2. user ID: 0,
3. last name selected by user: *Jana*,
4. history of all names selected by the user up to time t: {*Petra, Annabelle, Maria*}.

This can be encoded as a real valued feature vector of the form

$$\begin{aligned}&\mathbf{x}(u,t,n)\\ &= (\underbrace{0,1,0,\ldots}_{|N|},\underbrace{1,\ldots,0}_{|U|},\underbrace{0,\ldots,1,0,\ldots}_{|N|},\underbrace{0,0.33,\ldots,0.33,\ldots,0.33,\ldots 0}_{|N|}).\end{aligned} \quad (4.2)$$

The factorization machine (FM) model Rendle, 2012a of order $d = 2$ can be applied to the generated feature vector $\mathbf{x}$ and reads

$$\hat{y}^{\text{FM}}(\mathbf{x}(u,t,n)) := w_0 + \sum_{j=1}^{p} w_j x_j + \sum_{j=1}^{p} \sum_{j'=j+1}^{p} x_j x_{j'} \sum_{f=1}^{k} v_{j,f} v_{j',f} \tag{4.3}$$

Here, $w_0 \in \mathbb{R}, \mathbf{w} \in \mathbb{R}^p, V \in \mathbb{R}^{p \times k}$ are the model parameters, $k \in \mathbb{N}$ are the size/ dimensionality of the latent space. Thus, the model has one feature vector $\mathbf{v}_i$ for each variable x_i.

Empirically inspecting the generated recommendations with this model shows (see Table 4.2) that the semantic meaning of names is found; for example if a user searches mainly for female names, female names are recommended; if the user selects typically short names, short ones are recommended, and so on. The reason for the success of the model is that it automatically finds latent features $\mathbf{v}_n \in \mathbb{R}^k$ for each name n which describe its characteristics. Such characteristics could be gender, name length, and so on.

4.2.3. Prefix Smoothing

In general, the proposed model can express any kind of pairwise relation between names[1]. However, with small data sizes, the model might have problems finding relations between infrequent names. As an example, the model can express and will automatically learn that *Anna* and *Anne* are syntactically similar or that *Farid* and *Behrouz* are semantically similar (both Persian masculine names) if the data logs are large enough. However, the size of the observed logs is limited, and the model cannot learn all of these relations reliably — especially not for infrequent names.

To overcome the problem, we inject some syntactical relations manually into the model. We create indicators stating that two names (name n and last name $l(u,t)$) share a prefix of length m – we consider prefix lengths of $m \in \{1,2,3,4,5,6\}$. We add these indicators about syntactical similarity to the FM

[1] Note that semantic or syntactical similarities are also just pairwise relations.

model mentioned above:

$$\hat{y}(u,t,n) := \hat{y}^{\mathrm{FM}}(\mathbf{x}(u,t,n)) + \sum_{m\in\{1,2,3,4,5,6\}} z_m\,\delta(\mathrm{prefix}(n,m) = \mathrm{prefix}(l(u,t),m)), \quad (4.4)$$

where δ is the indicator function — i.e. $\delta(b) = 1$ if b is true — and $\mathrm{prefix}(s,m)$ returns the prefix of string s of length m.

The final model is slightly more complex and also considers syntactical similarity to the next-to-last name:

$$\hat{y}(u,t,n) := \hat{y}^{\mathrm{FM}}(\mathbf{x}(u,t,n)) + \sum_{t'\in\{t,t-1\}} \sum_{m\in\{1,2,3,4,5,6\}} z_{m,t'}\,\delta(\mathrm{prefix}(n,m) = \mathrm{prefix}(l(u,t'),m)). \quad (4.5)$$

Please note that the idea of prefix smoothing, specifically the indicator

$$\delta(\mathrm{prefix}(n,m) = \mathrm{prefix}(l(u,t),m))$$

— can be used directly in the FM by enlarging the feature vector $\mathbf{x}$. Using this representation, the parameters $z_{m,t}$ are part of the $\mathbf{w}$ parameters of the original FM. Extending the prefix smoothing to longer prefixes as well as taking into account names earlier than $t-1$ could further improve the model.

4.3. Learning

The log file describes a sparse graph as only a small subset of all possible user-item interactions that have been observed in the past. Learning our model by optimizing a loss function over all possible node pairs is clearly not possible. Instead, we use a sampling-based approach that optimizes a pair-wise loss function (eq. 4.6). A pair is constructed by drawing an observed user-name (u,n) combination as a positive sample and any other name n that this user hasn't shown interest in as a negative sample.

4.3.1. Optimization Criterion

The model parameters of the FM are learned by discriminating between previously selected and unselected names. This optimization criterion has been proposed for item recommendation as BPR (*Bayesian Personalized Ranking*) (Rendle, Freudenthaler, Gantner, et al., 2009) and has been used in several other recommendation tasks including sequential recommendation (Rendle, Freudenthaler, and Schmidt-Thieme, 2010). For the task of name recommendation, it reads:

$$\mathrm{BPR-OPT} := \sum_{(u,t,n)\in D} \sum_{n_2\in(N\setminus\{n\})} \ln \sigma(\hat{y}(u,t,n) - \hat{y}(u,t,n_2)) - \lambda_\Theta \|\Theta\|^2 \tag{4.6}$$

where $\hat{y}$ is the FM (using the predictor variables encoded in the real-valued vector $\mathbf{x}$) and Θ is a vector containing the model parameters $V, \mathbf{w}, w_0$.

4.3.2. Algorithm

The standard BPR algorithm (Rendle, Freudenthaler, Gantner, et al., 2009) is a stochastic gradient descent (SGD) algorithm. The algorithm samples first a positive observation $(u,t,n) \in D$ and then a negative name n_2 uniformly from $N \setminus \{n\}$. A gradient step is performed on this pairwise comparison. In our implementation, instead of using a uniform distribution for negative names, the names are sampled in an approximately proportional manner to their expected rank.

Even though FM parameters and prefix smoothing could be learned jointly with BPR, for simplicity[2] we learned only the FM parameters with BPR and selected the prefix smoothing parameters (only 12 parameters) manually.

[2]The reason for separating both parts was a practical matter because we reused an existing implementation.

4.3.3. Ensembling Factor Models

The BPR algorithm is a point estimator and returns one ranking. We consider uncertainty in the ranking by running the learning algorithm three times, each time with a different sampling hyperparameter. While training each model, we select[3] 20 iterations in total for which we predict the ranking of each — i.e. we have 20 (slightly) different scores $\hat{y}(u, t, n)$ for each triple (u, t, n). The final scoring function is an unweighted average of the 20 scoring functions.

4.4. Experimental Results

In this section, we first present our scores from the official leaderboard and then discuss the latent features learned by our model on randomly selected samples.

4.4.1. Evaluation on Holdout Set

We separate the training data into a new validation and training set using the splitting script provided. This process gives us a training set with 60,922 users and a test set with 13,008 users. We use the first 3,000 users from the new test set to calibrate our models.

4.4.2. Results

Table 4.1 shows the results we achieved on the official leaderboard. The table also lists the scores that we obtained on our the validation set. The score difference between validation and challenge data might be partially explained by the different ensemble size that we used in the two evaluations. We selected an ensemble of 20 rankings for the last submitted model (as described in Section 4.3.3), but an ensemble of 100 rankings for our validation set evaluation. After

[3]The iterations that we selected were close to the best iterations on the holdout set, in other words, slightly before and after the best iteration.

some last-minute changes, we had only time to select 20 rankings for the final submission.

Table 4.1.: Results as reported on the challenge leaderboard and the official evaluation script (modified MAP@1000).

Model	validation	challenge
most-popular-item	0.0294	0.0259
FM-ens (with prefix smoothing)	0.0550	0.0472

4.4.3. Ranking Analysis

In this section, we demonstrate on samples that our model is able to represent name similarities and user preferences through its latent variables. We illustrate this on name characteristics such as gender or length because the effects of those items are easy to recognize. More subtle effects might also be captured by our model, but a casual inspection on a small sample might not suffice to identify them.

We select a subset of users from the challenge test set that have at least 30, but at most 50 activities. (We did so to avoid users without history and to save space by avoiding users with a very large histories). From this set, we randomly select 18 users and list them in Table 4.2. The first line $h(u,t)$ for each user lists all names in the user history. Duplicates and names that are not used in the competition (not listed in namelist.txt) have been removed. The second line lists the top 14 recommendations generated by our FM model[4].

Table 4.2.: Top 14 recommended names, for a random selection of users. Top14 shows the list of recommended names, where the leftmost name is the top recommended one. $h(u,t)$ are all names in the user history up to the last available time t. The last name selected by the user is printed in bold.

userId	set	names
3017	$h(u,t)$	tanja, birgit, antje, karin, heide, kevin, maria, arthur, mohammed, celine, cem, **mia**

[4]The number of listed recommendations and the number of randomly selected users are adjusted to the available space.

	top14	emma, anna, julia, johanna, michael, katharina, marie, thomas, sophie, eva, alexander, christian, charlotte, lena
1681	$h(u,t)$	roman, boris, leon, robin, david, stikar, krasimira, julian, fabian, erik, henrik, **bastian**
	top14	emma, daniel, alexander, thomas, anna, david, christian, andreas, michael, sebastian, paul, jan, julia, kevin
9617	$h(u,t)$	tobias, lasse, niklas, malte, anton, **justus**
	top14	malte, jan, arne, ole, mats, nils, tim, lars, mika, finn, ben, paul, matti, linus
22311	$h(u,t)$	jakob, karin, paula, charlotte, luise, miriam, rosa, caroline, olivia, **marie**
	top14	emma, pauline, greta, anna, luisa, julia, lena, marie, paul, johanna, mathilda, ida, maria, frieda
38852	$h(u,t)$	alma, reto, christoph, philipp, paul, peter, remo, karl, sarah, lukas, **alex**
	top14	emma, anna, alexander, julia, katharina, johanna, sebastian, lena, michael, marie, hannah, maximilian, jakob, elisabeth
17280	$h(u,t)$	ida, lotta, lena, sara, charlotte, emma, greta, nora, leo, leonie, emilie, **mara**
	top14	emil, greta, paul, emma, theo, frieda, anna, oskar, anton, paula, jakob, ben, moritz, julia
1399	$h(u,t)$	verena, kay, anni, eva, effi, friedrich, friedolin, xavier, james, paulchen, resi, **paul**
	top14	emma, anna, michael, thomas, christian, stefan, mia, alexander, andreas, markus, marie, daniel, greta, paul
21911	$h(u,t)$	mia, claudia, jasmin, nicolas, niklas, andris, leinis, lara, lavina, ben, **kilian**
	top14	emma, sophie, laura, julia, anna, mia, leon, lena, david, lea, sarah, luca, sophia, lisa
29470	$h(u,t)$	lasse, kjell, bodo, enno, benno, luis, mads, mats, **caspar**
	top14	louis, henri, emil, ben, levi, finn, paul, leo, ole, max, jonas, anton, till, noah
11408	$h(u,t)$	clemens, paul, anton, matthias, **isabel**
	top14	emma, anna, emil, jakob, alexander, andreas, thomas, julia, michael, moritz, felix, peter, jonas, johannes

28995	$h(u,t)$	noa, luitgard, heidrun, yeni, nova, zoe, naemi, sune, ruben, cano, **jon**
	top14	mila, emma, luca, mia, laura, lilith, mara, sarah, leonie, lea, julia, anna, sophie, sandra
6367	$h(u,t)$	jonas, anna, esther, anja, maris, linda, lorenz, lennard, jannes, **gerhard**
	top14	emma, anna, paul, julia, emil, greta, alexander, jakob, david, daniel, jonas, katharina, johanna, anton
33815	$h(u,t)$	raphael, yannick, timon, joël, rafael, neo, nearchos, iason, jason, **phineus**
	top14	daniel, alexander, thomas, david, sebastian, emma, christian, michael, anna, andreas, leon, jonas, kevin, elias
16175	$h(u,t)$	mirka, liane, irina, vincent, **carlos**
	top14	alexander, thomas, daniel, michael, andreas, christian, david, peter, sebastian, anna, emma, martin, markus, maximilian
6631	$h(u,t)$	tim, ellis, lino, nino, paul, leon, leonard, **tobi**
	top14	luis, ben, luca, finn, noah, jonas, max, julian, leo, lukas, tom, maximilian, jan, mika
14192	$h(u,t)$	jasmin, maja, matthias, felix, emilia, **luisa**
	top14	emma, paul, leon, anna, daniel, thomas, sebastian, moritz, alexander, simon, christian, julia, mia, max
31937	$h(u,t)$	ida, kalle, melanie, mara, katja, merle, nele, juno, linda, **marlen**
	top14	greta, emma, lotta, lina, mia, anna, ella, emil, frieda, lotte, charlotte, lasse, frida, julia
0508	$h(u,t)$	luisa, anna, bettina, emilia, yngwie, oskar, emil, albert, **frederick**
	top14	anton, paul, jakob, emma, moritz, theo, johann, karl, julia, julius, greta, paula, felix, jonathan

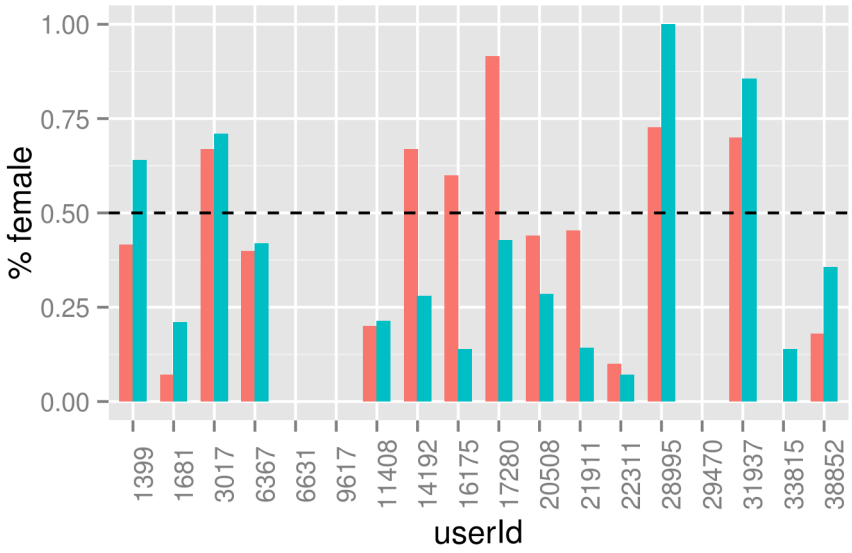


Figure 4.1.: Each bar pair represents the percentage of female names in the user history $h(u,t)$ (left) and the top 14 recommended names (right). The users evaluated here are the same ones as those listed in Table 4.2. Similar heights for both bars (left and right) for one user indicates that the user-preferred ratio of female to male names is closely reflected in the recommendations.

4.4.4. Discussion of Learned Effects

Even though the predictive strength is best judged by the leaderboard score, we want to give an impression of the kind of relationships that our model has learned from the training data. Please note that the rankings presented here are without prefix smoothing. This means that the model had no information about how long a certain name was or which characters were included since names are only represented by unique identifiers.
Our model learns to distinguish between **female** and **male** names and recognizes whether a user prefers male or female names. For users who have very strong preferences, such as user 6631, 9617, 29470, 3017, 28995, the recommendations are accordingly balanced (see Table 4.2 and Figure 4.1). For users with very few user activities, this heuristic becomes less reliable as can be seen with users 16175 and 14192.
The model also learns if a name is **long** (user 3017, 38852, 16175) or very **short**

(user 9617, 29470). It also recognizes if a user prefers **infrequent** names User 28995, for example, has *kjell* and *enno* in his history and has names like *levi* and *finn* recommended to him. **Double consonant** names are surprisingly frequent in the recommendations for user 31937, while names with **similar prefixes** have similar ranks for user 16175 (*martin, markus, maximilian*) or user 20508 (*julia, julius*); *lotte* and *charlotte* are ranked next to each other for user 31937.
A more detailed analysis of the learned ranking could reveal more information regarding how people judge the similarities of given names.

4.5. Conclusion

In this chapter, we have shown that a Factorization Machine (Rendle, 2012a) is well suited for the task of sequential link prediction. When the model parameters are optimized with respect to the personalized ranking criterion BPR-Opt (Rendle, Freudenthaler, Gantner, et al., 2009), the latent variables are able to express name preferences such as name length and gender. It is important to note that this information is not part of the model input. Being able to learn this characteristic is useful if this information is not readily available.

We have further shown that for infrequent names and small data sets, the injection of regularizing information such as syntactical similarity does improve the prediction quality. However, we expect that the benefit of information injection diminishes with the size of available data.

A casual inspection of the latent variables learned in this competition suggest that they represent syntactical and semantic similarity between given names. However, our approach is general and the latent variables will therefore represent whatever is useful for the specific data set to which we apply our model. We will build on this property when we learn latent relational features for relational classification (Section 8).

One of the main challenges when training our model was the selection of the right hyper-parameter for the stochastic gradient descent solver and the

sampling of training pairs. The next section will introduce a coordinate descent solver that does not suffer from these issues.

5. Implicit Factorization Machines

We propose a new coordinate descent solver to optimize the scoring function $\hat{y}\colon C \times I \to \mathbb{R}$ that generates the link predictions as defined in Section 5.1.

We chose a factorization machine to model the scoring function (as discussed in Chapter 4). This enables us to describe nodes by a general real-valued vector that allows us to characterize a node with general attributes. We can therefore use non-unique node descriptions which can improve the generalization capability of the model. Such node descriptions allow us to apply link prediction in situations where the node identities are unknown (Chapter 6).

The implicit factorization machine model that we propose in this chapter can be applied to general graphs. However, we will use a bipartite (two mode) graph setting and notation that is common in the recommender system community because this is where factorization machines are known best and because we use this setting in the next chapter to evaluate the new coordinate descent algorithm.

We will therefore refer to the bipartite interaction graph as *implicit feedback* (Section 2.3.2) and differentiate between *context-nodes* and *item-nodes* (or short context and item).

Next, we briefly recapitulate the standard FM model and standard coordinate descent learning algorithm (see Rendle, 2012a for details).

5.1. Learning from Implicit Feedback

We formalize the link prediction problem as modeling a scoring function $\hat{y}\colon C \times I \to \mathbb{R}$ which estimates a preference score $\hat{y}(c, i)$ for every context c and item i. The scoring function can be used to rank items within a context by

their preference, i.e. in a context c, an item i is preferred over an item j iff $\hat{y}(c,i) > \hat{y}(c,j)$.

We assume that the scoring function is parameterized by a set of model parameters Θ. We next describe the optimization objective for the model parameters Θ.

The implicit observations can be used to formulate a least squares problem where the optimal choice for the model parameters Θ is

$$\underset{\Theta}{\operatorname{argmin}} \sum_{c \in C} \sum_{i \in I} \alpha_{c,i} (\hat{y}(c,i) - y_{c,i})^2 + \sum_{\theta \in \Theta} \lambda_\theta \theta^2, \tag{5.1}$$

with regularization parameters λ_θ and costs $\alpha_{c,i}$. Note that this optimization objective is computationally very challenging because it spans over all context-item pairs, or, in other words $|C \times I|$ many pairs. In large-scale industrial applications, the number of items $|I|$ is on the order of millions or billions and likewise the number of contexts $|C|$. Clearly, an algorithm that requires $\mathcal{O}(|C|\,|I|)$ steps to solve eq. (5.1) is infeasible. In Section 5.2, we propose a learning algorithm for FM that solves this objective efficiently.

5.1.1. Background: Factorization Machines

Standard FM Model

A second-order FM for a feature vector $\boldsymbol{x}$ is defined (Rendle, 2012a) as

$$\hat{y}(\boldsymbol{x}) = w_0 + \sum_{j=1}^{p} w_j x_j + \sum_{j=1}^{p} \sum_{j'=j+1}^{p} x_j\, x_{j'} \langle \boldsymbol{v}_j, \boldsymbol{v}_{j'} \rangle \tag{5.2}$$

with model parameters $\Theta = \{w_0, w_1, \ldots, w_p, v_{1,1}, \ldots, v_{p,k}\}$. Note that $\langle \boldsymbol{v}_j, \boldsymbol{v}_{j'} \rangle$ models the effect of the variable interaction $x_j\, x_{j'}$ with the dot product of two k-dimensional latent vectors $\boldsymbol{v}_j, \boldsymbol{v}_{j'} \in \mathbb{R}^k$. According to (Rendle, 2012a), eq. (5.2) can be computed efficiently in $\mathcal{O}(kN_z(\boldsymbol{x}))$, where $N_z(\boldsymbol{x})$ is the number of non-zero entries in $\boldsymbol{x}$. An appealing property of an FM is the multi-linearity in each model parameter $\theta \in \Theta$:

$$\hat{y}(\boldsymbol{x}) = g_\theta(\boldsymbol{x}) + \theta h_\theta(\boldsymbol{x}), \quad \forall \theta \in \Theta \tag{5.3}$$

where g and h are independent of the value of θ and

$$h_\theta(\boldsymbol{x}) = \frac{\partial \hat{y}(\boldsymbol{x})}{\partial \theta} = \begin{cases} 1, & \text{if } \theta \text{ is } w_0 \\ x_l, & \text{if } \theta \text{ is } w_l \\ x_l \sum_{j \neq l} v_{j,f} x_j, & \text{if } \theta \text{ is } v_{l,f} \end{cases} \tag{5.4}$$

The definition of g_θ is skipped as we will never use it directly.

Standard FM Learning

Coordinate Descent (CD) is a popular algorithm to learn the model parameters Θ of an FM given a set S of labeled and weighted training samples $(\boldsymbol{x}, y, \alpha)$ — or $X, \boldsymbol{y}, \boldsymbol{\alpha}$ in matrix notation. CD iterates over the model parameters Θ and updates one model parameter $\theta \in \Theta$ at a time with

$$\theta^* = \frac{\sum_{(\boldsymbol{x},y,\alpha) \in S} \alpha \left[h_\theta(\boldsymbol{x})(y - \hat{y}(\boldsymbol{x})) + \theta h_\theta(\boldsymbol{x})^2 \right]}{\sum_{(\boldsymbol{x},y,\alpha) \in S} \alpha \, h_\theta(\boldsymbol{x})^2 + \lambda_\theta} \tag{5.5}$$

Clearly, the update costs are dominated by the two sufficient statistics

$$\sum_{(\boldsymbol{x},y,\alpha) \in S} \alpha \, h_\theta(\boldsymbol{x})^2, \qquad \sum_{(\boldsymbol{x},y,\alpha) \in S} \alpha \, h_\theta(\boldsymbol{x})(y - \hat{y}(\boldsymbol{x})). \tag{5.6}$$

A CD iteration over all training examples and model parameters can be performed efficiently in $\mathcal{O}(k \, N_z(X))$ (Rendle, 2012a). This requires that predictions be cached $\hat{\boldsymbol{y}} \in \mathbb{R}^{|S|}$ and sums factored $Q \in \mathbb{R}^{|S| \times k}$ for efficient computation of gradients (see eq. 5.4)

$$q_{i,f} := \sum_{j=1}^{p} v_{j,f} \, x_{i,j}. \tag{5.7}$$

5.1.2. Implicit FM (IFM)

In the following, we discuss how the FM model can be used as the ranking function $\hat{y} \colon C \times I \to \mathbb{R}$ (see sec. 5.1) and how to learn the FM model parameters Θ for the relational loss objective (eq. 5.1).

Feature Encoding

We encode every context c with a real-valued feature vector $\boldsymbol{x}_c^C \in \mathbb{R}^{p^C}$ of length p^C and every item i with a feature vector $\boldsymbol{x}_i^I \in \mathbb{R}^{p^I}$ of length p^I. Figure 6.2 shows an example for encoding the session context with real-valued vectors. For scoring an item i in a context c with a FM (eq. 5.2), we concatenate the two feature vectors $[\boldsymbol{x}_c^C, \boldsymbol{x}_i^I] \in \mathbb{R}^{p^C+p^I}$ and define

$$\hat{y}(c, i) := \hat{y}([\boldsymbol{x}_c^C, \boldsymbol{x}_i^I]). \tag{5.8}$$

IFM Learning

The optimization objective for the relational loss in eq. 5.1 can be seen as a standard regularized least squares problem with one training example $((\boldsymbol{x}_c^C, \boldsymbol{x}_i^I), y_{c,i}, \alpha_{c,i})$ for each context-item pair (c, i). Figure 5.1a shows how the naive training design matrix would look like for the context and item matrices in Figure 5.1b. It is obvious that the number of training examples (i.e. $|S| = |C \times I|$) created by relational feedback makes it infeasible to solve the optimization objective with the standard FM coordinate descent learning algorithm (see sec. 5.1.1).

5.2. Efficient IFM Learning

In the following section, we propose a new efficient coordinate descent learning algorithm for IFM . The algorithm is based on two observations: (i) the expensive part of the loss can be isolated and (ii) the IFM model can be decomposed into context- and item-specfic parts. This allows us to break the nested sums (over pairs) when computing sufficient statistics.

$$X^{\text{naive}} = \begin{pmatrix} c_{1,1} & c_{1,2} & i_{1,1} & i_{1,2} & i_{1,3} \\ c_{1,1} & c_{1,2} & i_{2,1} & i_{2,2} & i_{2,3} \\ c_{1,1} & c_{1,2} & i_{3,1} & i_{3,2} & i_{3,3} \\ c_{2,1} & c_{2,2} & i_{1,1} & i_{1,2} & i_{1,3} \\ c_{2,1} & c_{2,2} & i_{2,1} & i_{2,2} & i_{2,3} \\ c_{2,1} & c_{2,2} & i_{3,1} & i_{3,2} & i_{3,3} \\ c_{3,1} & c_{3,2} & i_{1,1} & i_{1,2} & i_{1,3} \\ c_{3,1} & c_{3,2} & i_{2,1} & i_{2,2} & i_{2,3} \\ c_{3,1} & c_{3,2} & i_{3,1} & i_{3,2} & i_{3,3} \end{pmatrix}, \quad \boldsymbol{y}^{\text{naive}} = \begin{pmatrix} a \\ b \\ 0 \\ 0 \\ 0 \\ c \\ 0 \\ d \\ 0 \end{pmatrix}, \quad \boldsymbol{\alpha}^{\text{naive}} = \begin{pmatrix} \alpha_a \\ \alpha_b \\ \alpha_0 \\ \alpha_0 \\ \alpha_0 \\ \alpha_c \\ \alpha_0 \\ \alpha_d \\ \alpha_0 \end{pmatrix},$$

(a) The naive design matrix X^{naive} has a row (example) for each possible context-item combination. As indicated by their zero value in $\boldsymbol{y}^{\text{naive}}$, very few of these combinations are actually observed.

$$X^C = \begin{pmatrix} c_{1,1} & c_{1,2} \\ c_{2,1} & c_{2,2} \\ c_{3,1} & c_{3,2} \end{pmatrix}, \quad X^I = \begin{pmatrix} i_{1,1} & i_{1,2} & i_{1,3} \\ i_{2,1} & i_{2,2} & i_{2,3} \\ i_{3,1} & i_{3,2} & i_{3,3} \end{pmatrix}$$

(b) The context C is described by a matrix X^C and the items I by X^I.

$$X^{\text{pos}} = \begin{pmatrix} c_{1,1} & c_{1,2} & i_{1,1} & i_{1,2} & i_{1,3} \\ c_{1,1} & c_{1,2} & i_{2,1} & i_{2,2} & i_{2,3} \\ c_{2,1} & c_{2,2} & i_{3,1} & i_{3,2} & i_{3,3} \\ c_{3,1} & c_{3,2} & i_{2,1} & i_{2,2} & i_{2,3} \end{pmatrix},$$

(c) The positive design matrix X^{pos} contains all of the observed session/item combinations with non-zero labels $\boldsymbol{y}^{\text{pos}}$.

Figure 5.1.: The IFM model uses the representations in figure 5.1b and 5.1c as a compressed version of the information in figure 5.1a.

5.2.1. Decompositions

Optimization Objective

In relational problems, most of the context- item pairs (c, i) have a count of zero, $y_{c,i} = 0$ (see fig. 5.1a), and only a few of them are non-zero – at most as many as there are observed events. We first decompose the loss from eq. 5.1 into a part that depends only on the (few) non-zeros and a part that depends on the rest.

$$\underset{\Theta}{\operatorname{argmin}} \sum_{(c,i)\in(C\times I), y_{c,i}\neq 0} \alpha_{c,i}(\hat{y}(c,i) - y_{c,i})^2 + \alpha_0 \sum_{(c,i)\in(C\times I), y_{c,i}=0} \hat{y}(c,i)^2 + \sum_{\theta\in\Theta} \lambda_\theta \theta^2. \quad (5.9)$$

Next, we reformulate[1] the loss such that the second part depends on **all** context-item pairs:

$$\underset{\Theta}{\operatorname{argmin}} \sum_{(c,i,y,\alpha)\in S_{pos}} \alpha(\hat{y}(c,i) - y)^2 + \sum_{c\in C}\sum_{i\in I} \hat{y}(c,i)^2 + \sum_{\theta\in\Theta} \frac{\lambda_\theta}{\alpha_0}\theta^2 \quad (5.10)$$

with

$$S_{pos} := \left\{ \left(c, i, \frac{\alpha_{c,i}}{\alpha_{c,i} - \alpha_0} y_{c,i}, \frac{\alpha_{c,i} - \alpha_0}{\alpha_0} \right) : y_{c,i} \neq 0 \right\}. \quad (5.11)$$

From the objective follows the optimal choice for the CD step[2]

$$\theta^* = \frac{\sum_{(c,i,y,\alpha)\in S_{\text{pos}}} \alpha \left[h_\theta (y - \hat{y}) + \theta h_\theta^2 \right] + \sum_{c\in C}\sum_{i\in I} \left[-h_\theta \hat{y} + \theta h_\theta^2 \right]}{\sum_{(c,i,y,\alpha)\in S_{\text{pos}}} \alpha h_\theta^2 + \sum_{c\in C}\sum_{i\in I} h_\theta^2 + \lambda_\theta / \alpha_0}. \quad (5.12)$$

[1] Note that the reformulation has the same optimum as eqs. 5.1,5.9.
[2] For the sake of brevity, we omit the argument (c, i) from $\hat{y}$ and h_θ.

Comparing this update to the standard CD step (eq. 5.5), we can see that in addition to the standard sufficient statistics (eq. 5.6), two additional sufficient statistics have to be computed

$$\sum_{c \in C} \sum_{i \in I} h_\theta^2(c,i), \quad \sum_{c \in C} \sum_{i \in I} h_\theta(c,i)\hat{y}(c,i). \tag{5.13}$$

The trivial computation of these statistics iterates over $|C \times I|$ pairs; however, it is independent of S_{pos}.

Model

In the IFM model (eq. 5.8) every model parameter $\Theta = [\Theta^C, \Theta^I]$ is either associated with a context or an item feature vector. This allows us to decompose the model into

$$\hat{y}(c,i) = \hat{y}_c^C + \hat{y}_i^I + \underbrace{\sum_{f=1}^{k} q_{c,f}^C \, q_{i,f}^I}_{\text{cross context-item}}, \tag{5.14}$$

where analogously to the standard FM (eq. 5.7), we define for the context

$$\hat{y}_c^C := \hat{y}(\boldsymbol{x}_c^C), \quad q_{c,f}^C := \sum_{j=1}^{p^C} v_{j,f}^C x_{c,j}^C. \tag{5.15}$$

The definitions for $\hat{y}_i^I$ and $q_{i,f}^I$ are symmetric to $\hat{y}_c^C$ and $q_{c,f}^C$.

5.2.2. Efficient Computation of Sufficient Statistics

In the following section, we demonstrate how to compute the sufficient statistics (eq. 5.13) efficiently by avoiding the two nested sums. We use the fact that the model (and consequently the gradient) can be decomposed and propose to precompute one of the sums. We show this computation for the sufficient statistics of the context parameters Θ^C. Because the model and the loss are symmetric, all results hold for the item parameters Θ^I as well.

Algorithm 2 Coordinate Descent Algorithm for IFM

1: **procedure** SOLVE($X^{\text{pos}}, \boldsymbol{y}^{\text{pos}}, \boldsymbol{\alpha}^{\text{pos}}, X^C, X^I$)
2: $\quad \boldsymbol{w}^C, \boldsymbol{w}^I \leftarrow (0, \cdots, 0)$
3: $\quad V^C, V^I \leftarrow \mathcal{N}(0, 1/\lambda_\theta)$
4: $\quad$ init Q^{II}, Q^{CC} $\quad \triangleright$ eq. (5.22), $\mathcal{O}(k^2 (N_z(X^C) + N_z(X^I)))$
5: $\quad$ **while** stopping criterion not met **do**
6: $\quad\quad$ init $\hat{\boldsymbol{y}}^{\text{pos}}$ $\quad \triangleright \mathcal{O}(k\, N_z(X^{\text{pos}}))$
7: $\quad\quad$ init $\hat{\boldsymbol{y}}^I$, $\hat{\boldsymbol{y}}^C$, $\quad \triangleright \mathcal{O}(k\, (N_z(X^I) + N_z(X^C)))$
8: $\quad\quad$ init $\boldsymbol{q}^{CI}$ $\quad \triangleright$ eq. (5.20), $\mathcal{O}(k\, (N_z(X^I) + N_z(X^C)))$
9: $\quad\quad$ **for** $l \in \{1, \cdots, p^C\}$ **do** $\quad \triangleright \mathcal{O}(N_z(X^{\text{pos}}) + N_z(X^C))$
10: $\quad\quad\quad$ calculate sufficient stats $\quad \triangleright$ eqs. (5.16, 5.18)
11: $\quad\quad\quad$ $w_l^C \leftarrow w_l^{C*}$ $\quad \triangleright$ eq. (5.12)
12: $\quad\quad\quad$ update $\hat{\boldsymbol{y}}^{\text{pos}}$, $\hat{\boldsymbol{y}}^C$
13: $\quad\quad$ **end for**
14: $\quad\quad$ Repeat lines 8 - 13 for the items.
15: $\quad\quad$ **for** $f \in \{1, \cdots, k\}$ **do**
16: $\quad\quad\quad$ init $\boldsymbol{q}_f^{\text{pos}}$ $\quad \triangleright \mathcal{O}(N_z(X^{\text{pos}}))$
17: $\quad\quad\quad$ init $\boldsymbol{q}_f^I$, $\boldsymbol{q}_f^C$ $\quad \triangleright$ eq. (5.15), $\mathcal{O}(N_z(X^I) + N_z(X^C))$
18: $\quad\quad\quad$ init $\boldsymbol{q}_{\neq f}^{CI}, \boldsymbol{q}_{\neq f}^{CII}$ $\quad \triangleright \mathcal{O}(k\, (N_z(X^C) + N_z(X^I)))$
19: $\quad\quad\quad$ **for** $l \in \{1, \cdots, p^C\}$ **do** $\quad \triangleright \mathcal{O}(N_z(X^{\text{pos}}) + N_z(X^C))$
20: $\quad\quad\quad\quad$ calculate sufficient stats $\quad \triangleright$ eqs. (5.17, 5.19)
21: $\quad\quad\quad\quad$ $v_{f,l}^C \leftarrow v_{f,l}^{C\,*}$ $\quad \triangleright$ eq. (5.12)
22: $\quad\quad\quad\quad$ update $\boldsymbol{q}_f^C$, $\hat{\boldsymbol{y}}^C$, $\hat{\boldsymbol{y}}^{\text{pos}}$, $\boldsymbol{q}_f^{\text{pos}}$
23: $\quad\quad\quad$ **end for**
24: $\quad\quad\quad$ update Q^{CC} $\quad \triangleright \mathcal{O}(k\, N_z(X^C))$
25: $\quad\quad\quad$ Repeat lines 18 - 24 for the items.
26: $\quad\quad$ **end for**
27: $\quad$ **end while**
28: **end procedure**

Calculating h^2 Efficiently

First-Order Model Parameters For the first-order model parameter w_l^C, the item sum in the first statistic in eq. (5.13) can be simplified

$$\sum_{c\in C}\sum_{i\in I} h^2_{w_l^C}(c,i) = \sum_{c\in C}\sum_{i\in I}(x_{c,l}^C)^2 = |I|\sum_{c\in C}(x_{c,l}^C)^2. \tag{5.16}$$

Thus, computing this sufficient statistic for all first-order context model parameters $\boldsymbol{w}^C$ is in $\mathcal{O}(N_z(X^C))$.

Second-Order Model Parameters For the second-order model parameter $v_{l,f}^C$, we isolate parts that depend only on the item

$$\sum_{c\in C}\sum_{i\in I} h^2_{v_{l,f}}(c,i) = \sum_{c\in C}(x_{c,l}^C)^2\left[\sum_{i\in I}(q_{i,f}^I)^2 + |I|(q_{c,f}^C - v_{l,f}^C x_{c,l}^C)^2 + 2(q_{c,f}^C - v_{l,f}^C x_{c,l}^C)\sum_{i\in I} q_{i,f}^I\right] \tag{5.17}$$

All item-related sums $\sum_{i\in I} q_{i,f}^I$ and $\sum_{i\in I}(q_{i,f}^I)^2$ are independent of context parameters and can be precomputed in $\mathcal{O}(N_z(X^I))$. This precomputation allows us to calculate the sufficient statistics for all context parameters in layer f in $\mathcal{O}(N_z(X^C))$.

Calculating $h\,\hat{y}$ Efficiently

First-Order Model Parameters For the first-order model parameter w_l^C, the second sufficient statistic (eq. 5.13) can be decomposed into

$$\begin{aligned}\sum_{c\in C}\sum_{i\in I} h_{w_l^C}(c,i)\hat{y}(c,i) &= \sum_{c\in C}\sum_{i\in I} x_{c,l}^C \hat{y}(c,i)\\ &= \sum_{c\in C} x_{c,l}^C\Bigg[|I|\,\hat{y}_c^C + \sum_{i\in I}\hat{y}_i^I + \underbrace{\langle \boldsymbol{q}_c^C, \sum_{i\in I}\boldsymbol{q}_i^I\rangle}_{q_c^{CI}}\Bigg]\end{aligned} \tag{5.18}$$

We can use this formulation to calculate the statistic for all first-order context parameters $\boldsymbol{w}^C$ in $\mathcal{O}(N_z(X^C))$ plus a cost of $\mathcal{O}(k\, N_z(X^C)$ for the q^{CI} cache. The sum $\sum_i \hat{y}_i^I$ is independent of the context and can be precomputed in $\mathcal{O}(k\, N_z(X^I))$. $\hat{\boldsymbol{y}}^C$ can be updated in $\mathcal{O}(N_z(X^C))$ per layer.

Second-Order Model Parameters For the second-order model parameter $v_{l,f}^C$, we decompose the second sufficient statistic of eq. (5.13) to

$$\sum_{c\in C}\sum_{i\in I} h(c,i)\,\hat{y}(c,i) = \sum_{c\in C}\Bigg[h_c^C\left(|I|\,\hat{y}_c^C + \sum_{i\in I}\hat{y}_i^I + q_c^{CI}\right) + x_{c,l}^C\Bigg(\hat{y}_c^C\sum_{i\in I} q_{i,f}^I + \sum_{i\in I} q_{i,f}^I\hat{y}_i^I + \underbrace{\langle \boldsymbol{q}_c^C, \sum_{i\in I} q_{i,f}^I \boldsymbol{q}_i^I\rangle}_{q_{c,f}^{CII}}\Bigg)\Bigg] \tag{5.19}$$

Again, we can precompute the item sums $\sum_i q_{i,f}^I, \sum_{i\in I} q_{i,f}^I\hat{y}_i^I$ in $\mathcal{O}(k\, N_z(X^C))$. Furthermore, we precompute q^{CII} and q^{CI} (details follow in the next section). These caches allow us to calculate the second statistic for all context parameters $\boldsymbol{v}_f^C$ in layer f in $\mathcal{O}(N_z(X^C))$.

Updating Caches

While iterating over context parameters, the values of the caches q^{CI} and q^{CII} change. We propose to remove the contribution of layer f from the cache, such that the cached value is independent of parameter updates in this layer

$$q_c^{CI} = \underbrace{\sum_{f'=1,\, f'\neq f}^{k} q_{c,f'}^C \sum_{i\in I} q_{i,f'}^I}_{\boldsymbol{q}_{\neq f}^{CI}} + q_{c,f}^C \sum_{i\in I} q_{i,f}^I. \tag{5.20}$$

This means we actually store $\boldsymbol{q}^{CI}_{\neq f}$ and $\boldsymbol{q}^{CII}_{\neq f}$ instead of $\boldsymbol{q}^{CI}$ and $\boldsymbol{q}^{CII}$ and add the contribution of layer f on the fly in constant time ($\mathcal{O}(1)$).

$$q^{CII}_{c,f} = \underbrace{\sum_{f'=1,\, f'\neq f}^{k} q^{C}_{c,f'} q^{II}_{f,f'}}_{\boldsymbol{q}^{CII}_{\neq f}} + q^{II}_{f,f} q^{C}_{c,f} \tag{5.21}$$

Secondly, Q^{II} is precomputed, an action which allows us to calculate $\boldsymbol{q}^{CII}_{\neq f}$ in $\mathcal{O}(k\, N_z(X^C))$ instead of $\mathcal{O}(k\, N_z(X^C) + k^2\, N_z(X^I))$.

$$q^{II}_{f,f'} = \sum_{i \in I} q^{I}_{i,f} q^{I}_{i,f'} \tag{5.22}$$

The Q^{II} cache is needed for $\boldsymbol{q}^{CII}_{\neq f}$ costs $\mathcal{O}(k^2\, N_z(X^I))$ to be computed, but can be updated in $\mathcal{O}(k\, N_z(X^I))$ per layer.

5.3. Algorithm Summary

Algorithm 2 shows how the computation of all sufficient statistics and caches and the updates of the model parameters is organized. One iteration over all model parameters Θ can be done in $\mathcal{O}(k\, N_z(X^{\text{pos}}))$ for the positive samples with an additional cost of $\mathcal{O}(k^2\, (N_z(X^C) + N_z(X^I)))$ for the remaining part. In practice, the number of factors k is usually small compared to the ratio of positive observations of the context and item complexity (i.e. $k \leq \frac{N_z(X^{\text{pos}})}{N_z(X^C)+N_Z(X^I)}$), which means that, $k^2\, (N_z(X^C) + N_z(X^I)) \leq k\, N_z(X^{\text{pos}})$. Consequently, the proposed efficient algorithm has an overall complexity of $\mathcal{O}(k\, N_z(X^{\text{pos}}))$. Note that this means, even though the original objective (eq. 5.1) has $|C \times I|$ many pairs and $|C \times I| \gg |S_{\text{pos}}|$, the proposed algorithm can solve the objective (eq. 5.1) with $\mathcal{O}(|S_{\text{pos}}|)$ costs.

6. Link Prediction using Node Attributes

In this chapter, we apply the Implicit Factorization Machine model introduced in the previous chapter to predict links for previously unseen nodes. We use this setting to test whether the IFM model can be used to model regular equivalence. By describing nodes only by attributes, we can use a general node description that doesn't require uniquely identifiable node connections.

For the evaluation, we use a *session*-based user personalization approach in which we represent a user not by an id, but in terms of the actions in the session so far. Our approach allows us to adapt the recommendations to a user's actions in real time. We build this model over a system called *IFM* (Implicit Factorization Machine, Section 5, that generalizes standard factorization machines to work over implicit feedback data. To optimize IFM, we use the efficient coordinate descent algorithm introduced in Section 5,an algorithm that exploits repeated structure in the objective function. We empirically demonstrate the benefits of IFM on YouTube video recommendation. In particular, we show that even without user ids, IFMs achieve better recommendation quality than user-item matrix factorization.

6.1. Implicit Feedback based Recommendations

Modern recommender systems (Amazon, Netflix, YouTube, etc.) employ sophisticated personalization models (Alvino and Basilico, 2015; Amatriain and Basilico, 2012; Davidson et al., 2010; Linden, Smith, and Com, 2003) to provide new and interesting content to users online. User personalization has been shown to improve both user satisfaction, as well as increase revenue for content

providers. To this end, there is a large body of work (Rendle, Freudenthaler, and Schmidt-Thieme, 2010; Koren and Bell, 2011; Agarwal and B.-C. Chen, 2010; Agarwal and B.-C. Chen, 2009; Gunawardana and Meek, 2009) on effectively incorporating personalization aspects into recommender systems with applications in advertising, movies, e-commerce, etc. Much of this research is focused on building machine-learned models where *user identities* play a key role, e.g. matrix factorization between user ids and item ids and its variations (Koren and Bell, 2011; Rendle, Freudenthaler, and Schmidt-Thieme, 2010; Koren, 2008; Koren, 2009a). These approaches capture user preferences in model parameters that are tied to the user identity, e.g. an embedding/latent factor vector that absorbs (through model parameter learning) the user's past interaction with the system.

While this approach has shown much success in the industry in recent years, there are a number of shortcomings.

1. User ids are only available if the user is logged in. In many applications, user identities are not available, e.g., while browsing incognito. In such cases, we only have limited short-term history within a session.
2. In anonymous sessions, user history is only tied to a (short-lived) session, and we have a limited window in which to make recommendations. Retraining a matrix factorization model with user ids is expensive and prohibitive to do online. Hence, we cannot instantly refresh recommendations for the user based on immediate actions.
3. User feedback is typically *implicit*, and it is hard to distinguish *unknown* and *negative* intent. Dealing with the large number of negative examples is challenging, and traditional matrix factorization algorithms do not scale.

In this chapter, we develop a session-based recommendation model that can be used to personalize anonymous sessions in real time. In this model, we represent a user, not by the user id (as is commonly done in matrix factorization methods), but rather by the actions (features) that have been observed in the session so far. This allows us to instantly update recommendations as soon as a new action of the user is observed and also deal with cold-start problems. Further, since the model does not contain parameters for user ids or session ids, we can train it offline. This is crucial for large-scale industrial serving environments because it avoids complex online learning, which would make the software stack

much more complex (Sculley et al., 2014). Note that even though the model is trained offline, it adapts to users feedback in real-time by updating the users feature representations.

Factorization Machines (FM) (Rendle, 2012a) are a natural fit for our model, and they can be used to provide a general feature representation for the users. In fact, FMs have been used successfully for several prediction and ranking problems through effective feature engineering. However, FMs in their current form are designed to model *explicit* feedback data, i.e., data with positive and negative examples. The main technical challenges with implicit feedback data is distinguishing between the positive and negative examples (Hu, Koren, and Volinsky, 2008; Rendle, Freudenthaler, Gantner, et al., 2009). There have been two main approaches to handle implicit feedback data: *Pairwise-ranking* based methods such as *Bayesian Personalized Ranking (BPR)* (Rendle, Freudenthaler, Gantner, et al., 2009; Weston, S. Bengio, and Usunier, 2010; Weston, S. Bengio, and Usunier, 2011), and SVD based methods (Hu, Koren, and Volinsky, 2008). However, both approaches have drawbacks: BPR is sensitive to negative item sampling (Rendle and Freudenthaler, 2014), and the SVD algorithm cannot handle side information. These drawbacks prevent us from being able to use them for session-based personalization models.

In this chapter, we use Implicit Factorization Machines (IFM), which extend the Factorization Machines model to work with implicit feedback data. Implicit feedback data is much cheaper to obtain and is available on a much larger scale: e.g., a user watched a video or purchased an item, etc. Owing to implicit feedback, the user-item matrix is not *sparse*. The observed actions of a user need to be contrasted against *all* the non-observed actions. In contrast, matrix factorization models only focus on observed entries in the user-item matrix. The resulting objective function for IFM has a quadratic complexity (U*I) which is infeasible for web-scale applications. By exploiting the repeated structure in the objective function, we develop a novel coordinate descent algorithm to optimize IFMs. We use IFMs to train a personalized session-based model to recommend videos on YouTube and continuously update them in response to user actions. The research contributions of the chapter include the following:

1. We develop algorithms to *instantly personalize* users sessions based on short-term feedback: we refresh our recommendations upon every user action, interleaving feedback gathering & recommendations.

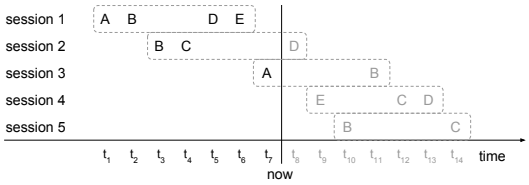


Figure 6.1.: A session is a sequence of choices among items I (here $I = \{A, B, C, D, E\}$) by a user. At any point in time, a recommender system should predict what items a user is interested in next.

2. We develop a novel co-ordinate descent algorithm to optimize Factorization Machine models with implicit feedback (IFM), generalizing the SVD model of Koren et al. (Hu, Koren, and Volinsky, 2008).
3. We empirically demonstrate the value of session-based recommendation models, obtaining better quality metrics than state-of-the-art user personalization models.

6.2. Personalization in Anonymous Sessions

6.2.1. Problem Setting

Let I be a set of *items* (e.g. videos, books, links) which can be consumed by a *user*. Recommender systems infer the users preferences from their past interactions with the application. Information that is gathered by observing the user's behavior is referred to as *implicit feedback* (Hu, Koren, and Volinsky, 2008). This is in contrast to explicit feedback, e.g. ratings, where the users state their preferences explicitly. Implicit feedback is available in much larger quantity as it can be gathered at no cost for the user. In this chapter, we will focus on implicit feedback.

A *session* is defined as a series of implicit feedback (i.e. choices of items) that can be tied to one user. Sessions are usually short term and last over a few minutes or hours. In this chapter, we deal with *anonymous sessions*, anonymous in that we don't know which user initiated the session. Moreover, there is no knowledge that would allow two sessions be linked to the same user.

Context Features

time	Country	Sequence	History
t_1	US	ø	ø
t_2	US	A	{A}
t_3	JP	ø	ø
t_4	JP	B	{B}
t_5	US	B	{A,B}
t_6	US	D	{A,B,D}
t_7	US	ø	ø
t_8	JP	C	{B,C}
t_9	US	ø	ø
t_{10}	JP	ø	ø
t_{11}	US	A	{A}
t_{12}	US	E	{E}
t_{13}	US	C	{C,E}
t_{14}	JP	B	{B}

Context Features: Real-valued encoding

Country		Sequence					History				
US	JP	A	B	C	D	E	A	B	C	D	E
1	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	1	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	1	0	0	0
1	0	0	1	0	0	0	1	1	0	0	0
1	0	0	0	0	1	0	1	1	0	1	0
1	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	1	0	0	0	1	1	0	0
1	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	1	0	0	0	0
1	0	0	0	0	0	1	0	0	0	0	1
1	0	0	0	1	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	1	0	0	0

Figure 6.2.: A context describes the situation of a user at a certain point in time. The context is generated from the past observations in the session (see fig. 6.1). The right table encodes the left table as real-valued feature vectors using the standard *one-hot* encoding.

Personalization is the task of selecting a subset of items that are of interest to a user in a given *context*. The context of the user is e.g. the past actions of the user or the video (s)he is currently watching. Obviously, good recommendations heavily depend on the context and furthermore the context changes very quickly, e.g. the user starts watching another video. This requires a real-time adaption of the recommender system to changes in a user's context.

Fig. 6.1 shows an example of five user sessions where items are observed over time. E.g. in session 1, first item A is selected by the user, then B, D and E. At any point in time, the goal of a recommender system is to predict what item the user wants to consume in her/his current context. E.g. for the second session, after observing B and C, the user actually selects D next.

6.2.2. Feature-based Modeling

We propose to extract a feature representation from the session that describes the user's context. We distinguish between static and dynamic context features. Static features are constant and do not change during a session, whereas dynamic features change as the session progresses. Fig. 6.2 shows an example for feature representations that describes the context of the sessions in fig. 6.1.

Static Context Features

- *Device*: information like browser (e.g. Chrome, Safari), operating system (e.g. iOS, Android, Windows) or device type (e.g. mobile, desktop).
- *Location*: the country of the user can be inferred from the IP address. More detailed location information is sometimes available, e.g. using GPS in mobile phones. If the resolution is sufficiently high, location might even become a dynamic feature when the user moves during the session.
- *Time*: date information can be used. More detailed time information (e.g. minutes, seconds) becomes a dynamic feature.
- *Session identifier*: Learning model parameters for a session id feature requires real-time updates, as the feature is not available in historical data but only at serving time. E.g. in fig. 6.1, if a recommender with the session id as a feature would be trained at time t_7, the recommender system could not make good predictions for session 4 or 5 because for both of them, the session id was not available during training.
- *User identifier*: We assume that user identifiers are not available. In sec. 6.2.3, we show experiments that compare user identifier features with our proposed features.

Dynamic Context Features

A dynamic feature depends on the user's context within a session.

- *History*: The set of all items selected by the user in the past. E.g. in fig. 6.1 for session 1 at time t_5, the user's history is $\{A, B\}$. At the next point in time t_6, the user's history has grown to $\{A, B, D\}$.
- *Sequences*: A sequential feature captures the previous interactions of the user. E.g. for session 1 (see fig. 6.1), at time t_5, the previously chosen item was B. At the next point in time t_6, the previously chosen item was D. The example in fig. 6.1 uses a sequence length of 1; longer sequences can be described similarly and might include a time decay.

As shown in the examples, both history and sequence features change over time as the session progresses.

The implicit observations indicate which items are preferred in a context. Let $y_{c,i}$ be how often item i has been chosen by a user under a context c in the

historical data. E.g. from the historical sessions in fig. 6.1 up to *now*, we can see that in two sessions item A was selected as the first item, B was selected once as the first item, and C, D, E were never selected as the first item. That means if the context is expressed by a sequential feature (i.e. $C = \{\perp, A, B, C, D, E\}$), then for the first item (i.e. $c = \perp$) the observed counts are $\hat{y}_{\perp,A} = 2$, $\hat{y}_{\perp,B} = 1$, $\hat{y}_{\perp,C} = \hat{y}_{\perp,D} = \hat{y}_{\perp,E} = 0$. And e.g. for $c = B$, the observed counts are $\hat{y}_{B,C} = \hat{y}_{B,D} = 1$ and $\hat{y}_{B,A} = \hat{y}_{B,B} = \hat{y}_{B,E} = 0$.

Discussion

History features capture the general interest of the user and this description gets more valuable over time. E.g. for a new session, the history feature is useless, but the longer the session takes, the more informative it is. In this sense, the history feature captures similar information as a user id. Both target the general tastes of the user and get better as the more feedback is available for the user. The difference is that the user id itself does not carry information but only through learning the model parameters associated with the user id is the user's feedback is captured. On the other hand, the history feature carries information by itself. That means, a user id needs real-time updates of model parameters to incorporate new feedback of the user whereas a history feature can use the existing model parameters and incorporate the new feedback simply by changing the feature representation of the context.

In contrast to history features, sequential features take ordering into account. Many applications have strong sequential effects, e.g. after watching episode 4 of a series, commonly users watch episode 5. History features cannot capture such ordering effects.

6.2.3. Experiments

The main objective of our experimental analysis is to show the benefits of using IFMs for instant personalization. We demonstrate that IFMs can digest new user feedback in real time and achieve better quality than commonly used methodologies such as item-based collaborative filtering. Further, even in offline scenarios where user ids are indeed available to be used as features,

we demonstrate that IFMs outperform traditional algorithms. We start with a brief description of the experimental setup.

Experimental Setup

Dataset

We sample around 205K user sessions from YouTube watch histories spanning a few hours. Here, the implicit feedback denotes whether a user watched at least 3 minutes or 80% of the video. The dataset also has side attributes for the users corresponding to age, country, gender and device information. In all, we have 68K videos and 850K watches. Over 25% of users have only 1 watch and about 68% users have less than 5 watches.

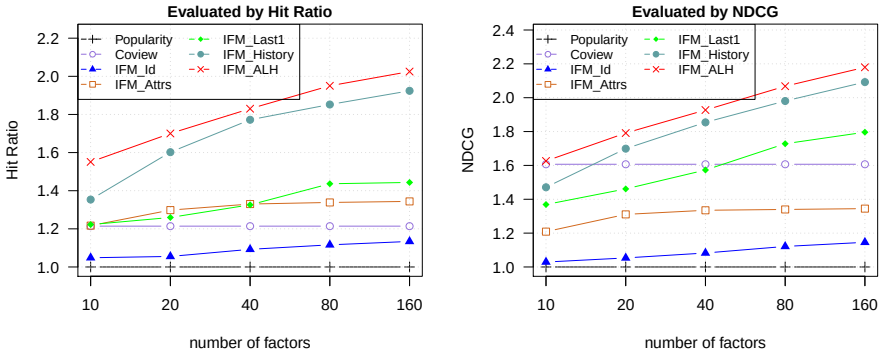
Recommendation Tasks

We consider two recommendation scenarios:

- *Online Protocol: Instant Personalization*
  We simulate realistic user sessions by splitting the dataset by a global timestamp; training on the first half only and holding out the second half for evaluation. Our task is to predict each of the held-out videos. Although we do not use the second half for training, we use it at prediction time to update the context. The results for this task are shown in Section 6.2.3.
- *Offline Protocol*
  Here, we use *leave-one-out* evaluation in which we holdout the last watch and train on the remaining data. While this is a commonly used strategy in the literature (Rendle, Freudenthaler, and Schmidt-Thieme, 2010), holding out the last item does not correspond to an absolute time-split and is unrealistic. We present the results for the offline setting in Section 6.2.3.

Evaluation Methodology

Popularity: Videos are ranked by popularity, i.e., the number of watches from the dataset. Even though it is not personalized, this technique is quite commonly used in anonymous sessions and cold-start scenarios.



(a) Hit Ratio (b) NDCG

Figure 6.3.: We illustrate the advantages of using non-user id based methodologies for Session-based recommendation. Note how IFM(A+L+H) outperforms IFM(Id). The figure also illustrates the improvements of IFM models over Co-view.

Co-View: Videos are ranked based on the co-occurrence statistics. This method captures the temporal nature of watch history and also allows for instant personalization. Co-View has been successfully used for recommendations on Amazon (Linden, Smith, and Com, 2003), for Netflix movie recommendations, and on Youtube (Davidson et al., 2010) etc.

IFM: We experiment with different features in the context (see Section 6.2.1), as shown below.

- User Id (I): We use user id in the context. This model, denoted by IFM(I) is equivalent to the SVD model of Hu, Koren, and Volinsky, 2008.
- User Attributes (A): The context includes user attributes – age, country, gender, device. This allows us to *weakly* personalize user sessions. Note that we do not use age and gender attributes in anonymous sessions.
- Last watched video (L): We use the last video watched in the context. This feature captures a user's short term interest.
- History (H): We use the entire history of previously watched videos in the session as context. Note that this is a *dynamic* feature that continuously grows in size as the session progresses.

The generality of IFMs enables us to use combinations of these features. For example, IFM(I+A+L) corresponds to the IFM model with user ids, attributes

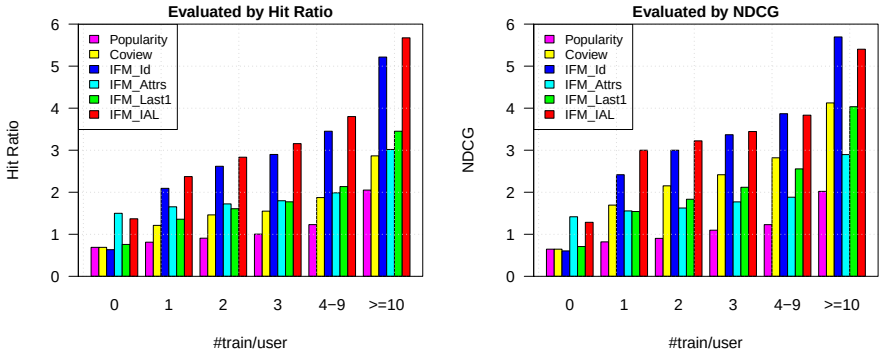


Figure 6.4.: Performance breakdown by number of training watches per user for Figure 6.5 at factors 40.

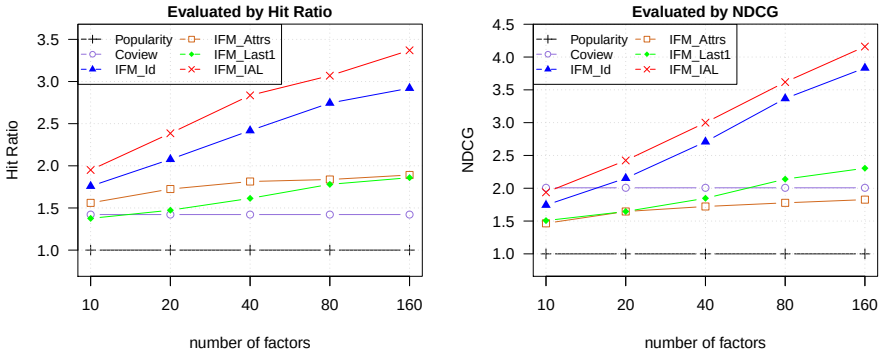
and the last watched video as context and IFM(A+L+H) corresponds to the IFM model with attributes, last watched video and history as context.

Evaluation Metrics

Given a ranked list of recommendations, we evaluate its quality using two popular ranking metrics: *Hit Ratio* (HR) and *Normalized Discounted Cumulative Gain* (NDCG) (Croft, Metzler, and Strohman, 2010). Note that HR is recall-based, measuring how many hold out items belong to the ranked list, whereas as NDCG is a precision-based metric. Also, note that both metrics are cut off at position 100. We only report relative improvements in the metrics owing to confidentiality. For all of our experiments, we determine the best hyper-parameters using grid search on a tuning set which is separate from the training data.

Online Protocol

As discussed in Section 6.2.3, to simulate real-user sessions, we split the dataset using an absolute time stamp (as demonstrated in Figure 6.1) such that the training and test sets have a roughly equal number of watches. We predict each video in the test set using the previous watches in the session. We evaluate the performance of each model described in Section 6.2.3. The results (for both HR and NDCG) are shown in Figure 6.3.



(a) Hit Ratio (b) NDCG

Figure 6.5.

First, we note that as the number of factors is increased, the IFM-based models progressively improve their quality over the baselines Co-view & Popularity. In particular, IFM(L) with 80 factors is at least (+x% NDCG, +y% HR) better than Co-view despite using exactly the same training data. This illustrates that IFMs can effectively use complex sequential features, and that Co-View-based instant-personalization approaches can be significantly improved with IFMs. We further note that using the entire watch history for the session, IFM(H) outperforms IFM(L) significantly and using all of the attributes IFM(A+L+H) beats IFM(H). These results further illustrate the benefits of using session-based personalization models for instant personalization.

Next, we observe that IFM(I) performs poorly, only marginally better than Popularity. This illustrates the shortcomings of user id-based methods in a real scenario as described in Section 6.1: Only 17% of the users occur in both training and test datasets, which means the ids are not useful for a large fraction of the users in the holdout set. This experiment highlights the superiority of utilizing histories over user ids – when new session data comes in, our history-based model can easily adapt the personalization by updating the predictors only; however, user id models have to be re-trained to incorporate the new data, something which is unrealistic to do online.

Finally, we note that the best performance is obtained by using all kinds of session features IFM(A+L+H), illustrating IFM's ability to provide enhanced personalization by sophisticated feature engineering. Also, while all of our

arguments apply to both Hit Ratio and NDCG, the effects are more pronounced for Hit Ratio, since our hyper parameters were optimized for HR.

Offline Protocol

As indicated in Section 6.2.3, in this experiment we hold out the last watched video and predict it based on the training set. This methodology is widely used in the literature (Rendle, Freudenthaler, and Schmidt-Thieme, 2010; Ahmed et al., 2013); however, it is flawed since the split point does not correspond to an absolute timestamp. This protocol provides favorable conditions to the user identity models by unintentionally enriching the training data (only one watch is in the test set for a user). In spite of this drawback, we show that IFM models can outperform traditional approaches.

In Figure 6.5, we show the recommendation metrics plotted against the number of factors in the model. In part(a), we show the metrics corresponding to HR, whereas in part(b) we show the NDCG values. We make the following observations: First, the IFM (I) model performs quite well in this case. This is because user id captures the user's watching history and provides a finer granularity of personalization than the other features in the model. This performance also demonstrates the bias in the leave-one-out evaluation strategy which is unrealistic in practice. Second, note that IFM(A) performs quite well, at least 50% better HR than the popularity baseline – demonstrating the power of side information to combat the cold-start problem. Third, we observe that using the combined set of features IFM(I+A+L) achieves the best performance overall, again showing the benefits of feature engineering with IFMs.

It is also interesting to compare the performance of Co-view and IFM (L) methods. While IFM (L) is consistently better than Co-view in terms of HR, it outperforms Co-view in NDCG only with more than 80 factors. IFM (L) and Co-view rely on the same transition matrix for prediction – IFM (L) predicts by factorizing, while Co-view directly uses the transition matrix for prediction. By factorizing, clearly IFM (L) can achieve better recall, but to be better in terms of precision, we need to use a much higher number of factors. This illustrates the effectiveness of our proposed IFM method in handling data sparsity when it is used for factorizing the transition matrix.

Instant personalization & Cold Start

To better understand our results, we bucket the evaluation metrics by the number of watches per user, as shown in Figure 6.4. For the cold-start users (#feedback per session = 0), IFM (A), which employs user attributes, provides a two-fold improvements over IFM (L), an improvement which boils down to popularity. This illustrates how IFMs can be used to ameliorate cold-start problems in recommender systems.

However, for users with just one training example, IFM(I) yields better performance than IFM(A), i.e., there is no benefit to using only attributes. And with more training examples, this gap actually widens. The cold start problem has been investigated in detailin the literature (Agarwal and B.-C. Chen, 2009; Kanagal et al., 2012); however, this experiment reveals the importance of effective instant personalization, even more than the problem of cold-start. Incorporating instantaneous user feedback into the model and refreshing the recommendations in real time provides significant improvement in the precision/recall metrics for all types of users. In modern web applications, users continuously interact with the system and expect recommendations to be updated continuously as well.

Finally, comparing the performance of IFM(A) with IFM(L) we can conclude that user attributes are most useful for cold-start users, while the last watched video is more effective for active users.

6.3. Related Work

Our work covers several research topics within recommender systems: implicit feedback, cooccurence recommender, cold-start and context-awareness.

The two most common approaches to learn from implicit feedback are to optimize a pairwise ranking objective with stochastic gradient descent (Rendle, Freudenthaler, Gantner, et al., 2009; Weston, S. Bengio, and Usunier, 2011) or to solve a least-squares problem over a dense feedback matrix (Hu, Koren, and Volinsky, 2008). The IFM algorithm is also a least-squares approach but in contrast to Hu, Koren, and Volinsky, 2008 which works only for the matrix

factorization model, we learn a general FM model which allows us to use arbitrary context representations. Pairwise models have been studied by Rendle and Freudenthaler, 2014 with more complex models, including FM; however, they require a good sampling algorithm for negative items which makes them harder to apply and consequently less attractive in industrial applications (Sculley et al., 2014).

One of the key advantages of our proposed approach is that it can instantly adapt to user feedback without retraining the model. Item-to-item co-occurrence recommenders share this advantage, one which makes them very popular in industrial applications (Linden, Smith, and York, 2003; Davidson et al., 2010). However, these models take only one signal into account and moreover have issues with tail items. Factorized co-occurrence methods (Rendle, Freudenthaler, and Schmidt-Thieme, 2010) have been studied to overcome the popularity issue and also to integrate long term user preferences, leveraging user identities as features.

From a modeling perspective, our IFM approach is related to context-aware recommender systems. Recent work incorporates context variables by tensor factorization models (Karatzoglou et al., 2010) or FM models (Rendle et al., 2011b), though, existing context-aware recommenders are usually not aimed for real-time personalization or for working without user ids.

Cold-start recommenders are the standard approach to deal with new users. E.g. Park and Chu, 2009; Agarwal and B.-C. Chen, 2009 incorporate user attributes in matrix factorization models to solve the cold-start problem. In our proposed approach, we also use such user attributes, but we argue in Section 6.2.3 that real-time personalization is much more important than the cold-start problem because after gathering the first observation, the effect of the attributes already vanishes.

6.4. Conclusion

In this chapter, we have presented a new approach to personalize sessions in real-time. We propose to describe a user not by his/her identity, but by observed actions. This approach, allows us to personalize the recommendations,

even if user is anonymous, and moreover allows us to instantly adapt the recommendations in real-time to the user's actions. We use a factorization machine (FM) to model the item preferences of a user within a context. We develop a new optimization algorithm that can learn FMs for implicit feedback data efficiently without additional costs for the negative/unobserved feedback. We show empirically on Youtube video recommendation data that our proposed approach achieves high quality recommendations and can adapt to user actions instantly.

Part III

Relational Classification

This part continues the development of a *computationally efficient prediction model for relational data* by investigating the relational classification problem as defined in Section 3.2.3. Now that we have an efficient link prediction model, we first show how to construct relational features that can be used with every maximum-likelihood-based machine learning model to achieve state-of-the-art performance for relational classification problems. We then show how this work can be combined with the results from part II by learning relational features automatically with the IFM model.

We start in Chapter 7 by introducing new *relational features* for standard machine learning methods by extracting information from *direct* and *indirect relations*. We then demonstrate, empirically, on three standard benchmark data sets that our relational features yield relational classification results comparable to state-of-the-art Structural Relational Learning methods. Finally, we show that our proposal outperforms these methods when additional information is available.

After establishing the high predictive accuracy of maximum likelihood based machine learning models for relational classification we introduce a new relational classification approach that is based on the IFM model in Chapter 8. This approach can easily be combined with the graph-based relational features that we introduced in Chapter 7. We further introduce a random-search-based algorithm to automate the hyper-parameter optimization. The performance of the latent features is investigated in detail and compared against the manually constructed features.

7. Graph Based Relational Features

Statistical relational learning (SRL) methods are used when samples are connected by one or more relations. These relations are helpful in tasks like scientific article classification where patterns such as "connected samples have similar labels" are very predictive. Feature-based ML methods, in contrast, often assume that samples are independently, identically distributed (iid). This approach is well established, allows for efficient parameter estimation and simplified prediction, but ignores the relational information available in SRL settings. Recently, a number of methods have been proposed (Macskassy and Provost, 2003; Taskar, Segal, and Koller, 2001; Lu and Getoor, 2003; Neville and Jensen, 2000) that significantly improve performance over *classical* methods by using joint inference. Exact joint inference has high runtime complexity, which often requires approximate solutions (Sen et al., 2008). These approximated joint inference techniques introduce new difficulties such as the need for specialized implementations that are expensive to run and difficult to tune (Sen et al., 2008).

In this chapter we propose to transfer the relational information into classical features. This approach allows for a straightforward combination of relational and classical (attribute) information. It also renders traditional feature-based ML methods competitive in settings in which relational information is available and allows for the leverage the large body of classical ML methods and their scalable algorithms.

We use three standard *relational classification* (RC) benchmark datasets to show that classical ML with relational features are strong competitors for state-of-the-art SRL methods on this task. Note that on these datasets, collective classification is considered the best performing methods in the current

literature (Jensen, Neville, and Gallagher, 2004; Macskassy and Provost, 2007). In particular, we make the following contributions:

- We discuss how joint inference could be avoided by extending the sample description with relational features (Section 7.1.1).
- We extend relational features to indirect relations (Section 7.1.2). This process is new and crucial to achieve high accuracy when only a few samples are labeled (Sect/on 7.3.3).
- We introduce a new cluster-based relational feature (Section 7.1.2) that provides strong results and is cheap to compute.
- We show that our approach improves state-of-the-art relational classification even in network-only settings (Section 7.3.2).

7.1. Problem Setting

We start by giving the necessary definitions with the traditional setting of samples $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ where y_i is the class label and x_i is a feature vector describing sample i. We assume that for the first u samples ($i \in \{1, \ldots, u\}$), the class label y_i is known and for the samples with index $i > u$, the class label is unknown. Relations among samples are represented by weighted, symmetric adjacency matrices $R_k \in \mathbb{R}^{n \times n}$, and the complete relational information is denoted as $\mathcal{R} = \{R_k\}_{k=1}^{K}$. While in general, each of the k relations could be complete, i.e. provide some similarity between each pair of samples, we explicitly consider the case of sparse, unweighted relations where only a minority of node pairs are connected by an edge. We start with a statistical argument to motivate the representation of relational information in a way that is compatible with the iid assumption.

7.1.1. Preserving IID

Many machine learning algorithms are based on the maximum likelihood principal to learn the optimal value θ^* for model parameters given a dataset $\mathcal{D}$ (c.f. (Murphy, 2012))

$$\theta^* := \underset{\theta}{\operatorname{argmax}}\, p(\theta|\mathcal{D}) = \underset{\theta}{\operatorname{argmax}}\, p(\mathcal{D}|\theta)$$

where $l(\theta) := p(\mathcal{D}|\theta)$ is called the likelihood. A very common assumption in many ML approaches is that the samples in the dataset D are independent and identically distributed (iid). This assumption simplifies the likelihood to

$$p(\mathcal{D}|\theta) \overset{iid}{\propto} \prod_{i=1}^{N} p(y_i|x_i, \theta).$$

One of the central arguments of relational learning is that examples are not iid. In particular for any pair of examples (x_i, y_i) and (x_j, y_j) conditional independence does not hold

$$p((x_i, y_i), (x_j, y_j)|\theta) \not\propto p(y_i|x_i, \theta)\, p(y_j|x_j, \theta).$$

Note that in this formulation the relational information $\mathcal{R}$ is completely neglected. However, if $\mathcal{R}$ is used, it can render the probabilities independent

$$\begin{aligned} &p((x_i, y_i), (x_j, y_j), \mathcal{R}|\theta) \\ &\qquad \propto p(y_i|x_i, \mathcal{R}, \theta)\, p(y_j|x_j, \mathcal{R}, \theta). \end{aligned} \tag{7.1}$$

This formulation is close to standard non-relational ML with iid of samples.

Standard ML algorithms assume that all information about a sample can be encoded in a (usually real-valued) feature vector x. Let us assume that the influence of $\mathcal{R}$ on sample i can be described through a finite number of real-valued variables x_i^r. We call x_i^r *relational features*. To simplify notation, we can define $\tilde{x}_i$ as the extended feature vector of an example i that combines both non-relational features x_i and relational features x_i^r. In total, this allows us to rewrite Equation 7.1 as

$$p(\mathcal{D}|\theta) \overset{iid}{\propto} \prod_{i=1}^{N} p(y_i|\tilde{x}_i, \theta).$$

Note that due to the relational information in $\tilde{x}_i$, the iid assumption can be preserved. In the remainder of this section, we discuss several ways to generate relational features.

7.1.2. Graph Based Relational Features

Samples can be linked through multiple relations, each of which can be described as a graph. Representing each relation by an independent set of features allows us

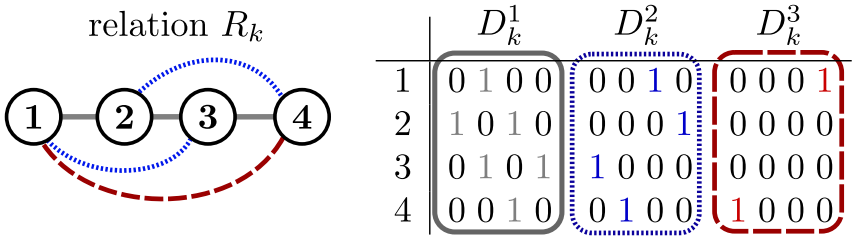


Figure 7.1.: Feature matrix for the node neighborhoods of a relation. The edges of the original relation are depicted as gray lines, while connections in distance two and three are shown as blue/short dashed and red/long dashed curves.

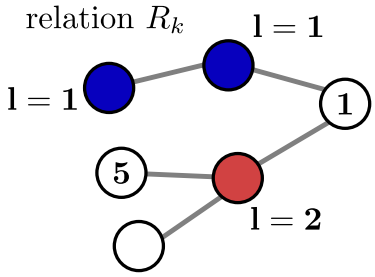
to integrate an arbitrary number of relations per problem into a standard feature matrix. All of the proposed features have in common that the encoded relational information does not only consist of direct relations, but also captures indirect relations, a circumstance which we found to be the key to their performance.

Neighbor Ids

Encoding the direct neighbors of a sample i in relation R_k can be achieved by treating each sample as a categorical variable which is true when samples are connected and false otherwise (Bernstein, Clearwater, and Provost, 2003; Perlich and Provost, 2003; Perlich and Provost, 2006). As this information is very local and yields limited information about i's position in R_k, we extend it by additionally including indirect neighbors at various distances to i. Distance refers to the number of edges $d_k(i, j)$ on a shortest path connecting i and j in R_k. In particular, for a (small) set of distances $d \geq 1$ we describe each relation R_k by the distance-d neighborhood matrices $D_k^d \in \{0, 1\}^{n \times n}$ with $\left(D_k^d\right)_{i,j} = 1$, if $d_k(i, j) = d$ and 0 otherwise. We illustrate the derivation of this feature in Figure 7.1.

Aggregated Neighbor Attributes

Relational position can also be described by individual features of direct and indirect neighbors (Neville, Jensen, Friedland, et al., 2003; Neville, Jensen, and Gallagher, 2003). As before, we extend the idea by calculating individual



sample	C_k^1	C_k^2	C_k^3
1	1 1	1 0	0 0
5	0 1	0 0	1 0

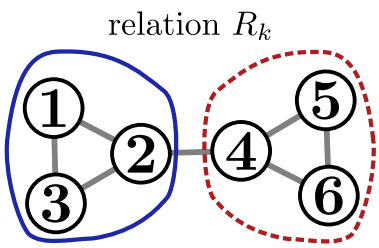
Figure 7.2.: Aggregation for attribute counts on distances one, two and three in the relation R_k. Labels are given in text and as color (1-blue, 2-red). Sample 1 has a single node of each label as direct neighbors which is reflected in the first two columns(C_k^1) of its row, while the next two columns (C_k^2) encode the single 2-labeled node in distance 2. Note that unlabeled nodes (white) are ignored in the features.

features at various distances. For a categorical attribute with categories $1, \ldots, c$, we define an $n \times c$ matrix L with $L_{i,j} = 1$ if x_i is labeled as j and $L_{i,j} = 0$ otherwise. Then the count matrix $C_k^d =: D_k^d L$ can be derived as the projection of the corresponding neighborhood matrix to the label matrix L such that $(C_k^d)_{i,j}$ yields the number of category j nodes in distance d of sample i. We denote this feature by *neighbor class counts* (NCC) and provide an illustration in Figure 7.2. An additional row normalization yields a probability matrix for the class labels in distance d, a situation which we will label *neighbor class probabilities* (NCP).

Random Walk Similarity

While the features described above are based strictly on shortest paths, random walks with restart (rwr) incorporate a different notion of connectivity. They have been proposed as a similarity measure in the context of auto-captioning images (J.-Y. Pan et al., 2004).

The similarity between two nodes is measured as the probability of a random walk connecting them, i.e. the probability of the random walk process visiting one node when it has started at another. To control for locality this includes a restart probability: in each step the walk will jump back to the starting point



sample	M_1^k	
1	1	0
2	1	0
3	1	0
4	0	1
5	0	1
6	0	1

Figure 7.3.: An example for encoding a clustering of a relation as a feature vector. Memberships of samples in clusters are represented as binary features.

with probability r. This can be modeled as

$$p_i = (1 - r)Wp_i + re_i \tag{7.2}$$

where the column vector p_i of matrix P describes the steady-state probability distribution over all samples for walks starting at i. W is the matrix encoding transition probabilities, i.e. a L_1 row normalized version of R_k, e_i is the vector of zeros and unit value at position i and the parameter r is the restart probability. P can be determined as the solution of a linear system or approximated efficiently (Tong, Faloutsos, and J.-Y. Pan, 2006), leaving r as free parameter. We derive $\hat{P}$ from P by column-wise L_2 normalization and use analogues to the neighbor features row i of $\hat{P}$ as features for sample i.

Clustering Memberships

Clustering methods can be used to identify groups of similar samples. Clustering features encode this information by representing this group membership. Given a clustering of the graph representing relation R_k into c clusters, we obtain an $n \times c$ feature matrix M_k^c with $(M_k^c)_{i,m} = 1$, if sample i belongs to cluster m and zero otherwise. Since a single clustering yields limited information about the dense groups in the graph, we create features for various clusterings, i.e. different c clusters. We limit c to $c = 2^j$ subject to $2 \leq c \leq n$ which limits the number of clusterings to $\lfloor \log_2(n) \rfloor$, while also providing a wide range of cluster sizes. This results in $O(n)$ features per relation that are very sparse with only $O(\log(n))$ non-zero features per sample. The clusters can be calculated

with negligible runtime using the METIS clustering framework[1] (Karypis and Kumar, 1998). Note that the dense subgroups identified in the clusterings can be directly related to the homophily assumption often exploited in relational learning.

7.2. Related Work

We build on two main categories of related work. The first, in Section 7.2.1 uses features derived from the network structure to improve iid-based inference. The second, discussed in Section 7.2.2 is work that views collective classification as a joint inference problem, simultaneously inferring the class label on every instance. The challenges specific to problems with few labeled data points have received special attention (Macskassy and Provost, 2007; Saar-Tsechansky and Provost, 2007; Gallagher, Tong, et al., 2008; Gallagher and Eliassi-Rad, 2010) and helped us to understand the importance of indirect relations.

7.2.1. Relational Features

Relational features can be combined with collective inference or directly used with standard ML methods as we argue in Section 7.1.1. Models such as Relational Probabilistic Trees (Neville, Jensen, Friedland, et al., 2003), Relational Bayes Classifier (Neville, Jensen, and Gallagher, 2003) and the Link Based Classifier (LBC, (Lu and Getoor, 2003)) concentrate primarily on the aggregation of attributes of connected samples. Others use rows from the (weighted) adjacency matrix as the basis for feature construction (Bernstein, Clearwater, and Provost, 2003; Perlich and Provost, 2006; Perlich and Provost, 2003). We were especially inspired by the suggestion to extend the neighborhood of samples with few neighbors with distance-two neighborhoods (Preisach and Schmidt-Thieme, 2006) or ghost edges (Gallagher, Tong, et al., 2008). In contrast to previous work, we kept the information from various neighborhood distances separated and introduced the concept of multiple indirect relations.

[1] We used the implementation available at `http://glaros.dtc.umn.edu/gkhome/views/metis` with default parameters.

7.2.2. Collective Inference

Full relational models such as Markov Logic Networks (MLN, (Richardson and Domingos, 2006)) or the Probabilistic Relational Model (Taskar, Segal, and Koller, 2001) can be used for CC (Crane and McDowell, 2012). We refer to Sen et al. Sen et al., 2008 for a comprehensive overview of collective inference based CC algorithms. Their strength in high label autocorrelation settings and the problem of error propagation has been examined by Jensen, Neville, and Gallagher, 2004; Xiang and Neville, 2011 and improved inference schemes have been proposed McDowell, Gupta, and Aha, 2009. Recently, stacking of non relational model has been introduced (Kou and Cohen, 2007) as a fast approximation of Relational Dependency Networks (Neville and Jensen, 2003).

7.3. Evaluation

In our experiments[2] we investigate the following three questions:

1. Are classical ML methods with relational features competitive with MLN and Collective Classification approaches?
2. What are the main ingredients that make relational features effective?
3. Does the combination of relational and attribute information improve results?

7.3.1. Experimental Setup

Datasets We use three standard benchmark SRL datasets. The *Cora* and *CiteSeer* scientific paper collections have been used in different versions; we chose the versions[3] presented in Sen et al., 2008 and the *IMDb* dataset[4]. Both, Cora and CiteSeer include text features in the form of bag of words (bow) representations. We give some statistics of these datasets in Table 7.1.

[2] Our code is available from https://github.com/ibayer/PAKDD2015
[3] http://linqs.cs.umd.edu/projects/projects/lbc/index.html
[4] http://netkit-srl.sourceforge.net/data.html

	Nodes	Links	Classes	|Dictionary|	avg. Degree
Cora	2708	5278	7	1433	3.8981
CiteSeer	3312	4660	6	3703	2.8140
IMDb (all)	1377	46124	2	—	66.9920

Table 7.1.: Summary statistics of the datasets.

Benchmark Models As baseline models we use the well-established relational learning methods wvRN (Macskassy and Provost, 2003; Macskassy, 2007), nLB (Lu and Getoor, 2003) and MLN (Crane and McDowell, 2012). We chose *relaxation labeling* (Chakrabarti, Dom, and Indyk, 1998) as the collective inference method for wvRN and nLB, as it has been shown to outperform Gibbs sampling and iterative classification on our datasets (Macskassy and Provost, 2007). For MLN we used the rules `HasWord(p,+w)=>Topic(p,+t)` and `Topic(p,t)^Link(p,p')=>Topic(p',t)`, together with discriminative weight learning and MC-SAT inference as recommended for Cora and Citeseer in a previous study (Crane and McDowell, 2012).

Measures & Protocol We follow Macskassy and Provost, 2007 and remove samples that are not connected to any other sample (singletons) in all experiments. Each experiment is repeated 10 times with class-balanced splits into ratios of $0.1, 0.2, \ldots, 0.9$ for the train-test set (shown as percentage in the figures). The MLN experiments are only repeated 3 times due to their extremely high evaluation time. We calculate multi-class accuracies by micro averaging and plot them on the y-axis of each figure in this section. We used the netkit-srl framework (Macskassy and Provost, 2007) in version (1.4.0)[5] to evaluate the wvRN and nLB classifiers and the Alchemy 2.0 framework[6] to evaluate the MLN model. The graph clusterings were obtained using METIS, version (Karypis and Kumar, 1998). Relational features are learned with an L_2 penalized logistic regression model[7] included in scikit-learn (Pedregosa et al., 2011) version 0.14.01. The penalty hyperparameter C is optimized via grid search over $\{0.001, 0.01, 0.1, 1, 10, 100, 1000\}$ on the training set.

[5] http://netkit-srl.sourceforge.net/
[6] https://code.google.com/p/alchemy-2/
[7] We use a one-vs-all scheme for multiclass classification.

7.3.2. Comparing Relational Features to SRL

This section examines whether feature-based relational models (without collective inference) are able to compete with the prediction quality of specialized relational learning methods. Consequently, the benchmark is a task in which only relational data is available. In the first experiment, we compare wvRN and nLB with two logistic regression models that use only our relational features. The first relational feature model (rwr) is based on a random walk. The second model uses both neighborhood and aggregated (NCP) features with distances 1, 2 and 3. We exclude distances higher than three, since almost every node can be reached over three edges from every other node and therefore further distances do not provide additional information.

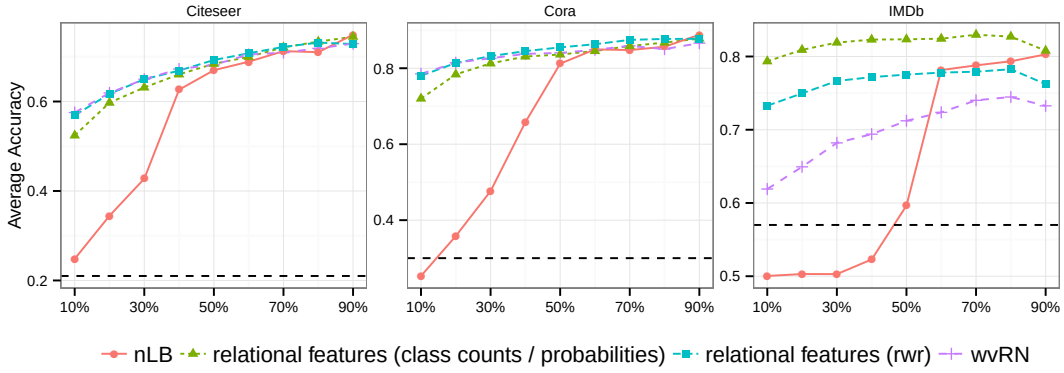


Figure 7.4.: Comparison between two SRL methods (nLB, wvRN) with relaxation labeling and two relational feature-based models on network-only data. The dashed black lines indicate base accuracy.

Figure 7.4 illustrates two problems of SRL models: (i) nLB performs poorly[8] when labels are sparse and (ii) wvRN is sensitive to violations of its built-in assumptions — i.e. if label consistency among neighbors is not met, as with the IMDb dataset.

The relational feature-based models show a very consistent performance not much affected by the number of labeled samples. The results of the neighbor and NCP feature combination on IMDb illustrate the flexibility of relational features.

[8]This has been attributed to a lack of training data (Macskassy and Provost, 2007).

7.3.3. Engineering Relational Features

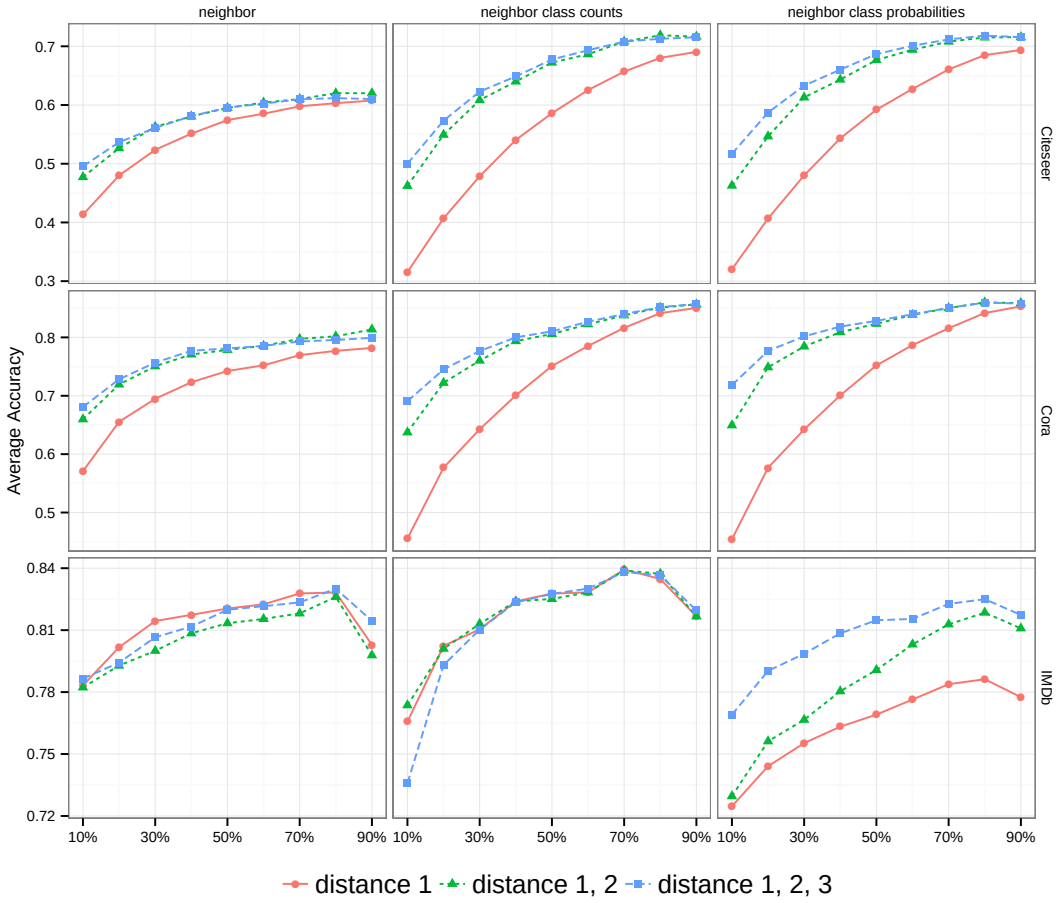


Figure 7.5.: Influence of the distance parameter for label dependent and independent relational features. Including information from indirect neighbors improves results, especially if few samples are labeled.

We now examine the different relational features and the influence of their parameters. Two questions will be addressed: (i) *How important are indirect relations?* (ii) *Which of the proposed relational features lead to the best results?* All relational features that we consider can incorporate information about indirect neighbors. Each method has parameters that adjust the locality of the resulting features. We first examine the effect of including indirect neighbors

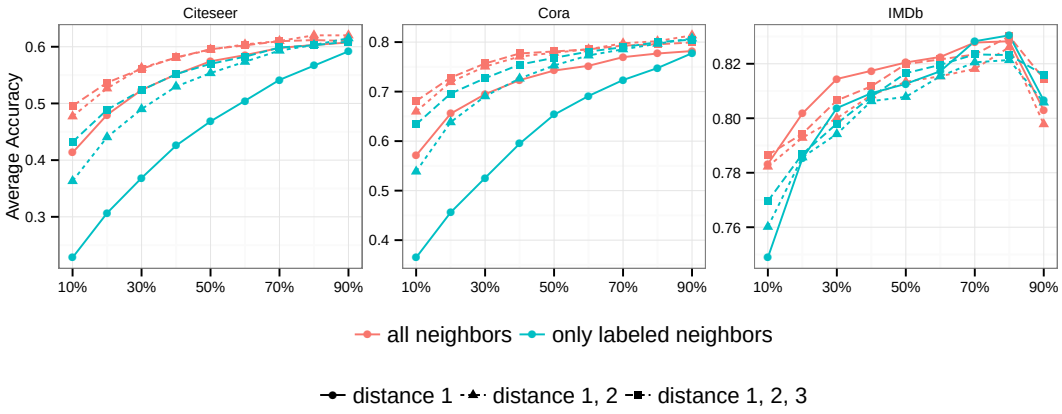


Figure 7.6.: Comparison of the use of only labeled vs all neighbors to construct relational neighbor features (NCC). Including unlabeled neighbors improves the results in all settings.

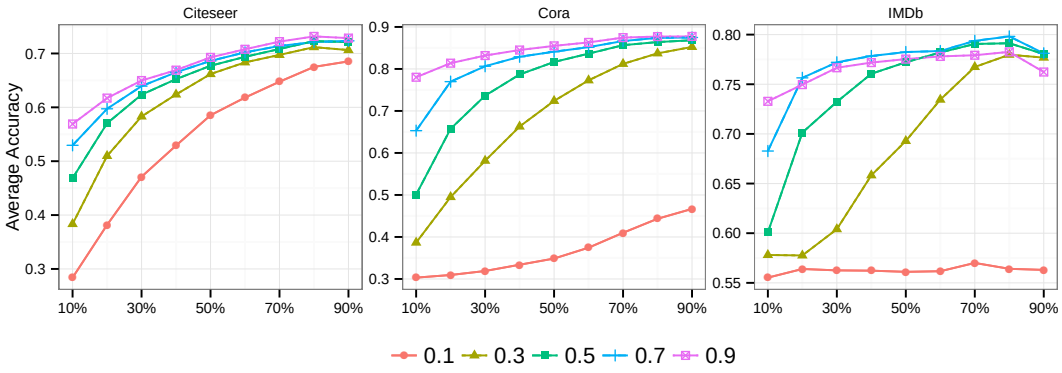


Figure 7.7.: Influence of various restart values on the prediction accuracy. While as a general trend larger values of r (c.f. Equation 7.2) lead to better results, the value $r = 0.9$ yields consistently good results.

(Figure 7.5) and the importance of unlabeled neighbors (Figure 7.6) for relational neighbor features. The influence of the restart parameter r (c.f. Equation 7.2) for the rwr features can be seen in Figure 7.7, and an informative subset of results for various numbers of clusters is shown in Figure 7.8. The results suggest that the inclusion of indirect neighbors in the relational features is beneficial independently of whether they are used directly or for aggregation.

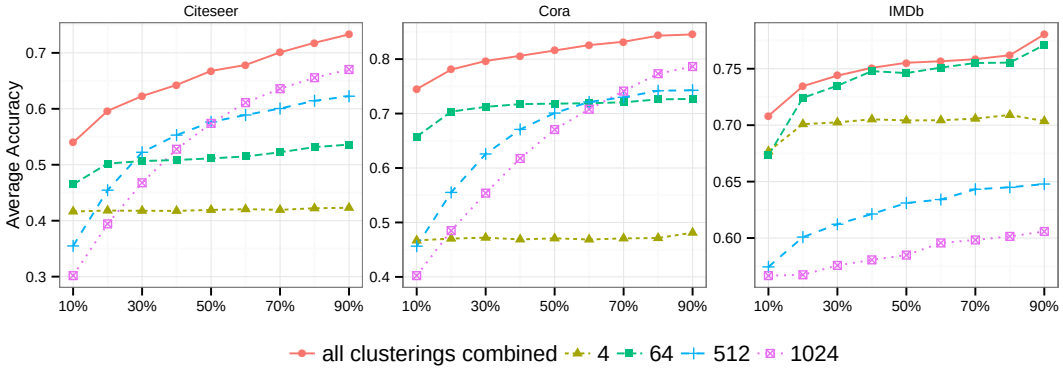


Figure 7.8.: Examination of various clusterings of the dependency network and their usage as relational features. The optimal number of clusters strongly depends on the dataset and ratio of labeled samples. The combination of all clusterings, however, is consistently superior to individual clusterings.

Figure 7.6 shows that unlabeled neighbors contribute significantly to the overall performance. Together this answers our first question: unlabeled samples and indirect neighborhood relations are essential ingredients for relational features.

7.3.4. Combining Relational and Local Information

In the following section, we examine the effect of adding local attributes. Figure 7.9 shows the results with neighborhood count features (NCC) of distances 1,2,3. Interestingly, the bag-of-words model performs better than network-only models on Citeseer but worse on Cora. Combining relational and local attributes, on the other hand, improves results in both cases. The figure further shows that our features outperform MLN on both datasets. In summary, our experiments suggest that the combination of relational features and attributes is beneficial, even with a simple model such as logistic regression.

7.3.5. Discussion

Our experiments indicate that relational feature-based models compare well to specialized relational learners even in network only and sparse labeling settings.

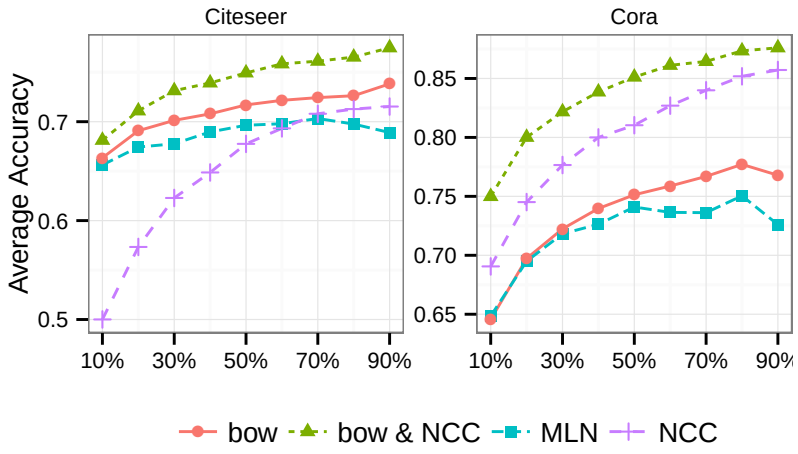


Figure 7.9.: Comparison of attribute only (bow), network only (NCC), MLN, and our combination of relational features with local attributes.

This has been verified on three standard SRL benchmark datasets and with three state-of-the art SRL methods for comparison. The inclusion of indirect neighbors has proven extremely important, especially in sparse label settings. We have further shown that the combination of relational features and local attributes is both straightforward and has the potential to improve considerably over both feature-only and network-only models.

Note that our relational features can lead to very high dimensional representations. Such feature spaces are, however, common in recommender systems, click-through rate prediction, and websearch where regularized logistic regression has been shown to be very effective (Graepel et al., 2010; J. Liu, J. Chen, and J. Ye, 2009). In addition, we use a standard implementation of logistic regression and can consequently employ scalable versions that can be trained with billions of samples of high dimensions (Mukherjee et al., 2013; Rendle, 2013). We are further not committed to the logistic regression model, as our features could be used as input for arbitrary-vector-space models.

7.4. Conclusion

We have shown that dependencies between samples can be exploited using relational feature engineering. Our method allows us to combine relational information from various sources with attributes attached to individual samples. We tested this on standard SRL benchmark datasets, showing that even on network-only data, our features are competitive with specialized relational learning models. In addition, our features can outperform them when additional information is available. Note that in contrast to the SRL methods, our proposal achieves these results without collective inference. While we restricted our experiments to logistic regression as a prediction model, the proposed features could be used as input to any other feature-based learning algorithm such as SVM, neural networks, or random forests. Extending the use of relational features to multi relational datasets would be straight-forward and an interesting direction for further research.

8. Learning Latent Relational Features

This chapter examines how Factorization Machines trained on implicit feedback and combined with a standard classifier can be used for relational classification. Our approach doesn't assume that connected observations share the same class and allows for the inclusion of node attributes easily. We use the fast coordinate descent algorithm for implicit feedback from Chapter 5 to train the IFM model on a surrogate link prediction task. We evaluate our approach on three well-studied relational classification data sets and compare it to state-of-the-art collective classification methods.

8.1. Introduction

Most machine learning techniques used to classify observations assume that observations are independent, given the attributes that describes them. This includes popular methods such as SVMs, Neural Networks, Decision Trees, and others. It is also widely known that these methods work very well in practice, even in situations when the observations aren't completely independent. However, knowing the dependence between observations can be very informative, for example when observations form pairs that always belong to the same or distinct categories. In such situations, the data can be described by a graph with the observations as nodes connected by links that correspond to the observed dependencies.

Classifying a node in a graph is known as relational classification. A wide range of collective classification methods (Sen et al., 2008) have been proposed for this task, methods that assume that connected nodes have a high probability of

belonging to the same class. These methods have been demonstrated to perform very well when linked observations have a high probability of belonging to the same class. However, when the dependency between observations is not directly related to the classification task, the performance of these methods deteriorates quickly. Methods such as cautious collective classification (McDowell, Gupta, and Aha, 2009) aim to mitigate this problem. Another popular research direction is to include node attributes into the collective classification process (Gallagher, Tong, et al., 2008).

We instead propose to address the relational classification problem by learning latent node attributes from the dependency structure and use them as input for a standard classification model. This approach doesn't require us to make any assumptions about how similar connected nodes are with respect to the classification task and allows for easy inclusion of node attributes. We train an Implicit Factorization Machine (IFM) model to predict links in the data graph and use the learned parameters as latent node attributes. We use the fast coordinate descent algorithm for implicit feedback from Chapter 5 to train the IFM model on the surrogate link prediction task and use a simple logistic regression model as classifier. We evaluate our approach on three well-studied relational classification data sets and compare it against start-of-the-art collective classification methods. To summarize, our contributions include the following:

1. We introduce a new feature-based relational classification approach based on learned latent node attributes.
2. We introduce a robust random search algorithm that combines link prediction and relational classification.
3. We show that Implicit Factorization Machines can be used to learn latent attributes for relational classification efficiently.
4. We conduct empirical experiments on three well-known relational classification data sets.

8.2. The Problem

Consider a directed graph $G = (V, E, x, d)$ with a set of vertices (nodes) V, and a set of edges (links) E, where each edge is an ordered pair (u, v) of vertices

with the edge pointing from u to v. We further define node attributes (features) $x\colon V \to \mathbb{R}^p$ and dyad weights $d\colon V \times V \to \mathbb{R}$. We use D for the set of all possible node pairs or dyads. We define E as the set of dyads with non-zero weight $E = D \setminus \{d(u,v)\} = 0$.

Given a graph $G = (V, E, x, d)$ with a subset of labeled nodes $V_l \subset V$, a set of unlabeled nodes $V_u = V \setminus V_l$, and a set of binary labels Y, we define the node classification problem as the task of assigning labels $\tilde{y}\colon V \to Y$ to all unlabeled nodes V_u of the graph.

8.3. Relational Classification with IFMs

We propose a new relational classifier that uses the IFM model to learn latent node attributes. The model consists of two components: The first is the IFM model that is used to learn a latent variable representation for each node. The second component is an iid-based classifier trained on the labeled nodes V_l of our data graph G. This approach shares the feature-based modeling flexibility with the IFM model, as well as the computation efficiency if an appropriate classifier is chosen. The observable node attributes are combined with the latent attributes as input for the classification model. This step can be seen as a transformation from the original input graph G to a new graph with extended node attributes and no edges.

8.3.1. Combining relational classification and link prediction

Jointly learning a model for relational classification and link prediction has the advantage that uncertainty and incompleteness in both — node and edge attributes — can be incorporated into the model. Link prediction models such as IFMs learn latent node attributes that can't be constructed by aggregating attributes from neighbor nodes. For example, a simple way to construct and include latent node attributes is to cluster a graph and use the cluster membership as a latent attribute. We demonstrated that this approach is able to capture relational information in Chapter 7. Using a more expressive IFM model to learn latent node attributes allows us to improve on these results.

8.3.2. Edge and Node Model

Looking at feature-based link prediction and relational classification methods together demonstrates that they have much in common. For relational classification we estimate the probability that a node u has a certain label based on its set of features $x^C(u)$ and the model parameter Θ_C.

$$p(y^C|G) = p(y^C|X^C, \Theta^C)$$

The model parameters Θ^C are obtained by optimizing the maximum likelihood for the given data in the form of the known labels y^C and the feature matrix X^C. The feature matrix contains one row for each node with a known node label.

The link prediction model estimates the model parameter Θ^E based on the observed link structure ($y^E_{i,j}$ has a value of one for connected pairs and zero otherwise).

$$p(y^E|G) = p(y^E|X^E, \Theta^E)$$

The feature matrix $X^E \in \mathbb{R}^{|V|^2 \times p^E}$ is constructed from the graph G and contains a row for each of the $|V| \cdot |V|$ node pairs.

Concatenating Node Pairs

Both models construct a feature matrix from graph G and infer model parameters based on known labels also derived from the graph. What sets them apart is what an observation represents. An observation of the relational classification model represents a node, while the link prediction model operates on pairs of nodes. But by defining a link as a concatenation of two node representations $x(i, j) := [x_i, x_j]$, the observations of the two models become compatible. (Matrix factorization models that include node attributes are known as context-aware when they incorporate node attributes in to the model.)

Conditioning on Latent Node Attributes

Relational classification can be transformed into a classical classification problem by adding information to the observations that make them conditionally independent (for more detail see Chapter 7). We effectively condition the

classification model on previously learned hidden node attributes. This allows us to use logistic regression for the classification step. Logistic regression is a simple classifier that has been successfully used for relational classification before (Bilgic and Getoor, 2008).

8.4. Modeling and Inference

This section demonstrates how to combine logistic regression and IFM to a unified relational classification model. This unified model is illustrated by Figure 8.1. Gray nodes represent the observed data. G is the complete data graph, y^E is the link structure, and y^C are the known node labels. The square nodes represent deterministic functions. $\mathcal{X}^E$ is the feature-engineering process that extracts the design matrix X^E from the data graph G, and $\mathcal{X}^C$ is the feature-engineering process that combines the latent features (V, w) and the data graph G to the relational feature matrix X^C for the node classifier. The plates show how often the nodes covered by the plate occur in the model. For example, we have one y_i^C for each labeled node $y^C \in \mathbb{R}^{N^c}$. The relational design matrix X^c is covered by two plates and therefore has two indices $X^c \in \mathbb{R}^{N^c \times p^c}$. The hyper-parameters for the parameters w, V and z are introduced in the next sections.

8.4.1. Learning Latent Node Attributes

We use the implicit coordinate descent algorithm (Chapter 5) to learn an Implicit Factorization Machine model that learns to predict links in our data graph. We use the link prediction task as a surrogate problem in order to learn meaningful latent node attributes. We place a Gaussian prior on the first and second order IFM parameter. We draw the hyper-parameters λ_w and λ_V from a uniform distribution.

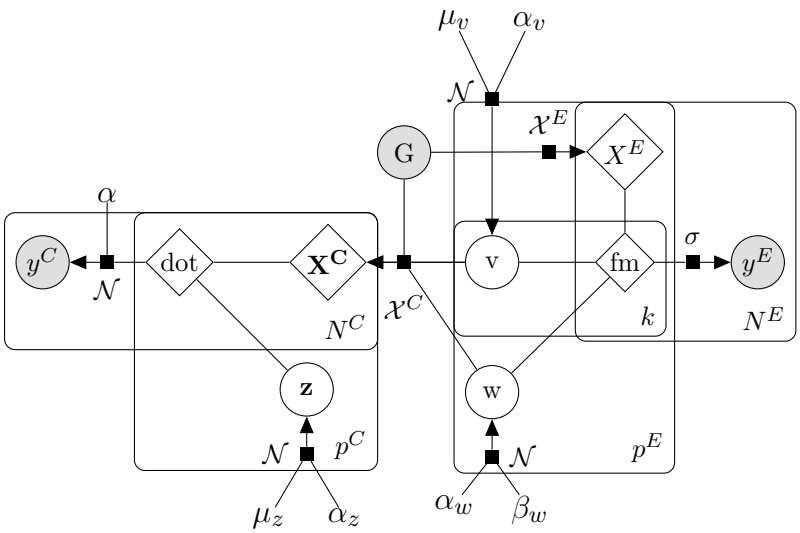


Figure 8.1.: Directed factor graph of the combined link prediction and relational classification model.

$$
\begin{aligned}
y^E &\approx \text{Normal}(\mu^E, 1) \\
\mu^E &= \hat{y}^{IFM}(X_E, w, V) \\
w &= \text{Normal}(0, \lambda_w) \\
V &= \text{Normal}(0, \lambda_v) \\
\lambda_w &= \text{Uniform}(a^w, b^w) \\
\lambda_v &= \text{Uniform}(a^v, b^v)
\end{aligned}
$$

The IFM model requires a set of training samples S^+ that describe the connected nodes. The target $y_{i,j}$ is set to one if the nodes n_i and n_j are connected and to zero otherwise. However, we don't have to construct the set of non-connected note pairs explicitly (for details see Chapter 5).

Each training example is constructed by concatenating the vector representation of the two nodes. Good feature engineering is required to find the best node representation for the problem at hand. A simple, but powerful node representation is to one-hot (dummy encode) the node ID. This means that we can represent the node pair (u, v) by the following vector:

$x(u,v) = [0,\dots,\underbrace{1}_{u},0,\dots,0,\underbrace{1}_{v},0,\dots]$. We treat the data set as an undirected graph in our experiments by including each pair (u,v) and also (v,u). This representation corresponds to standard matrix factorization, but more complex features can be used as well.

8.4.2. Node classification

The following discussion describes the generative model for the logistic regression classifier with L_2 regularization. The regularization parameter λ_w is again drawn from a uniform distribution.

$$\begin{aligned} y^C &\approx \text{Normal}(\sigma(\mu^c), 1) \\ \mu^C &= (X^C)^T z \\ z &= \text{Normal}(0, \lambda_z) \\ \lambda_z &= \text{Uniform}(a^z, b^z) \end{aligned}$$

For the stacked classifier, each sample $x^C(i) \in R^{p^c}$, represents a node in the graph. Feature engineering is required to find a good node representation. We only use the latent vectors learned by the link prediction model as node representation for our experiments $x^c(i) = V_i$. But we could also use any other information $x^a(i)$ that we have about the node as additional features.

$$x^c(i) := [V_i, w_i, x^a(i)] \tag{8.1}$$

This means that the output of any other relational classifier can be easily included in our model.

8.4.3. Parameter Tuning

Different approaches can be used to jointly learn the parameter of the logistic model and the IFM model. For example, one could alternatingly optimize one and then the other model, but it is difficult to guarantee that these approaches will converge. We therefore use a simple two-stage procedure: We first optimize the IFM model, then construct the classification design matrix X^C based on

the latent and observed node attributes, then optimize the logistic regression model.

Approximating the joint inference with a two-stage process stills leaves us with the challenge of tuning hyper-parameters. The next section describes the robust random search algorithm that we use for our experiments.

Random Search Algorithm

We use a random search algorithm to tune all model parameters $\Theta = \Theta_{FM} \cup \Theta_C$. The two optimization problems are governed by the hyper-parameter $\Theta^H = \Theta^H_{IFM} \cup \Theta^H_C$.

Algorithm 3 performs a random search for a predefined budget of draws. We perform multiple draws of the hyper-parameter Θ^H_{IFM} for each draw of the IFM hyper-parameter for two reasons: First, the classifier is faster to train; second, the classifier performance is sensitive to how we split the training data. We use stratified sampling to select new holdout nodes at each iteration and evaluate the classification accuracy. Then the classifier is trained on a subset of the labeled nodes and evaluated on the remaining training nodes. Finally, the best hyper-parameters are selected and used to retrain the model on all available node labels. This model is then used once to predict the labels for nodes without known labels. All steps are listed in more detail in algorithm 3.

Estimation Costs

Algorithm 3 uses a random sampling of the hyper-parameter. This has been demonstrated to be often more effective than a grid search (Bergstra, Bardenet, et al., 2011; Bergstra and Y. Bengio, 2012) . Random sampling makes our tuning algorithm embarrassingly parallel on both the inner and outer sampling loop. However, random sampling ignores the information available from previous draws. The overall complexity of the algorithms is linear to the number of links $|E|$ and depends on the density of the two design matrices, X^C and X^E. One iteration of the logistic regression model can be done in $O(N_z(X^E))$, and the implicit IFM solver requires $O(|V|k^2 + |E|k)$ time for one iteration. Most importantly, our approach works well on sparse graphs and doesn't require

Algorithm 3 Parameter Optimization Algorithm

procedure SOLVE(G, V_l)
  init scores
  while IFM budget not empty **do**
    $\Theta^H_{IFM} \leftarrow$ uniform(a^w, b^w), uniform(a^V, b^V)
    $\Theta_{IFM} \leftarrow$ trainIFM(Θ^H_{IFM}, X^E, y^E)
    $X \leftarrow P(\Theta_{IFM}, G)$
    while Classification budget not empty **do**
      $\Theta^H_C \leftarrow$ uniform(a^z, b^z)
      train, test $\leftarrow$ sampleTrainTestSplit()
      $\Theta_C \leftarrow$ trainClassifier($\Theta^H_C, X^C(train), y^C(train)$)
      score $\leftarrow$ evaluateClassifier($\Theta_C, X^C(test), y^C(test)$)
      scores.add(score, Θ^H)
    end while
  end while
  $\Theta^H_* \leftarrow$ selectBest(scores)
  $\Theta_* \leftarrow$ retrainModel(Θ^H_*, X^C, y^C)
  $\hat{y}^U \leftarrow$ predict(Θ_*, X^C_u) ▷ Predict unknow node labels
end procedure

sub-sampling of unconnected node pairs in order to avoid a complexity of $O(|V|^2)$.

8.5. Evaluation

We performed experiments on three well-known relational classification data sets (Cora, Citeseer and IMDb) in order to evaluate the usefulness of latent node attributes for relational classification. The features used for the IFM model are the node IDs, and the classifier uses only the latent attributes associated with the node (see Section 8.4.2 and 8.4.1 for details). We first evaluate our approach against state-of-the-art baselines and then examine how the number of hidden node attributes influence the model performance.

Experimental Setup

We evaluated performance by hiding a fixed percentage of node labels, (ranging from 0.1 to 0.9), training the model on the remaining node labels, and evaluating the accuracy on the hidden labels. Each experiment was repeated ten times, and the average accuracy was reported. The hidden labels are only used to evaluate the performance so that no information can leak into the model-tuning process. We use a budget of two hundred repetitions for the IFM embeddings (line 3) and a budget of fifty repetitions for the stacked classifier (line 7). For more details about the data sets, evaluation procedure, and base line models see Chapter 7 which uses an identical setup.

8.5.1. Comparison against Collective Classification Baselines

We first compare our latent attribute model against two popular collective classification modes (nLB and wvRn) that are known to perform well on these data sets.

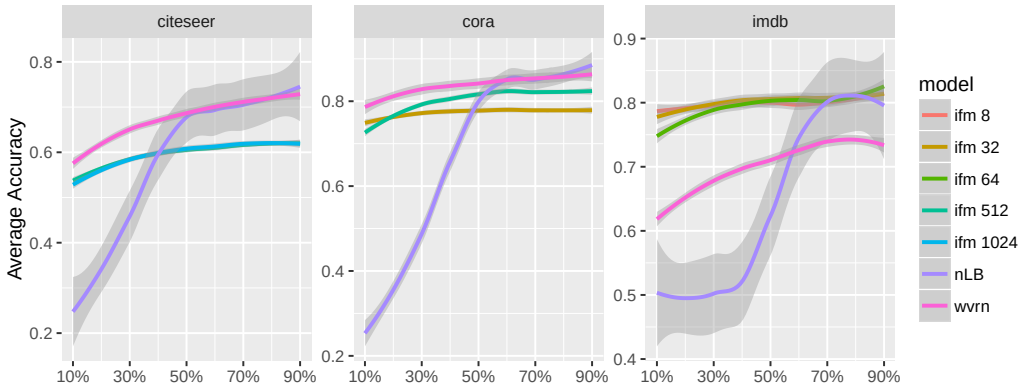


Figure 8.2.: The x-axis denotes the number of labeled nodes in the graph and the y-axis demonstrates the average accuracy on the holdout nodes. For each data set, results are presented for the best performing rank.

The results in Figure 8.2 show that our model performance varies when the percentage of labeled nodes changes; this is especially clear in contrast to the nLB model. However, when comparing our prediction performance with the

baselines, no clear picture emerges. The model outperforms the baselines on imdb, achieves slightly worse results on cora and performs much worse on citeseer.

8.5.2. Number of Latent Attributes and Sparse Labeling

Choosing the number of latent attributes (rank) is a trade-off between being able to model complex relation-ships and overfitting the model on spurious information.

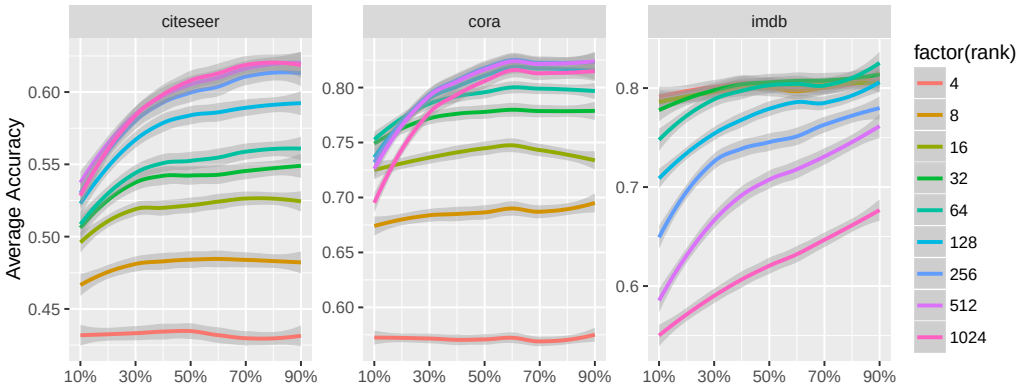


Figure 8.3.: The x-axis denotes the number of labeled nodes in the graph, and the y-axis shows the average accuracy on the holdout nodes. For each data set, the results are presented for a range of different ranks.

Figure 8.3 shows us that the best rank is independent of the percentage of labeled nodes, except for a small region of the Cora data set. This indicates that the latent node attributes are able to extract information that are independent of the node labels. A model tuned with a small number of labeled nodes can therefore increase its performance only by relearning the classifier component.

8.5.3. Data Set Specific

Models with a high number of latent attributes are able to capture more complicated relationships than simpler models. Figure 8.4 shows that each data

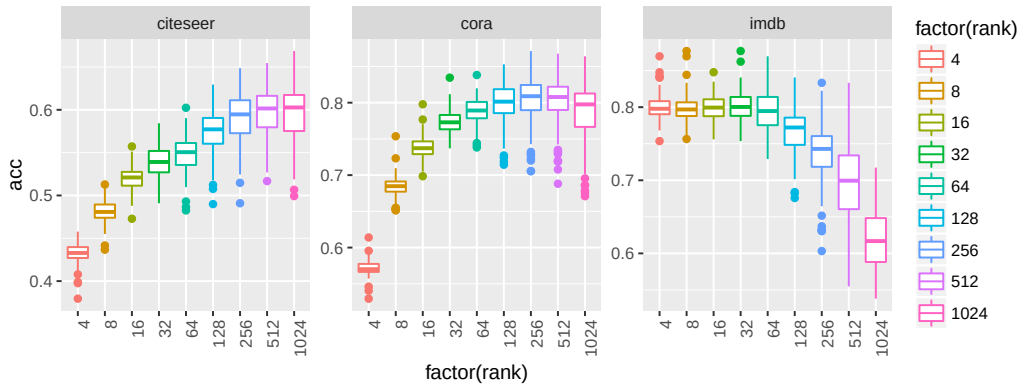


Figure 8.4.: Each boxplot summarizes the performance over all different sparsity levels on which this rank has been evaluated on.

set has a different range for the number of latent attributes that achieve the best performance. For the IMDb data set the model is able to achieve very good results (compared to the base lines) with a low number of latent attributes. For the Cora data set, the performance first improves, but deteriorates again for complex models. The performance on Citeseer improves with the complexity of the model, but comes never close to the base lines. This could be an indication that the feature engineering for X^G wasn't able to capture all of the available information.

8.6. Related Work

The challenge of combining link prediction and relational classification has been approached from many different directions. Iteratively optimizing a link prediction and relational classification loss has been investigated by Bilgic, Namata, and Getoor, 2007. Predictions of one model are included in the other if the predictions surpass a confidence threshold. The relational classification task can be cast as a link prediction problem by adding the classes as nodes to the graph. The link prediction task is then optimized via the RESCAL tensor factorization model (Nickel, Tresp, and Kriegel, 2011).

Label propagation assumes that connected nodes have the tendency to belong

to the same class. Link prediction can therefore be used to reweight the graph to reinforce this tendency. Methods to reweight the graph range from introducing new edges based on random walk with restart (Gallagher, Tong, et al., 2008) to using an edge scoring function such as common neighbors count and jaccard index (Perez-Sola and Herrera-Joancomarti, 2013) to down-weighting edges that cross community boundaries based on an edge-betweenness score (Bangcharoensap et al., 2016). The approach closest to ours learn latent note attributes via matrix factorization. A loss combining the classification and matrix factorization problem is optimized via conjugate gradient descent (D. Zhang et al., 2016).

8.7. Conclusion and Future Work

Our experiments have shown that latent node attributes can be used as features for relational classification. We have further demonstrated that our random search algorithm is able to achieve state-of-the-art performance on data sets such as IMDb. The overall results are promising, but require further investigation to achieve top performance on all evaluated data sets. We have seen that the latent node attributes perform well independent of how many nodes have known labels.

Future Work Our experiments have demonstrated that using latent node attributes can sometimes produce better performance than label propagation. We plan to study situations in which label propagation fails, in order to understand if our approach can help is such situations. Another interesting research direction is to investigate what effects combining collective classification and link prediction into one optimization loss will have. We plan to use the classification loss as likelihood and the link prediction as prior. The goal of the prior is to regularize the latent features of the classifier.

Part IV

*fast*FM: a Factorization Machine Library

9. *fast*FM

This work aims to facilitate research for matrix factorization-based machine learning (ML) models. Factorization Machines are able to express many different latent factor models and are widely used for collaborative filtering tasks (Rendle, 2012a). An important advantage of FM is that the model equation

$$w_0 \in \mathbb{R}, x, w \in \mathbb{R}^p, v_i \in \mathbb{R}^k$$

$$\hat{y}^{FM}(x) := w_0 + \sum_{i=1}^{p} w_i x_i + \sum_{i=1}^{p} \sum_{j>i}^{p} \langle v_i, v_j \rangle x_i x_j \tag{9.1}$$

conforms to the standard notation for vector based ML. FMs learn a factorized coefficient $\langle v_i, v_j \rangle$ for each feature pair $x_i x_j$ (eq. 9.1). This makes it possible to model very sparse feature interactions, as for example, encoding a sample as $x = \{\cdots, 0, \overbrace{1}^{x_i}, 0, \cdots, 0, \overbrace{1}^{x_j}, 0, \cdots\}$ yields $\hat{y}^{FM}(x) = w_0 + w_i + w_j + v_i^T v_j$ which is equivalent to (biased) matrix factorization $R_{i,j} \approx b_0 + b_i + b_j + u_i^T v_j$ (Srebro, Rennie, and Jaakkola, 2004). (Please refer to Rendle, 2012a for more encoding examples.) FMs have been the top performing models in various machine learning competitions (Rendle and Schmidt-Thieme, 2009; Rendle, 2012c; Bayer and Rendle, 2013) with different objectives (e.g. the 'What Do You Know?' Challenge[1], EMI Music Hackathon[2]). *fast*FM includes solvers for regression, classification, and ranking problems (see Table 9.2) and addresses the following needs of the research community: (i) easy interfacing for dynamic and interactive languages such as R, Python and Matlab; (ii) a Python interface allowing interactive work; (iii) a publicly available test suite strongly simplifying modifications or the adding of new features; (iv) released under the **BSD-license** allowing the integration in (almost) any open source project.

[1] http://www.kaggle.com/c/WhatDoYouKnow
[2] http://www.kaggle.com/c/MusicHackathon

9.1. Design Overview

The *fast*FM library has a multi-layered software architecture (Figure 9.1) that separates the interface code from the performance-critical parts (*fast*FM-core). The core contains the solvers, is written in C, and can be used stand alone. Two user interfaces are available: a command line interface (CLI) and a Python interface. Cython (Behnel et al., 2011) is used to create a Python extension from the C library. Both the Python and C interfaces serve as reference implementation for bindings to additional languages.

*fast*FM (Py)	
Cython	CLI
*fast*FM-core (C)	

Table 9.1.: Library Architecture

9.1.1. fastFM-core

FMs are usually applied to very sparse design matrices (often with a sparsity over 95 %) due to their ability to model interaction between very high dimensional categorical features. We use the standard compressed row storage (CRS) matrix format as the underlying data structure and rely on the CXSparse[3] library (Davis, 2006) for fast sparse matrix/vector operations. This simplifies the code and makes memory sharing between Python and C straight-forward.
*fast*FM contains a test suite that is run on each commit to the GitHub repository via a continuous integration server[4]. Solvers are tested using state-of-the-art techniques, such as Posterior Quantiles (Cook, Gelman, and Rubin, 2006) for the MCMC sampler and Finite Differences for the SGD based solvers.

9.1.2. Solver and Loss Functions

*fast*FM provides a range of solvers for all supported tasks (Table 9.2). The MCMC solver implements the Bayesian Factorization Machine model (Freuden-

[3] CXSparse is LGPL licensed.
[4] https://travis-ci.org/ibayer/fastFM-core

thaler, Schmidt–Thieme, and Rendle, 2011) via Gibbs sampling. We use the pairwise Bayesian Personalized Ranking (BPR) loss (Rendle, Freudenthaler, Gantner, et al., 2009) for ranking. More details on the classification and regression solvers can be found in Rendle, 2012a.

Task	Solver	Loss
Regression	ALS, MCMC, SGD	Square Loss
Classification	ALS, MCMC, SGD	Probit (MAP), Probit, Sigmoid
Ranking	SGD	BPR (Rendle, Freudenthaler, Gantner, et al., 2009)

Table 9.2.: Supported solvers and tasks

9.1.3. Python Interface

The Python interface is compatible with the API of the widely-used `scikit-learn` library (Pedregosa et al., 2011), which opens the library to a large user base. The following code snippet shows how to use MCMC sampling for an FM classifier and how to make predictions with new data.

```
fm = mcmc.FMClassification(init_std=0.01, rank=8)
y_pred = fm.fit_predict(X_train, y_train, X_test)
```

*fast*FM provides additional features such as warm starting a solver from a previous solution (see MCMC example).

```
fm = als.FMRegression(init_std=0.01, rank=8, l2_reg=2)
fm.fit(X_train, y_train)
```

9.2. Experiments

libFM[5] is the reference implementation for FMs and the only one that provides an ALS and MCMC solver. Our experiments show that the ALS and MCMC

[5] http://libfm.org

solver in *fast*FM compare favorably to libFM with respect to runtime (Figure 9.1) and are indistinguishable in terms of accuracy. The experiments have been conducted on the MovieLens 10M data set using the original split with a fixed number of iterations (200) for all experiments. The x-axis indicates the number of latent factors (rank), and the y-axis the runtime in seconds. The plots show that the runtime scales linearly with the rank for both implementations. The code snippet below shows how simple it is to write Python code that

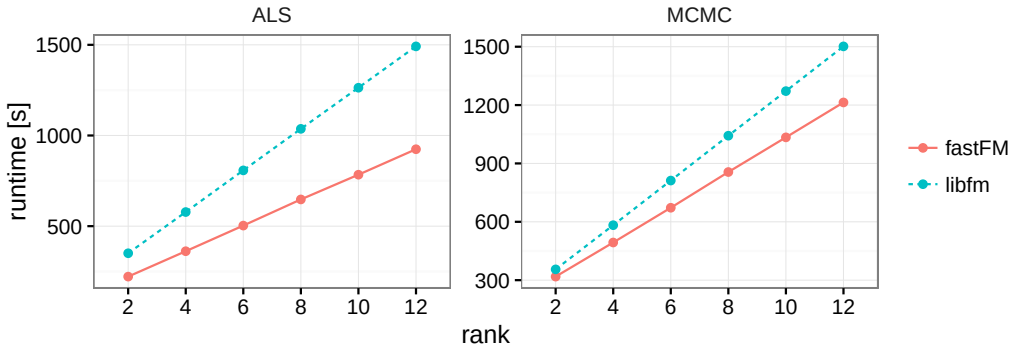


Figure 9.1.: A runtime comparison between *fast*FM and libFM is shown. The evaluation is done on the MovieLens 10M data set.

allows model inspection after every iteration. The induced Python function call overhead occurs only once per iteration and is therefore negligible. This feature can be used for Bayesian Model Checking as demonstrated in Figure 9.2. The figure shows MCMC summary statistics for the first order hyper-parameter σ_w. Please note that the MCMC solver uses Gaussian priors for the model parameter (Freudenthaler, Schmidt–Thieme, and Rendle, 2011).

```
fm = mcmc.FMRegression(n_iter=0)
# initialize coefficients
fm.fit_predict(X_train, y_train, X_test)

for i in range(number_of_iterations):
    y_pred = fm.fit_predict(X_train, y_train, X_test, n_more_iter=1)
    # save, or modify (hyper) parameter
    print(fm.w_, fm.V_, fm.hyper_param_)
```

Many other analyses and experiments can be realized with a few lines of Python code without the need to read or recompile the performance-critical C code.

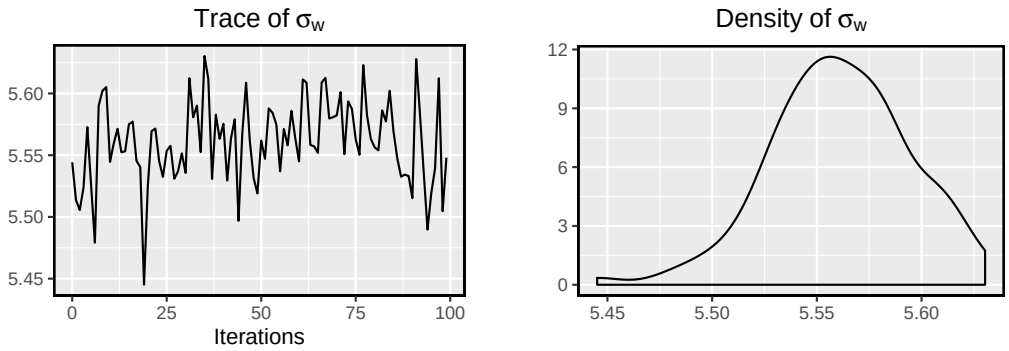


Figure 9.2.: MCMC chain analysis and convergence diagnostics example for the hyperparameter σ_w evaluated on the MovieLens 10M data set.

9.3. Related Work

Factorization Machines are available in the large-scale machine learning libraries GraphLab (Low et al., 2012) and Bidmach (Canny and Zhao, 2013). The toolkit Svdfeatures by T. Chen et al., 2012 provides a general MF model that is similar to a FM. The implementations in GraphLab, Bidmach and Svdfeatures only support SGD solvers and don't provide a ranking loss. It's not our objective to replace these distributed machine learning frameworks but to provide a FM implementation that is easy to use and easy to extend without sacrificing performance.

Part V

Conclusion

Contributions

This thesis has investigated the challenges and opportunities that relational data provide for predictive modeling. The following points summarize the contribution made while developing new approaches for link prediction and relational classification.

Modeling

- We have shown how to use a sequential Factorization Machine for time-dependent link prediction problems. This leads to greatly improved results over stationary time-series models (Chapter 4, 6).
- We have developed the IFM model which allows for the prediction of links between nodes that haven't been available at training time. This model naturally allows for the inclusion of node attributes into the modeling process (Chapter 5).
- We have identified a range of relational features that allow iid modeling for relational classification. These features allow the use of any iid classifier for relational classification, even when very few node labels are known (Chapter 7).
- We have presented an approach to use the IFM model for relational classification. This approach connects the link prediction and relational classification problem and allows to learn latent relational features (Chapter 8).

Algorithmic

To optimize the IFM model, we develop an efficient coordinate descent algorithm (Section 5) that exploits repeated structure in the objective function. This algorithm generalized the alternating least squares solver of Hu, Koren, and Volinsky, 2008. The algorithm optimizes an objective (Equation 5.1) over $|V|^2$ node pairs. For a sparse graph $|V|^2 \gg |E|$ the proposed algorithm can reduce the cost to optimize the objective from $\mathcal{O}(|V|^2)$ to $\mathcal{O}(E)$.

Experimental

- The sequential Factorization Machine model presented in Chapter 4 won the first (online) and second (offline) prize in the ECML PKDD Discovery Challenge 2013.

- *fast*FM, the scriptable Factorization Machine library presented in part IV, has been released as an open source project. The library allows the machine learning researcher to easily modify the learning process and model components. The project has an active and steadily growing user base (close to 300 bookmarks on GitHub) who contribute improvements and actively participate in discussion about the project.

A Computationally Efficient Prediction Model for Relational Data

This thesis has been guided by the goal of developing *a computationally efficient prediction model for relational data* that satisfies four requirements (Chapter 3). A short summary will now show that the IFM model fulfills those requirements.

1. The coordinate descent algorithm developed for the IFM model (Chapter 5) has a complexity of $\mathcal{O}(k|E|)$ where k is a parameter that defines the model complexity ($k \ll \max(|E|, |V|)$). The relational classification approach based on latent relational features (Chapter 8) also satisfies the scalability requirements with a complexity of $\mathcal{O}(k|V|)$.

2. The Implicit Factorization Machine model has shown state-of-the-art performance in our YouTube experiments. We have also shown that our relational features achieve state-of-the-art performance under conditions that are very favorable for the baseline model which we evaluate.

3. We have shown that the Implicit Factorization Machine model can learn latent relational features that can be used for relational classification. The model can therefore be used for link prediction and relational classification problems.

4. We have shown that the feature-based approach allows time-dependence to be modelled and additional information such as the textual node descriptions to be included without adjustments.

Future Work

The results presented in this thesis suggest the following promising directions for future work:

Link Prediction

- The coordinate descent solver developed for the IFM model can be extended to a Gibbs sampler as it was for the standard FM model (Freudenthaler, Schmidt–Thieme, and Rendle, 2011). Such a Bayesian IFM model can learn the parameter and hyper-parameter together and would allow for the investigation of graph properties with Bayesian Statistics. This would also allow the sampling of new graphs based on collected data.
- Our parameterization of the IFM model requires an in-going and out-going representation for each node to be learned. Coupling the in- and out-going representations of a node would allow for better modeling of un-directed graphs.
- The IFM can be modified to include sample weights. This would allow for the use of weighting schemes for negative samples (R. Pan and Scholz, 2009). This could be used to down-weight edges with certain characteristics such as a high betweeness score (Bangcharoensap et al., 2016).

Relational Classification

- Jointly learning the latent relational features together with the classifier would allow task specific node representations to be learned. This should reduce model size and improve prediction accuracy and would allow for the modeling of partially observed edges and node attributes together.

Bibliography

Adomavicius, Gediminas and Alexander Tuzhilin (2008). "Context-aware recommender systems." In: *RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems*. Lausanne, Switzerland: ACM, pp. 335–336 (cit. on p. 17).

Agarwal, Deepak and Bee-Chung Chen (2009). "Regression-based latent factor models." In: *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. Paris, France: ACM, pp. 19–28 (cit. on pp. 62, 73, 74).

Agarwal, Deepak and Bee-Chung Chen (2010). "fLDA: matrix factorization through latent dirichlet allocation." In: *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*. WSDM '10, pp. 91–100 (cit. on p. 62).

Ahmed, Amr et al. (2013). "Latent Factor Models with Additive and Hierarchically-smoothed User Preferences." In: *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*. WSDM '13. Rome, Italy: ACM, pp. 385–394 (cit. on p. 72).

Airoldi, Edoardo M et al. (2008). "Mixed membership stochastic blockmodels." In: *Journal of Machine Learning Research* 9.Sep, pp. 1981–2014 (cit. on p. 13).

Al Hasan, Mohammad et al. (2006). "Link prediction using supervised learning." In: *SDM 06: Workshop on Link Analysis, Counter-terrorism and Security* (cit. on p. 12).

Alvino, Chris and Justin Basilico (2015). *Learning a Personalized Homepage*. http://techblog.netflix.com/2015/04/learning-personalized-homepage.html. Blog (cit. on p. 61).

Amatriain, Xavier and Justin Basilico (2012). *Netflix Recommendations: Beyond the 5 stars*. http://techblog.netflix.com/2012/04/netflix-recommendations-beyond-5-stars.html. Blog (cit. on p. 61).

Baluja, Shumeet et al. (2008). "Video suggestion and discovery for youtube: taking random walks through the view graph." In: *Proceedings of the 17th international conference on World Wide Web*. ACM, pp. 895–904 (cit. on p. 10).

Bangcharoensap, Phiradet et al. (2016). "Transductive Classification on Heterogeneous Information Networks with Edge Betweenness-based Normalization." In: *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*. WSDM '16. San Francisco, California, USA: ACM, pp. 437–446 (cit. on pp. 109, 123).

Bayer, Immanuel and Steffen Rendle (2013). "Factor models for recommending given names." In: *Proceedings of the ECML-PKDD Discovery Challenge Workshop 2013*, pp. 81–89 (cit. on p. 113).

Bazire, Mary and Patrick Brézillon (2005). "Understanding context before using it." In: *International and Interdisciplinary Conference on Modeling and Using Context*. Springer, pp. 29–40 (cit. on p. 17).

Behnel, Stefan et al. (2011). "Cython: The best of both worlds." In: *Computing in Science & Engineering* 13.2, pp. 31–39 (cit. on p. 114).

Bennett, James and Stan Lanning (2007). "The Netflix Prize." In: *In KDD Cup and Workshop in conjunction with KDD* (cit. on p. 16).

Bergstra, James, Rémi Bardenet, et al. (2011). "Algorithms for hyper-parameter optimization." In: *Advances in Neural Information Processing Systems*, pp. 2546–2554 (cit. on p. 104).

Bergstra, James and Yoshua Bengio (2012). "Random search for hyper-parameter optimization." In: *Journal of Machine Learning Research* 13.Feb, pp. 281–305 (cit. on p. 104).

Bernstein, Abraham, Scott Clearwater, and Foster Provost (2003). "The relational vector-space model and industry classification." In: *IJCAI workshop*. Vol. 266 (cit. on pp. 14, 84, 87).

Bilgic, Mustafa and Lise Getoor (2008). "Effective label acquisition for collective classification." In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 43–51 (cit. on p. 101).

Bilgic, Mustafa, Galileo Mark Namata, and Lise Getoor (2007). "Combining collective classification and link prediction." In: *Seventh IEEE International Conference on Data Mining Workshops (ICDMW 2007)*. IEEE, pp. 381–386 (cit. on p. 108).

Blondel, Mathieu, Akinori Fujino, and Naonori Ueda (2015). "Convex factorization machines." In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, pp. 19–35 (cit. on p. 23).

Blondel, Mathieu, Akinori Fujino, Naonori Ueda, and Masakazu Ishihata (2016). "Higher-Order Factorization Machines." In: *Advances in Neural Information Processing Systems*, pp. 3351–3359 (cit. on p. 23).

Brandes, Ulrik et al. (2013). "What is network science?" In: *Network Science* 1.01, pp. 1–15 (cit. on p. 10).

Breese, John S., David Heckerman, and Carl Kadie (1998). "Empirical Analysis of Predictive Algorithms for Collaborative Filtering." In: *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98)*. San Francisco: Morgan Kaufmann, pp. 43–52 (cit. on p. 15).

Canny, John and Huasha Zhao (2013). "Bidmach: Large-scale learning with zero memory allocation." In: *BigLearn Workshop, NIPS* (cit. on p. 117).

Chakrabarti, Soumen, Byron Dom, and Piotr Indyk (1998). "Enhanced hypertext categorization using hyperlinks." In: *ACM SIGMOD Record* 27.2, pp. 307–318 (cit. on p. 89).

Chen, Tianqi et al. (2012). "SVDFeature: a toolkit for feature-based collaborative filtering." In: *The Journal of Machine Learning Research* 13.1, pp. 3619–3622 (cit. on p. 117).

Chin, Wei-Sheng et al. "An Efficient Alternating Newton Method for Learning Factorization Machines." In: *ACM Transactions on Intelligent Systems and Technology*. submitted (cit. on p. 23).

Cook, Samantha R, Andrew Gelman, and Donald B Rubin (2006). "Validation of software for Bayesian models using posterior quantiles." In: *Journal of Computational and Graphical Statistics* 15.3 (cit. on p. 114).

Crane, Robert and Luke McDowell (2012). "Investigating Markov Logic Networks for Collective Classification." In: *ICAART (1)*, pp. 5–15 (cit. on pp. 13, 88, 89).

Croft, W Bruce, Donald Metzler, and Trevor Strohman (2010). *Search engines: Information retrieval in practice*. Vol. 283. Addison-Wesley Reading (cit. on p. 70).

Davidson, James et al. (2010). "The YouTube Video Recommendation System." In: *Proceedings of the Fourth ACM Conference on Recommender Systems*. RecSys '10. Barcelona, Spain: ACM, pp. 293–296 (cit. on pp. 10, 61, 69, 74).

Davis, Timothy A (2006). *Direct methods for sparse linear systems*. Vol. 2. Siam (cit. on p. 114).

De Raedt, Luc (2008). *Logical and relational learning*. Springer Science & Business Media (cit. on p. 10).

Deshpande, M. and George Karypis (2004). "Item-based top-N recommendation algorithms." In: *ACM Transactions on Information Systems Springer-Verlag* 22/1 (cit. on p. 16).

Džeroski, Sašo (2009). *Relational data mining*. Springer (cit. on p. 10).

Freudenthaler, Christoph, Lars Schmidt–Thieme, and Steffen Rendle (2011). "Bayesian Factorization Machines." In: *NIPS workshop on Sparse Representation and Low-rank Approximation* (cit. on pp. 24, 114, 116, 123).

Friedman, Jerome H., Trevor Hastie, and Rob Tibshirani (2010). "Regularization Paths for Generalized Linear Models via Coordinate Descent." In: *Journal of Statistical Software* 33.1, pp. 1–22 (cit. on p. 23).

Gallagher, Brian and Tina Eliassi-Rad (2010). "Leveraging Label-Independent Features for Classification in Sparsely Labeled Networks: An Empirical Study." In: *ASONAM*. Ed. by Lee Giles et al. Springer, pp. 1–19 (cit. on p. 87).

Gallagher, Brian, Hanghang Tong, et al. (2008). "Using ghost edges for classification in sparsely labeled networks." In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 256–264 (cit. on pp. 87, 98, 109).

Getoor, Lise and Ben Taskar (2007). *Introduction to Statistical Relational Learning (Adaptive Computation and Machine Learning)*. The MIT Press (cit. on pp. 9, 10).

Graepel, Thore et al. (2010). "Web-scale bayesian click-through rate prediction for sponsored search advertising in microsoft's bing search engine." In: *ICML '10: Proceedings of the 27th international conference on Machine learning*, pp. 13–20 (cit. on p. 94).

Gunawardana, Asela and Christopher Meek (2009). "A unified approach to building hybrid recommender systems." In: *RecSys*, pp. 117–124 (cit. on p. 62).

Hong, Liangjie, Aziz S Doumith, and Brian D Davison (2013). "Co-factorization machines: modeling user interests and predicting individual decisions in twitter." In: *Proceedings of the sixth ACM international conference on Web search and data mining*. ACM, pp. 557–566 (cit. on p. 24).

Hu, Yifan, Yehuda Koren, and Chris Volinsky (2008). "Collaborative Filtering for Implicit Feedback Datasets." In: *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*. ICDM '08, pp. 263–272 (cit. on pp. 19, 63, 64, 69, 73, 121).

Jensen, David, Jennifer Neville, and Brian Gallagher (2004). "Why collective inference improves relational classification." In: *KDD*, pp. 593–598 (cit. on pp. 13, 82, 88).

Joachims, Thorsten (2002). "Optimizing search engines using clickthrough data." In: *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. Edmonton, Alberta, Canada: ACM, pp. 133–142 (cit. on p. 20).

Juan, Yuchin et al. "Field-aware factorization machines for CTR prediction." In: *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM, pp. 43–50 (cit. on p. 23).

Kanagal, Bhargav et al. (2012). "Supercharging Recommender Systems Using Taxonomies for Learning User Purchase Behavior." In: *Proceedings of the VLDB Endowment* 5.10, pp. 956–967 (cit. on p. 73).

Karatzoglou, Alexandros et al. (2010). "Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering." In: *RecSys '10: Proceedings of the fourth ACM conference on Recommender systems*. Barcelona, Spain: ACM, pp. 79–86 (cit. on p. 74).

Karypis, George and Vipin Kumar (1998). "A fast and high quality multilevel scheme for partitioning irregular graphs." In: *SIAM Journal on Scientific Computing* 20.1, pp. 359–392 (cit. on pp. 87, 89).

Koren, Yehuda (2008). "Factorization meets the neighborhood: a multifaceted collaborative filtering model." In: *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. Las Vegas, Nevada, USA: ACM, pp. 426–434 (cit. on p. 62).

Koren, Yehuda (2009a). "Collaborative filtering with temporal dynamics." In: *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. Paris, France: ACM, pp. 447–456 (cit. on pp. 17, 62).

Koren, Yehuda (2009b). "The BellKor Solution to the Netflix Grand Prize." In: *Netflix prize documentation* 81, pp. 1–10 (cit. on p. 16).

Koren, Yehuda and Robert Bell (2011). "Advances in Collaborative Filtering." In: *Recommender Systems Handbook*, pp. 145–186 (cit. on pp. 11, 62).

Koren, Yehuda, Robert Bell, Chris Volinsky, et al. (2009). "Matrix factorization techniques for recommender systems." In: *Computer* 42.8, pp. 30–37 (cit. on p. 11).

Kou, Zhenzhen and William W Cohen (2007). "Stacked Graphical Models for Efficient Inference in Markov Random Fields." In: *Proceedings of the SIAM International Conference on Data Mining (SDM-2007)*. SIAM, pp. 533–538 (cit. on pp. 13, 88).

Li, Hang, Tie-Yan Liu, and ChengXiang Zhai (2009). "Learning to Rank for Information Retrieval (LR4IR 2009)." In: *SIGIR Forum* 43.2, pp. 41–45 (cit. on pp. 19, 20).

Li, Mu et al. (2016). "DiFacto: Distributed Factorization Machines." In: *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*. ACM, pp. 377–386 (cit. on p. 24).

Liben-Nowell, David and Jon Kleinberg (2003). "The Link Prediction Problem for Social Networks." In: *Proceedings of the Twelfth International Conference on Information and Knowledge Management*. CIKM '03. New Orleans, LA, USA: ACM, pp. 556–559 (cit. on p. 12).

Lichtenwalter, Ryan N., Jake T. Lussier, and Nitesh V. Chawla (2010). "New Perspectives and Methods in Link Prediction." In: *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '10. Washington, DC, USA: ACM, pp. 243–252 (cit. on p. 12).

Linden, Greg, Brent Smith, and Jeremy York Amazon. Com (2003). "Industry report: Amazon.com recommendations: Item-to-item collaborative filtering." In: *IEEE Distributed Systems Online* (cit. on pp. 61, 69).

Linden, Greg, Brent Smith, and Jeremy York (2003). "Amazon.com Recommendations: Item-to-Item Collaborative Filtering." In: *IEEE Internet Computing* 7.1, pp. 76–80 (cit. on pp. 10, 74).

Liu, Jun, Jianhui Chen, and Jieping Ye (2009). "Large-scale sparse logistic regression." In: *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pp. 547–556 (cit. on p. 94).

Lops, Pasquale, Marco De Gemmis, and Giovanni Semeraro (2011). "Content-based recommender systems: State of the art and trends." In: *Recommender systems handbook*. Springer, pp. 73–105 (cit. on p. 11).

Low, Yucheng et al. (2012). In: *Proceedings of the VLDB Endowment* 5.8, pp. 716–727 (cit. on p. 117).

Lü, Linyuan and Tao Zhou (2011). "Link prediction in complex networks: A survey." In: *Physica A: Statistical Mechanics and its Applications* 390.6, pp. 1150–1170 (cit. on p. 12).

Lu, Qing and Lise Getoor (2003). "Link-based classification." In: *ICML '03: Proceedings of the twenty-first international conference on Machine learning*. Vol. 3, pp. 496–503 (cit. on pp. 14, 81, 87, 89).

Macskassy, Sofus A. (2007). "Improving learning in networked data by combining explicit and mined links." In: *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*. Vol. 22, pp. 590–595 (cit. on p. 89).

Macskassy, Sofus A. and Foster Provost (2003). "A Simple Relational Classifier." In: *Proceedings of the Multi-relational Data Mining Workshop ACM SIGKDD* (cit. on pp. 81, 89).

Macskassy, Sofus A. and Foster Provost (2007). "Classification in networked data: A toolkit and a univariate case study." In: *The Journal of Machine Learning Research* 8, pp. 935–983 (cit. on pp. 82, 87, 89, 90).

McDowell, Luke, Kalyan Moy Gupta, and David W Aha (2009). "Cautious collective classification." In: *The Journal of Machine Learning Research* 10, pp. 2777–2836 (cit. on pp. 13, 88, 98).

Menon, Aditya Krishna and Charles Elkan (2011). "Link prediction via matrix factorization." In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, pp. 437–452 (cit. on pp. 11–13).

Mitzlaff, Folke and Gerd Stumme (2012). "Namelings - Discover Given Name Relatedness Based on Data from the Social Web." In: *SocInfo*. Ed. by Karl Aberer et al. Vol. 7710. Lecture Notes in Computer Science. Springer, pp. 531–534 (cit. on p. 35).

Mukherjee, Indraneel et al. (2013). "Parallel boosting with momentum." In: *ECML PKDD*. Springer, pp. 17–32 (cit. on p. 94).

Murphy, Kevin P (2012). *Machine learning: A probabilistic perspective*. The MIT Press (cit. on p. 82).

Neville, Jennifer and David Jensen (2000). "Iterative classification in relational data." In: *Proceedings AAAI-2000 Workshop on Learning Statistical Models from Relational Data*, pp. 13–20 (cit. on p. 81).

Neville, Jennifer and David Jensen (2003). "Collective Classification with Relational Dependency Networks." In: *Proceedings of the second Workshop on Multi-relational Data Mining KDD* (cit. on pp. 13, 88).

Neville, Jennifer, David Jensen, L. Friedland, et al. (2003). "Learning Relational Probability Trees." In: *KDD '03: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining* (cit. on pp. 14, 84, 87).

Neville, Jennifer, David Jensen, and Brian Gallagher (2003). "Simple Estimators for Relational Bayesian Classifiers." In: *Proceedings of the Third IEEE International Conference on Data Mining*. ICDM '03. Washington, DC, USA: IEEE Computer Society, pp. 609–612 (cit. on pp. 14, 84, 87).

Nichols, David M (1998). "Implicit rating and filtering." In: *In Proceedings of the Fifth DELOS Workshop on Filtering and Collaborative Filtering* (cit. on p. 18).

Nickel, Maximilian, Volker Tresp, and Hans-Peter Kriegel (2011). "A three-way model for collective learning on multi-relational data." In: *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp. 809–816 (cit. on p. 108).

Nowicki, Krzysztof and Tom A B Snijders (2001). "Estimation and prediction for stochastic blockstructures." In: *Journal of the American Statistical Association* 96.455, pp. 1077–1087 (cit. on p. 13).

Oard, Douglas W, Jinmook Kim, et al. (1998). "Implicit feedback for recommender systems." In: *Proceedings of the AAAI workshop on recommender systems*, pp. 81–83 (cit. on p. 18).

Oentaryo, Richard J et al. (2014). "Predicting response in mobile advertising with hierarchical importance-aware factorization machine." In: *Proceedings of the 7th ACM international conference on Web search and data mining*. ACM, pp. 123–132 (cit. on p. 24).

Palmisano, Cosimo, Alexander Tuzhilin, and Michele Gorgoglione (2008). "Using Context to Improve Predictive Modeling of Customers in Personalization Applications." In: *IEEE Transactions on Knowledge and Data Engineering* 20.11, pp. 1535–1549 (cit. on p. 11).

Pan, Jia-Yu et al. (2004). "Automatic multimedia cross-modal correlation discovery." In: *Proceedings of the 10th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pp. 653–658 (cit. on p. 85).

Pan, Rong and Martin Scholz (2009). "Mind the gaps: weighting the unknown in large-scale one-class collaborative filtering." In: *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. Paris, France: ACM, pp. 667–676 (cit. on pp. 19, 123).

Pan, Rong, Yunhong Zhou, et al. (2008). "One-Class Collaborative Filtering." In: *IEEE International Conference on Data Mining (ICDM 2008)*, pp. 502–511 (cit. on p. 19).

Panniello, Umberto et al. (2009). "Experimental comparison of pre- vs. post-filtering approaches in context-aware recommender systems." In: *RecSys '09: Proceedings of the third ACM conference on Recommender systems*. New York, New York, USA: ACM, pp. 265–268 (cit. on p. 17).

Park, Seung-Taek and Wei Chu (2009). "Pairwise Preference Regression for Cold-start Recommendation." In: *Proceedings of the Third ACM Conference on Recommender Systems*. RecSys '09. New York, New York, USA: ACM, pp. 21–28 (cit. on p. 74).

Pedregosa, F. et al. (2011). "Scikit-learn: Machine Learning in Python." In: *The Journal of Machine Learning Research* 12, pp. 2825–2830 (cit. on pp. 89, 115).

Perez-Sola, Cristina and Jordi Herrera-Joancomarti (2013). "Improving relational classification using link prediction techniques." In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, pp. 590–605 (cit. on p. 109).

Perlich, Claudia and Foster Provost (2003). "Aggregation-based feature invention and relational concept classes." In: *Proceedings of the 9th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pp. 167–176 (cit. on pp. 14, 84, 87).

Perlich, Claudia and Foster Provost (2006). "Distribution-based aggregation for relational learning with identifier attributes." In: *Machine Learning* 62.1-2, pp. 65–105 (cit. on pp. 14, 84, 87).

Preisach, Christine and Lars Schmidt-Thieme (2006). "Relational Ensemble Classification." In: *Proceedings of the 6th IEEE International Conference on Data Mining*, pp. 499–509 (cit. on p. 87).

Rendle, Steffen (2012a). "Factorization Machines with libFM." In: *ACM Transactions on Intelligent Systems and Technology* 3.3, p. 57 (cit. on pp. 20, 22, 37, 38, 46, 49–51, 63, 113, 115).

Rendle, Steffen (2012b). "Learning recommender systems with adaptive regularization." In: *Proceedings of the fifth ACM international conference on Web search and data mining*. ACM, pp. 133–142 (cit. on p. 24).

Rendle, Steffen (2012c). "Social Network and Click-through Prediction with Factorization Machines." In: *KDD-Cup Workshop* (cit. on p. 113).

Rendle, Steffen (2013). "Scaling Factorization Machines to Relational Data." In: *Proceedings of the 39th international conference on Very Large Data Bases*. Trento, Italy (cit. on pp. 24, 94).

Rendle, Steffen, Dennis Fetterly, et al. (2016). "Robust Large-Scale Machine Learning in the Cloud." In: *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: ACM, pp. 1125–1134 (cit. on p. 23).

Rendle, Steffen and Christoph Freudenthaler (2014). "Improving Pairwise Learning for Item Recommendation from Implicit Feedback." In: *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*. WSDM '14. New York, New York, USA: ACM, pp. 273–282 (cit. on pp. 20, 63, 74).

Rendle, Steffen, Christoph Freudenthaler, Zeno Gantner, et al. (2009). "BPR: Bayesian Personalized Ranking from Implicit Feedback." In: *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*. UAI '09. Montreal, Quebec, Canada: AUAI Press, pp. 452–461 (cit. on pp. 20, 35, 40, 46, 63, 73, 115).

Rendle, Steffen, Christoph Freudenthaler, and Lars Schmidt-Thieme (2010). "Factorizing Personalized Markov Chains for Next-Basket Recommendation." In: *WWW '10: Proceedings of the 19th international conference on World wide web*. Raleigh, North Carolina, USA: ACM, pp. 811–820 (cit. on pp. 35, 37, 40, 62, 68, 72, 74).

Rendle, Steffen and Lars Schmidt-Thieme (2009). "Factor Models for Tag Recommendation in BibSonomy." In: *Proceedings of the ECML-PKDD Discovery Challenge Workshop* (cit. on p. 113).

Rendle, Steffen et al. (2011a). "Fast Context-aware Recommendations with Factorization Machines." In: *Proceedings of the 34th ACM SIGIR Conference on Reasearch and Development in Information Retrieval*. ACM (cit. on p. 17).

Rendle, Steffen et al. (2011b). "Fast context-aware recommendations with factorization machines." In: *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*. ACM, pp. 635–644 (cit. on p. 74).

Richardson, Matthew and Pedro Domingos (2006). "Markov logic networks." In: *Machine Learning* 62, pp. 107–136 (cit. on pp. 10, 13, 88).

Saar-Tsechansky, Maytal and Foster Provost (2007). "Handling missing values when applying classification models." In: *The Journal of Machine Learning Research* (cit. on p. 87).

Salakhutdinov, Ruslan and Andriy Mnih (2008). "Probabilistic matrix factorization." In: *Advances in neural information processing systems* 20, pp. 1257–1264 (cit. on p. 17).

Sarwar, Badrul et al. (2001). "Item-based collaborative filtering recommendation algorithms." In: *Proceedings of the 10th international conference on World Wide Web*. Hong Kong, Hong Kong: ACM Press, pp. 285–295 (cit. on p. 16).

Sculley, D. et al. (2014). "Machine Learning: The High Interest Credit Card of Technical Debt." In: *SE4ML: Software Engineering for Machine Learning (NIPS 2014 Workshop)* (cit. on pp. 63, 74).

Sen, Prithviraj et al. (2008). "Collective Classification in Network Data." In: *AI Magazine* 29.3, pp. 93–106 (cit. on pp. 10, 13, 81, 88, 97).

Srebro, Nathan, Jason Rennie, and Tommi S Jaakkola (2004). "Maximum-margin matrix factorization." In: *Advances in neural information processing systems*, pp. 1329–1336 (cit. on p. 113).

Taskar, Ben, Eran Segal, and Daphne Koller (2001). "Probabilistic classification and clustering in relational data." In: *IJCAI*. Vol. 17, pp. 870–878 (cit. on pp. 13, 81, 88).

Tong, Hanghang, Christos Faloutsos, and Jia-Yu Pan (2006). "Fast Random Walk with Restart and Its Applications." In: *Proceedings of the Third IEEE International Conference on Data Mining*, pp. 613–622 (cit. on p. 86).

Weston, Jason, Samy Bengio, and Nicolas Usunier (2010). "Large Scale Image Annotation: Learning to Rank with Joint Word-Image Embeddings." In: *European Conference on Machine Learning* (cit. on p. 63).

Weston, Jason, Samy Bengio, and Nicolas Usunier (2011). "Wsabie: Scaling Up To Large Vocabulary Image Annotation." In: *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI* (cit. on pp. 63, 73).

Xiang, Rongjing and Jennifer Neville (2011). "Understanding propagation error and its effect on collective classification." In: *Proceedings of the Third IEEE International Conference on Data Mining*. IEEE, pp. 834–843 (cit. on pp. 13, 88).

Ye, Mao et al. (2011). "Exploiting Geographical Influence for Collaborative Point-of-interest Recommendation." In: *Proceedings of the 34th Interna-*

tional ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR '11. Beijing, China: ACM, pp. 325–334 (cit. on p. 17).

Zhang, Daokun et al. (2016). "Collective Classification via Discriminative Matrix Factorization on Sparsely Labeled Networks." In: *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM, pp. 1563–1572 (cit. on p. 109).

Zhou, Tao, Linyuan Lü, and Yi-Cheng Zhang (2009). "Predicting missing links via local information." In: *The European Physical Journal B* 71.4, pp. 623–630 (cit. on p. 12).