University of Zurich^UZH

Heterogeneous Information Sources for Recommender Systems

Bachelor Thesis June 30, 2016

Marco Unternährer
of Romoos, LU, Switzerland

Student-ID: 07-118-771
marco.unternaehrer@uzh.ch

Advisor: **Bibek Paudel**

Prof. Abraham Bernstein, PhD
Institut für Informatik
Universität Zürich
http://www.ifi.uzh.ch/ddis

Acknowledgements

I would like to express my gratitude to Prof. Abraham Bernstein for giving me the opportunity to write this thesis. In particular I would like to thank my advisor Bibek Paudel for his great effort and patience. He has guided me well, provided me with many valuable suggestions and carefully reviewed this work. I am also very grateful for the enduring support throughout my studies of my family and friends.

Zusammenfassung

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Nullam a tellus. Aliquam commodo dui non ipsum. Duis mollis nisi id turpis. Donec quis ipsum. Curabitur sed nibh. Morbi suscipit justo quis orci. Ut massa tortor, ultricies vitae, lacinia eu, facilisis eu, nisl. Nulla mattis urna sed metus imperdiet ornare. Praesent sodales. Etiam laoreet. Mauris quam magna, sagittis et, pharetra eget, congue vitae, arcu. Fusce sollicitudin justo. Suspendisse lectus. Sed lobortis dolor quis lectus scelerisque ornare. Integer purus. Phasellus vel elit at nibh sagittis lobortis. Aliquam iaculis malesuada eros. Mauris metus

Abstract

Recommendation systems have become omni-present: we read online news articles, buy books from e-commerce websites and watch videos from online streaming services, which are all very likely to be suggested to us by a recommendation system. The most common applied algorithm utilizes the collaborative filtering technique, which makes only use of the user-item rating matrix. This thesis introduces two approaches, which make use of extra data about the items. One of our proposed models, MPCFs-SI, is based on a nonlinear matrix factorization model for collaborative filtering (MPCFs), and uses the extra data to regularize the model. The second model we propose, MFNN, is an ensemble of a matrix factorization and a neural network and makes direct use of the extra data. We show that MPCFs-SI slightly outperforms baseline recommender on a subset of both MovieLens 100k and 1M datasets. Our second model MFNN is always performing worse that the MPCFs model.

Table of Contents

1 Introduction

Recommender systems are software systems suggesting items to users which might fit their personal tastes. Such systems play a crucial part in e-commerce, social networks, news, music and video websites.

This work investigates methods which make use of data about the users, the items and ratings.

In Chapter 2 we present two ways of incorporating extra data into recommender systems. The evaluation of the proposed methods are compared with state-of-the-art methods in Chapter 3. The first section of this chapter gives an a brief overview over different methods for recommending systems. We further discuss related work in Section 1.2 and conclude this chapter with our motivation for this work.

1.1 Background

Methods which try to predict valuable items for a given user can be broadly classified into two groups: collaborative filtering (CF) based methods and content based methods. These methods differ in what kind of input data they need. Collaborative filtering methods make use of the user-item rating matrix in order to build models [Kabbur and Karypis, 2014]. Among all the CF algorithms, the most successful ones are the latent factor models [Bao et al., 2014]. These models try to explain user ratings by characterizing both items and users on factors inferred from the rating patterns [Bao et al., 2014]. Content based methods on the other hand make use of meta data about the items as well as the users. A combination of several methods is referred to as hybrid recommender systems [Ricci et al., 2011]. The purpose of such hybrid systems is to use the advantages of one method and fix the disadvantages of another [Ricci et al., 2011].

While CF techniques are very popular and widely used, one of its main disadvantages is known as the cold start problem [Christoffel, 2014]. Collaborative filtering methods require new users to rate some items before the system can suggest items, and new items need to be rated before it can be recommended to users. In case of the cold start problem, the rating history on a user or item is not sufficient to generalize well to future preferences. Often though, other sources of information provide richer cues on cold start users or items.

1.2 Related Work

In this section, we review related approaches to our work.

Simultaneously considering the ratings and the accompanied review text was proposed in [Bao et al., 2014]. The presented model, called **TopicMF**, learns a recommender model by combining latent factor modeling of rating data with topic modeling in user-review texts. [Bao et al., 2014] show on 22 real-world datasets that their method handles the data sparsity better than state-of-the-art latent factor models.

[Almahairi et al., 2015] also have shown that utilizing additional data as a way of regularizing the derived item representations improve the generalization performance of a rating prediction model. While [Bao et al., 2014] have used topic modeling for the user-review text, [Almahairi et al., 2015] proposed a distributed bag-of-word approach **BoWLF** and a recurrent neural network approach **LMLF** to model the user-review texts.

1.3 Motivation

As we have discussed in the previous section, recent work has shown that better rating predictions can be obtained by incorporating text-based side information. Motivated by these recent successes, here we explore alternative approaches to exploiting side information for top-N recommendations. Specifically, we study how feature vectors extracted from movie subtitles can be used to improve the performance of a latent factor model.

We introduce two approaches to incorporate side information into recommender and compare these to state-of-the-art-based approaches [Kabbur, 2015, Ning and Karypis, 2011, Rendle et al., 2009].

2

Incorporating Extra Data into Recommender

In this work we focus in particular on nonlinear methods for collaborative filtering. After giving an overview of the used notation in the first section, we show a recently developed nonlinear matrix factorization technique as proposed in [Kabbur, 2015] as part of Section 2.2. Section 2.3 introduces two models which make use of extra data. We conclude this chapter with an outline of our software setup in Section 2.4.

2.1 Notation

As in [Kabbur, 2015], all vectors are represented by bold lower case letters, all matrices are represented by bold upper case letters and predicted values are denoted by having a ˆ (hat) over it. All important symbols and definitions used in this work is summarized in Table 2.1.

2.2 Nonlinear Matrix Factorization for Collaborative Filtering

[Kabbur, 2015] developed a nonlinear matrix factorization method called **MPCF**, which models the user as a combination of global preference and interest-specific latent factors. In their work, given a user u, an item i and T user local preferences, the estimated rating $\hat{r}_{ui}$ is given by the sum of the estimations from global preference and interest-specific preference components. That is,

$$\hat{r}_{ui} = \mu + b_i + \mathbf{p}_u \mathbf{q}_i^\intercal + \max_{t=1,..,T} (b_{ut} + f(u,i,t)), \tag{2.1}$$

where μ is the global bias i.e., the average rating value of the entire data set, b_i is the item bias corresponding to item i, b_{ut} is the user-local preference bias corresponding to user u and local preference t, $\mathbf{p}_u$ is the latent vector associated with user u and $\mathbf{q}_i$ is the latent vector associated with item i [Kabbur, 2015]. [Kabbur, 2015] proposed two different methods to represent the interest-specific preference component *f(u, i, t)*. The

Symbol	Definition
$u,\ i$	Individual user u, item i
$n,\ m$	Number of users and items
k	Number of latent factors
T	Number of user local preferences
r_{ui}	Rating by user u on item i
$\hat{r}_{ui}$	Predicted rating for user u on item i
$\mathbf{R}$	User-item rating matrix, $\mathbf{R} \in \mathbb{R}^{n \times m}$
$\mathbf{P}$	User latent factor matrix, $\mathbf{P} \in \mathbb{R}^{n \times k}$
$\mathbf{Q}$	Item latent factor matrix, $\mathbf{Q} \in \mathbb{R}^{m \times k}$
$\mathbf{S}$	User latent factor tensor, $\mathbf{S} \in \mathbb{R}^{n \times k \times T}$
$\mathbf{B_u}$	User-Interest bias matrix, $\mathbf{B_u} \in \mathbb{R}^{n \times T}$
$\mathbf{b_i}$	Item bias vector, $\mathbf{b_i} \in \mathbb{R}^{m}$
λ_{reg}	l_2-regularization constant
d	Number of feature dimensions
$\mathbf{f}_i$	Feature vector for item i, $\mathbf{f}_i \in \mathbb{R}^{d}$
$\hat{\mathbf{f}}_i$	Predicted feature vector, $\hat{\mathbf{f}}_i \in \mathbb{R}^{d}$
$\mathbf{G}$	Weight matrix of the feature prediction model, $\mathbf{G} \in \mathbb{R}^{d \times k}$
$\mathbf{h}$	Bias vector of the feature prediction model, $\mathbf{h} \in \mathbb{R}^{d}$
λ_{ed}	Parameter balances matrix factorization model and the feature prediction model
λ_{cos}	Parameter balances the Euclidean and the cosine distance
$\hat{r}_{ui}^{\mathbf{MF}}$	Predicted rating for user u on item i of the matrix factorization model
$\hat{e}_{ui}$	Predicted error
$\mathbf{w}, \mathbf{W}'$	Weight vector and matrix of the neural network
$\mathbf{x}$	Input vector to the neural network, $\mathbf{x} \in \mathbb{R}^{2k+d}$
$\varphi(x)$	Activation function of the neural network
λ_{nn}	Parameter balances matrix factorization model and error prediction model

Table 2.1: Symbols used and definitions

first one, named **MPCFi**, has independent item factors in *f(u, i, t)* compared to that of global preference component, whereas the second one shares the item factors of *f(u, i, t)* with the global preference compontent, and thus named **MPCFs**. The local preference component *f(u, i, t)* for MPCFs is given by,

$$f(u, i, t) = \mathbf{s}_{ut}\mathbf{q}_i^\intercal, \tag{2.2}$$

where $\mathbf{s}_{ut}$ is the user latent vector for u in the local preference component corresponding to the local preference t and $\mathbf{q}_i^\intercal$ is the shared item latent vector between the global preference and the local preference components [Kabbur, 2015]. The following regularized optimization problem is minimized to learn $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{S}$, $\mathbf{B_u}$ and $\mathbf{b_i}$ matrices and vectors:

$$\mathcal{L}_{rating} = \frac{1}{2}\sum_{u,i \in R}(r_{ui} - \hat{r}_{ui})^2 + \frac{\lambda_{reg}}{2}(\|\mathbf{P}\|_F^2 + \|\mathbf{Q}\|_F^2 + \|\mathbf{S}\|_F^2 + \|\mathbf{B_u}\|_F^2 + \|\mathbf{b_i}\|_2^2), \tag{2.3}$$

where r_{ui} is the ground truth value, $\hat{r}_{ui}$ is the estimated value, $\mathbf{b_i}$ is the vector corresponding to the item biases, $\mathbf{B_u}$ is the matrix corresponding to the local preference user biases and λ_{reg} is the l_2-regularization constant for latent factor matrices [Kabbur, 2015].

2.3 Using Extra Data

In this section, we will show how feature vectors were extracted from movie subtitles, then propose two approaches to incorporate such vectors in order to improve top-N recommendations.

2.3.1 Feature Extraction from Movie Subtitles

We have downloaded the movie subtitles from a website called Opensubtitles[1]. Next, a pre-processing step was removing all non-alphanumerical characters which gave us a natural language document per subtitle with an average of 7521 words per document. With the help of a library called NLTK[2], each document was tokenized and stop words were removed. Gensim[3], a topic modelling library, was then used to generate a feature vector per document via a deep learning algorithm called Paragraph Vectors introduced in [Le and Mikolov, 2014]. The distributed-bag-of-words variant of the Paragraph Vector algorithm and negative sampling was used. We leave it to [Rong, 2014] and [Goldberg and Levy, 2014] in their excellent notes to explain in detail how Word2Vec, the common name of the base algorithm of Paragraph Vectors, works.

2.3.2 MPCFs-SI: Regularizing with Extra Data

[Almahairi et al., 2015] have shown that utilizing additional data as a way of regularizing the derived item representations improve the generalization performance of a rating

[1]Opensubtitles - http://opensubtitles.org
[2]NLTK - http://www.nltk.org/
[3]Gensim - https://radimrehurek.com/gensim/

prediction model. In a similar fashion, we propose to optimize a joint model, which consists of the nonlinear matrix factorization model MPCFs in Eq. 2.3 and a model which predicts the feature vector of an item $\hat{\mathbf{f}}_i$, given the latent factors for that item $\mathbf{q}_i$. The feature prediction is modeled as a simple affine transformation of the item latent factors, as shown in Eq.(2.4):

$$\hat{\mathbf{f}}_i = \mathbf{G}\mathbf{q}_i + \mathbf{h}, \tag{2.4}$$

where $\hat{\mathbf{f}}_i$ is the predicted feature vector for item i, $\mathbf{G}$ and $\mathbf{h}$ are a weight matrix and a bias vector to be learned. Since we are transforming from one vector space into another, we have decided to use the squared Euclidean and cosine distances as part of the loss term. We can informally think of minimizing the Euclidean distance as preserving the magnitude of the vector, while minimizing the cosine distance helps with the vector direction. Therefore, the loss term for the feature prediction model is given in Eq. 2.5:

$$\mathcal{L}_{ed} = \sum_{u,i \in R} (\|\mathbf{f}_i - \hat{\mathbf{f}}_i\|_F^2 + \lambda_{cos} D_C(\mathbf{f}_i, \hat{\mathbf{f}}_i)) + \lambda_{reg} \|\mathbf{G}\|_F^2, \tag{2.5}$$

where $\mathbf{f}_i$ is a feature vector for item i extracted from movie subtitles via methods discussed in the previous section, λ_{cos} is a parameter that balances the performance of the squared Euclidean distance and the cosine distance, $D_C(\mathbf{f}_i, \hat{\mathbf{f}}_i)$ is the cosine distance and λ_{reg} is the l_2-regularization constant for the weight matrix. The cosine similarity S_C and the cosine distance D_C are defined as,

$$S_C(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\|\|\mathbf{b}\|} \in [-1, 1] \tag{2.6}$$

$$D_C(\mathbf{a}, \mathbf{b}) = 1 - S_C(\mathbf{a}, \mathbf{b}). \tag{2.7}$$

We have also considered modeling the feature prediction with a neural network. Preliminary results have shown that a neural network model does not improve our metrics.

Therefore, the full loss term to optimize is the following:

$$\begin{aligned} \mathcal{L} &= \mathcal{L}_{rating} + \lambda_{ed} \mathcal{L}_{ed} \\ &= \frac{1}{2} \sum_{u,i \in R} (r_{ui} - \hat{r}_{ui})^2 + \frac{\lambda_{reg}}{2} (\|\mathbf{P}\|_F^2 + \|\mathbf{Q}\|_F^2 + \|\mathbf{S}\|_F^2 + \|\mathbf{B_u}\|_F^2 + \|\mathbf{b_i}\|_2^2) \\ &+ \lambda_{ed} (\sum_{u,i \in R} (\|\mathbf{f}_i - \hat{\mathbf{f}}_i\|_F^2 + \lambda_{cos} D_C(\mathbf{f}_i, \hat{\mathbf{f}}_i)) + \lambda_{reg} \|\mathbf{G}\|_F^2), \end{aligned} \tag{2.8}$$

where λ_{ed} is a parameter that balances the performance of the matrix factorization model and the feature prediction model.

We apply AdaGrad, a stochastic gradient descent variant, to update $\mu, \mathbf{P}, \mathbf{Q}, \mathbf{S}, \mathbf{B_u}$ and $\mathbf{b_i}$. The gradients for the matrix factorization model are directly from [Kabbur, 2015]. We have implemented the feature prediction model in Theano and update $\mathbf{G}$ and $\mathbf{h}$ with AdaGrad as well. Theano offers automatic differentiation such that we simply add $\lambda_{ed} \frac{\partial \mathcal{L}_{ed}}{\partial \mathbf{q}_i}$ to the item factor gradient $\frac{\partial \mathcal{L}_{rating}}{\partial \mathbf{q}_i}$.

As [Kabbur, 2015] suggest, gradient updates for model parameters are computed for both rated and non-rated entries of $\mathbf{R}$, which is common practice for the top-N recommendation task. We sample zero entries corresponding to non-rated items on every epoch and add these to the actual training ratings. [Kabbur, 2015] indicate that the ratio between rated and non-rated in the range of 1:3 to 1:5 is sufficient to produce the best model.

When we use the introduced feature prediction model together with the nonlinear matrix factorization model MPCFs, we call this joint model **MPCFs-SI**.

2.3.3 MFNN: Neural Network

The second model we propose is an ensemble of a matrix factorization model and a neural network model. We call this joint model **MFNN**. It is inspired by the MPCFs model and has the benefit of being able to directly incorporate additional user or item data as input to the neural network.

The estimated rating $\hat{r}_{ui}$ is given by:

$$\hat{r}_{ui} = \hat{r}_{ui}^{\mathbf{MF}} + \hat{e}_{ui}, \tag{2.9}$$

$$\hat{r}_{ui}^{\mathbf{MF}} = \mu + b_i + \mathbf{p}_u \mathbf{q}_i^{\intercal}, \tag{2.10}$$

where $\hat{r}_{ui}^{\mathbf{MF}}$ is the predicted rating of the matrix factorization model, and $\hat{e}_{ui}$ is the predicted error modeled in a neural network. The actual error and the predicted error are defined as,

$$e_{ui} = r_{ui} - \hat{r}_{ui}^{\mathbf{MF}}, \tag{2.11}$$

$$\hat{e}_{ui} = \sum_j (w_j \varphi(\sum_l w'_{j,l} x_l)), \tag{2.12}$$

$$\mathbf{x} = (\mathbf{p}_u, \mathbf{q}_i, \mathbf{f}_i), \tag{2.13}$$

$$\varphi(x) = max(0, x), \tag{2.14}$$

where $w_.$ and $w'_{.,.}$ are weights of the neural network model to be learned, $\mathbf{x}$ is a concatenation of the user latent factors $\mathbf{p}_u$, the item latent factors $\mathbf{q}_i$ and the side information as a feature vector $\mathbf{f}_i$, j is the number of neurons in the hidden layer, l is the length of $\mathbf{x}$ and $\varphi(x)$ is the activation function. Figure 2.1 shows the neural network in a diagram.

The loss term for this joint model is as follows:

$$\begin{aligned}\mathcal{L} = \frac{1}{2} \sum_{u,i \in R} (r_{ui} - \hat{r}_{ui}^{\mathbf{MF}})^2 + \frac{\lambda_{reg}}{2}(\|\mathbf{P}\|_F^2 + \|\mathbf{Q}\|_F^2 + \|\mathbf{b_i}\|_2^2) \\ + \frac{\lambda_{nn}}{2}(\sum_{u,i \in R} (e_{ui} - \hat{e}_{ui})^2 + \lambda_{reg}(\|\mathbf{w}\|_F^2 + \|\mathbf{W}'\|_F^2)),\end{aligned} \tag{2.15}$$

where λ_{nn} is a parameter that balances the performance of the matrix factorization model and the error prediction of the neural network model.

As in the previous model MPCFs-SI, we apply AdaGrad to update $\mu, \mathbf{P}, \mathbf{Q}$, and $\mathbf{b_i}$. The neural network model is implemented in Theano and its parameters get updated via AdaGrad. The sampling of non-rated items is done the same way as in our first model.

2.4 System Setup

We have built a custom software in Python. The two models just introduced as well as the baseline recommender MPCFs [Kabbur, 2015] and **BPRMF** [Rendle et al., 2009] were implemented by us. For the baseline recommender **SLIM** [Ning and Karypis, 2011], we were able use parts of an open source implementation by Mendeley[4]. Data handling was done with the Pandas library and we have used Numpy for linear algebra operations. As explained in Section 2.3.1, feature vectors for the movies were extracted from their subtitles with the help of Gensim[5] and NLTK[6]. All neural network related functionalities were implemented with Lasagne[7] and Theano[8]. The source code for this work can be downloaded under https://github.com/marcuniq/bsc-thesis.

[4]Mendeley mrec - https://github.com/Mendeley/mrec
[5]Gensim - https://radimrehurek.com/gensim/
[6]NLTK - http://www.nltk.org/
[7]Lasagne - https://github.com/Lasagne/Lasagne
[8]Theano - http://deeplearning.net/software/theano/

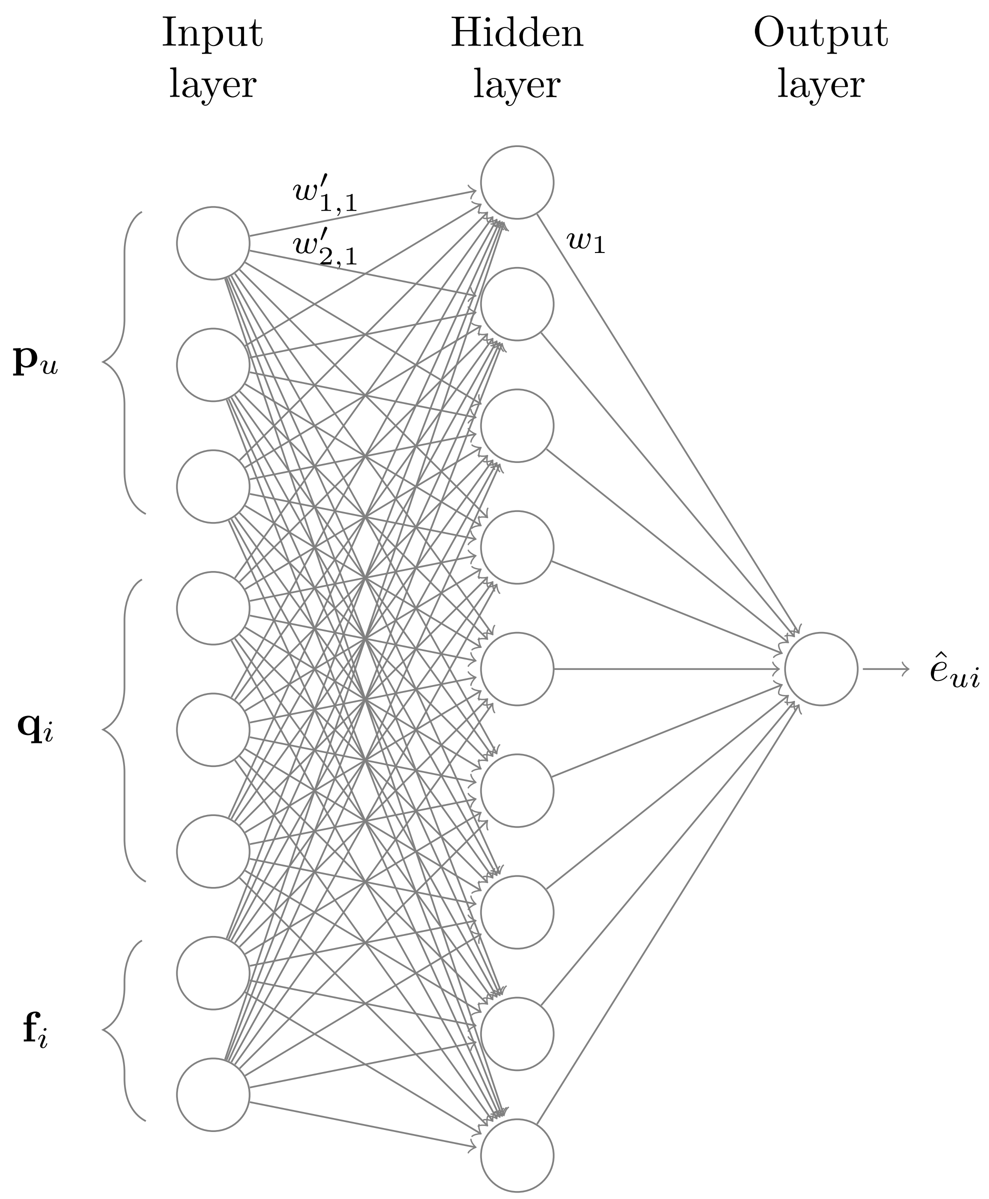

Figure 2.1: MFNN: Neural Network

3

Evaluation

In the previous chapter, we introduced two approaches to incorporate side information into recommender. The first section of this chapter gives an overview of the used datasets (Section 3.1), followed by the baseline recommender (Section 3.2) and the evaluation methodology (Section 3.3). We then state which performance metrics were calculated (Section 3.4). Section 3.5 concludes this chapter by showing how the proposed models perform compared to state-of-the-art models.

3.1 Datasets

Our datasets are based on the MovieLens 100k and the MovieLens 1M datasets[1]. We could only include ratings of movies for which we found the movie subtitles, which were downloaded from opensubtitles[2]. The datasets have rating values from 1 to 5. Converting the ratings into implicit feedback by setting the positive ratings to 1 was applied as it is commonly done for the top-N recommendation task [Kabbur, 2015]. Table 3.1 shows the number of ratings and movies we could preserve in our subsets.

Dataset	**# Ratings**	**# Movies**	**# Users**
MovieLens 100k	100'000	1682	943
Our ML-100k subset	91'178	1261	943
MovieLens 1M	1'000'000	3883	6040
Our ML-1M subset	978'026	3005	6040

Table 3.1: Datasets

3.2 Baseline

We compare the two proposed approaches, MPCFs-SI (Section 2.3.2) and MFNN (Section 2.3.3), against three baseline methods. The baseline recommender consist of the

[1]MovieLens Datasets - http://grouplens.org/datasets/movielens/
[2]Opensubtitles - http://opensubtitles.org

nonlinear matrix factorization model MPCFs discussed in Section 2.2, a linear recommender called SLIM [Ning and Karypis, 2011], and another state-of-the-art recommender also based on a latent factor model obtained with matrix factorization, called BPRMF [Rendle et al., 2009].

3.3 Methodology

The standard way of splitting datasets for evaluating recommendation systems is that a percentage of each user's ratings remain in the training set while the rest goes into the test set. However, we have subdivided our datasets, such that 70% of each movie's ratings was used for training, and the rest for testing. The idea behind this way of splitting the data is that we wanted to make the movie data especially sparse. We have also made sure, that each user has at least one rating in the test set in order to measure the performance. The performance of the methods are evaluated using five different randomly drawn splits in a manner just described.

We report the metrics after 20 iterations for MPCFs, MPCFs-SI and BPRMF. The number of iterations for the SLIM model was a hyperparameter. We have found that 5 epochs give us the best results. Our model MFNN is computationally quite expensive and takes a lot of time. We had to limit the number of iterations of that model for the MovieLens 1M dataset to 10 epochs.

3.4 Performance Metrics

We use area under the ROC curve (AUC) as our main metric to evaluate each approach. Moreover, we have adopted several commonly used metrics to measure the performance of the recommender. The following metrics were calculated: area under the ROC curve (AUC), Precision@20 (P@20), Recall@20 (R@20), Mean Reciprocal Rank (MRR) and Spearman Rank Correlation (SRC). Further, we have measured an additional metric related to AUC: Instead of recommending items to users and calculate the ranking metric, we were recommending users to items and calculating the rank metric for each item. We call this metric IAUC.

3.5 Performance Comparison

We have used the metrics defined in Section 3.4 to determine the performance of the models introduced in Section 2.3 and the baseline recommender. The results are summarized in Table 3.2. MPCFs-SI performs slightly better on both MovieLens datasets than the best baseline recommender, which is MPCFs. Our second model MFNN is inferior to MPCFs on both datasets.

We also show in Table 3.3 that similar movies have a high cosine similarity (defined in Eq. 2.6) in the document vector space. In fact, the sequals to the listed movies were the most similar or second most similar movie in that space. Moreover, when we look at the item factor vector space, the results show that using MPCFs-SI improves the cosine similarity of almost all listed items compared to using MPCFs. The best performing models on the MovieLens 1M dataset were used to calculate the cosine similarities in Table 3.3.

	Method	**AUC**	**IAUC**	**P@20**	**R@20**	**MRR**	**SRC**
ML-100k	Perfect	100.0	100.0	100.0	100.0	1.0	1.0
	MPCFs-SI	93.76	90.83	28.16	44.27	0.6866	0.2005
	MFNN	93.55	89.50	28.88	44.45	0.6850	0.1927
	MPCFs	93.65	90.78	27.79	43.55	0.6770	0.1974
	BPRMF	92.20	68.03	18.00	27.38	0.3703	0.1567
	SLIM	91.45	85.20	23.49	37.75	0.5993	0.1880
ML-1M	Perfect	100.0	100.0	100.0	100.0	1.0	1.0
	MPCFs-SI	92.88	92.15	32.11	32.77	0.6941	0.2260
	MFNN						
	MPCFs	92.81	92.08	32.71	33.44	0.7035	0.2223
	BPRMF	91.82	67.54	21.23	20.13	0.4430	0.2013
	SLIM	91.23	88.00	28.59	28.76	0.6465	0.2322

Table 3.2: Performance metrics

Movie	**Sequal**	**Doc2Vec**	**MPCFs**	**MPCFs-SI**
Free Willy	Free Willy 2	0.8694	0.6923	0.7051
Jurassic Park	Jurassic Park 2	0.6796	0.5837	0.6171
Scream	Scream 2	0.7514	0.6868	0.7233
Species	Species II	0.6831	0.6170	0.7064
Star Wars V	Star Wars VI	0.9231	0.6900	0.7221
Toy Story	Toy Story 2	0.7529	0.6696	0.6678

Table 3.3: Item cosine similarities, measured in the document vector space and the item factor vector space of the according model.

4

Conclusions

In this work we have developed two models (MPCFs-SI and MFNN) which incorporate feature vectors of side information in order to improve top-N recommendations. While similar approaches like [Bao et al., 2014] and [Almahairi et al., 2015] have shown improvement in rating prediction on almost all of their datasets, we were able to outperform the baseline recommender MPCFs on top-N recommendations only slightly on our two MovieLens datasets. Our second model MFNN always performed worse than the baseline recommender.

The learning scheme we employed consisted of two steps: first extract feature vectors from movie subtitles via the well established algorithm of Paragraph Vectors [Le and Mikolov, 2014], then using those feature vectors in our models. Future research efforts should attempt to fully joint learn a matrix factorization model and a movie subtitle model.

References

[Almahairi et al., 2015] Almahairi, A., Kastner, K., Cho, K., and Courville, A. (2015). Learning Distributed Representations from Reviews for Collaborative Filtering. *Proceedings of the 9th ACM Conference on Recommender Systems - RecSys '15*, pages 147–154.

[Bao et al., 2014] Bao, Y., Hui, F., and Zhang, J. (2014). TopicMF: Simultaneously Exploiting Ratings and Reviews for Recommendation. *Aaai*, pages 2–8.

[Christoffel, 2014] Christoffel, F. (2014). *Recommending Long-Tail Items with Short Random Walks over the User-Item-Feedback Graph.* PhD thesis.

[Goldberg and Levy, 2014] Goldberg, Y. and Levy, O. (2014). word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method. (2):1–5.

[Kabbur, 2015] Kabbur, S. (2015). *Machine Learning Methods for Recommender Systems.* PhD thesis, University of Minnesota.

[Kabbur and Karypis, 2014] Kabbur, S. and Karypis, G. (2014). NLMF: NonLinear Matrix Factorization Methods for Top-N Recommender Systems. *2014 IEEE International Conference on Data Mining Workshop*, (ii):167–174.

[Le and Mikolov, 2014] Le, Q. and Mikolov, T. (2014). Distributed Representations of Sentences and Documents. *International Conference on Machine Learning - ICML 2014*, 32:1188–1196.

[Ning and Karypis, 2011] Ning, X. and Karypis, G. (2011). SLIM: Sparse LInear Methods for top-N recommender systems. *Proceedings - IEEE International Conference on Data Mining, ICDM*, pages 497–506.

[Rendle et al., 2009] Rendle, S., Freudenthaler, C., Gantner, Z., and Schmidt-thieme, L. (2009). BPR : Bayesian Personalized Ranking from Implicit Feedback. *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, cs.LG:452–461.

[Ricci et al., 2011] Ricci, F., Rokach, L., and Shapira, B. (2011). Introduction to Recommender Systems Handbook. In *Recommender Systems Handbook*, chapter Introducti, page 842. Springer US.

[Rong, 2014] Rong, X. (2014). word2vec Parameter Learning Explained. pages 1–20.

A

Appendix

A.1 Best Parameters

A.1.1 MPCFs-SI

MovieLens 100k

Matrix factorization model:
learning rate: 0.01, learning rate decay: 0.02, number of epochs: 20, number of latent factors k: 128, number of user local preferences T: 4, regularization parameter λ_{reg}: 0.01, zero sample factor: 5
Feature prediction model:
learning rate: 0.03, learning rate decay: 0.02, regularization parameter λ_{reg}: 0.01, balancing parameter λ_{ed}: 0.001, balancing parameter λ_{cos}: 0.7

MovieLens 1M

Matrix factorization model:
learning rate: 0.03, learning rate decay: 0.02, number of epochs: 20, number of latent factors k: 96, number of user local preferences T: 2, regularization parameter λ_{reg}: 0.01, zero sample factor: 5
Feature prediction model:
learning rate: 0.03, learning rate decay: 0.02, regularization parameter λ_{reg}: 0.01, balancing parameter λ_{ed}: 0.0003, balancing parameter λ_{cos}: 2

A.1.2 MFNN

MovieLens 100k

Matrix factorization model:
learning rate: 0.03, learning rate decay: 0.02, number of epochs: 20, number of latent factors k: 128, regularization parameter λ_{reg}: 0.01, zero sample factor: 5 Neural network model:
layer dimensions: [306, 4, 1], learning rate: 0.01, regularization parameter λ_{reg}: 0.01, balancing parameter λ_{nn}: 0.1

MovieLens 1M

Matrix factorization model:
learning rate: 0.06, learning rate decay: 0.02, number of epochs: 10, number of latent factors k: 128, regularization parameter λ_{reg}: 0.01, zero sample factor: 4 Neural network model:
layer dimensions: [306, 4, 1], learning rate: 0.003, regularization parameter λ_{reg}: 0.003, balancing parameter λ_{nn}: 0.3

A.1.3 MPCFs

MovieLens 100k

learning rate: 0.01, learning rate decay: 0.02, number of epochs: 20, number of latent factors k: 128, number of user local preferences T: 4, regularization parameter λ_{reg}: 0.01, zero sample factor: 5

MovieLens 1M

learning rate: 0.03, learning rate decay: 0.02, number of epochs: 20, number of latent factors k: 128, number of user local preferences T: 2, regularization parameter λ_{reg}: 0.01, zero sample factor: 5

A.1.4 BPRMF

MovieLens 100k

learning rate: 0.03, learning rate decay: 0.02, number of epochs: 20, number of latent factors: 128, regularization parameter: 0.01, triplet sample factor: 5

MovieLens 1M

learning rate: 0.03, learning rate decay: 0.02, number of epochs: 20, number of latent factors: 128, regularization parameter: 0.001, triplet sample factor: 5

A.1.5 SLIM

MovieLens 100k

number of epochs: 5, fit intercept: false, ignore negative weights: true, l_1 regularization: 0.003, l_2 regularization: 0.00003

MovieLens 1M

number of epochs: 5, fit intercept: true, ignore negative weights: true, l_1 regularization: 0.001, l_2 regularization: 0.003

List of Figures

List of Tables