SD6125 - Data Mining
Project Report for Recommender System

Chen Yuhao
College of Computing and Data Science
Nanyang Technological University
CH0009AO@e.ntu.edu.sg

Li Chenyang
College of Computing and Data Science
Nanyang Technological University
CLI034@e.ntu.edu.sg

Qian Yujie
College of Computing and Data Science
Nanyang Technological University
qian0097@e.ntu.edu.sg

Ruan Yalin
College of Computing and Data Science
Nanyang Technological University
YALIN002@e.ntu.edu.sg

Abstract

Recommender systems play a significant role in helping companies retain and expand their customer base. Due to the complexity of information and data in recommendation tasks, recommender systems have undergone rapid iterations and improvements. Various methods have been proposed to solve problems encountered in different practical application scenarios. In order to derive deeper comprehension and insights about recommender systems. This report focuses on carrying out a comparative analysis of various recommendation algorithms utilizing both memory-based and model-based collaborative filtering techniques by implementing and testing them respectively on the MovieLens-20M and Book-Crossing datasets. Results indicate notable performance variations across algorithms, influenced by data sparsity and user-item interactions. Our findings provide insights into how recommendation is made based on data and algorithms, also reflect on some objective problems that may occur in recommending processes.

1. Introduction

1.1. Background

In the era of "information explosion", how to improve user experience to promote user growth has become an important topic for countless companies, enterprises and service providers to expand market size and increase revenue. Before the concept of recommender system was proposed, information (products) could only reach users through random dissemination, resulting in a lot of information (products) being unable to accurately reach customers who really have corresponding needs. With the development of recommender system, the process of these information transmissions can continuously learn rules and methods from historical data, helping users navigate vast amounts of information, personalizing experiences by filtering and suggesting content that matches individual preferences.

In general, a recommender system is designed to analyze user behavior, preferences, and item characteristics to suggest items a user might like, enhancing their experience and helping them discover relevant content. It operates by gathering and processing data on user interactions and item features, then using this information to identify patterns in preferences. Target of the system is to retrieve a list of potential recommendations, which are ranked based on their relevance to the user. Many systems combine multiple approaches, leveraging both user and item information to improve accuracy. The result is a personalized experience where users are shown products, content, or services tailored to their tastes, making navigation simpler and more engaging.

1.2. Overview on Recommendation Approach

Recommendation algorithms can be broadly categorized into memory-based and model-based collaborative filtering (CF) methods. This report explores these two main types, highlighting their methodologies, advantages, and limitations.

1.2.1 Memory-based Approach

Memory-based approaches, including User-based Collaborative Filtering (UserCF) and Item-based Collaborative Filtering (ItemCF), rely on user and item similarities. UserCF recommends items to a user based on the preferences of

similar users. Herlocker [4] introduced algorithms that calculate user similarity using measures like Pearson correlation or cosine similarity. The main advantage of UserCF is its simplicity and interpretability. However, it can suffer from scalability issues and data sparsity, especially in systems with a large number of users. ItemCF which was proposed by Sarwar [8] recommends items similar to those the user has previously liked. It computes item similarity based on user ratings or interactions. ItemCF tends to provide more stable recommendations over time compared to UserCF and handles sparse data better. Nonetheless, it may not capture the dynamic preferences of users effectively.

1.2.2 Model-based Approach

Model-based CF methods, such as SVD, NMF, KNNWithMeans, and the DSSM model, utilize machine learning and neural networks to uncover deeper relationships in the data. They are straightforward and easy to implement but may struggle with sparsity and scalability in large systems. Besides, they can provide better accuracy and handle sparsity but at the expense of increased complexity and resource requirements. Roughly speaking, model-based methods can be divided into three categories:

- **Matrix Factorization (MF):** Techniques like Singular Value Decomposition (SVD) and Non-negative Matrix Factorization (NMF) decompose the user-item interaction matrix into lower-dimensional representations, see more details in [6]. MF models efficiently handle large, sparse datasets and capture underlying latent features influencing user preferences. The downside is that they require extensive computational resources and careful tuning of hyperparameters.

- **Clustering-Based Models:** These models group users or items into clusters based on similarity metrics, see more details in [11]. Clustering simplifies the recommendation process by treating all members of a cluster similarly, which enhances scalability. However, this approach can lead to loss of personalization and may not account for individual user nuances.

- **Latent Vector Representation Models:** Deep learning models like the Deep Structured Semantic Model (DSSM) map users and items into a latent semantic space to learn complex, non-linear relationships, see more details in [5]. These models can capture intricate patterns in the data and improve recommendation accuracy. Their disadvantages include high computational cost and the need for large amounts of training data to prevent overfitting.

1.3. Report content

In the following part of this report, we will compare these algorithms using two datasets: the MovieLens-20M dataset, which has extensive rating data on movies, and the Book-Crossing dataset, characterized by sparser user-book ratings. Our objectives are threefold: to analyze and compare the accuracy of different recommendation algorithms, to assess how data sparsity impacts performance, and to identify techniques that optimize predictions across datasets with varied user-item interaction density. By evaluating their performance using MAE and RMSE metrics, we aim to provide insights into their effectiveness under different data conditions and offer strategies for enhancing recommendation accuracy, particularly in scenarios with limited interaction data.

2. Methods

2.1. Memory-Based Collaborative Filtering

2.1.1 User-based Collaborative Filtering (User-based CF)

User-based collaborative filtering (User-based CF) algorithm makes recommendations based on the similarity between users. The algorithm first computes the similarity between the target user and other users, and then recommends items based on the behavior of users similar to the target users. The specific process is as follows:

Calculate user similarity: Calculate the similarity between the target user and other users using the user-item interaction matrix, where each row represents a user, each column represents an item, and the elements of the matrix are the interaction data between the user and the item. Common similarity calculation methods include cosine similarity, Pearson correlation coefficient, Jaccard similarity coefficient and so on. In our experiment, cosine similarity is used as the similarity measurement method:

$$sim(u,v) = \frac{\sum_{i \in I_{uv}} r_{ui} \cdot r_{uj}}{\sqrt{\sum_{i \in I_{uv}} r_{ui}^2} \cdot \sqrt{\sum_{i \in I_{uv}} r_{uj}^2}} \tag{1}$$

Here, u and v represent two users, r_{ui} and r_{vi} represent the ratings given by users u and v to item i respectively, and I_{uv} represents the set of items rated by both users u and v.

Select similar users as neighbors: Select the top N users that are most similar to the target user, based on the order of value of user similarity

Generate a recommendation list: Recommend items that similar users like but the target user has not yet contacted. The ratings of the target user for these items are predicted by the ratings of similar users for these items, that is, computes a weight average of the actual ratings of similar users for these items and the similarity between similar

users and target the user as the predicted ratings of target users for these items:

$$\hat{r}_{ui} = \frac{\sum_{v \in N(u)} sim(u,v) \cdot r_{vi}}{\sum_{v \in N(u)} |sim(u,v)|} \tag{2}$$

Here, $\hat{r}_{ui}$ represents the predicted rating of user u for item i, $N(u)$ is the set of users similar to user u, r_{vi} is the actual rating of user v for item i, and $sim(u,v)$ is the similarity between user u and user v.

2.1.2 Item-based Collaborative Filtering (Item-based CF)

Item-based collaborative filtering (Item-based CF) algorithm recommends items similar to the target user's favorite items by calculating the similarity between items. The specific process is as follows:

Calculate item similarity: Calculate the similarity between items using the user-item interaction matrix, which has been introduced in User-based Collaborative Filtering Algorithm. Similarly, cosine similarity, Pearson correlation coefficient, Jaccard similarity coefficient can be used as the similarity calculation methods. In our experiment, cosine similarity is used as the similarity measurement method:

$$\text{sim}(i,j) = \frac{\sum_{u \in U_{ij}} r_{ui} \cdot r_{uj}}{\sqrt{\sum_{u \in U_{ij}} r_{ui}^2} \cdot \sqrt{\sum_{u \in U_{ij}} r_{uj}^2}} \tag{3}$$

Here, i and j represent two items, r_{ui} and r_{uj} represent the ratings of user u on item i and item j respectively, and U_{ij} represents the set of users who have rated both items i and j.

Select similar items: Select the top N items that are most similar to the target item, based on the order of value of item similarity

Generate a recommendation list: Recommend items that are similar to the items that the target user has given a high rating. The recommendation ratings for these items can be predicted based on the target user's ratings of similar items, that is, computes a weight average of the actual ratings of the target users for rated items and the similarity between the rated items and the target item as the predicted ratings of target users for these items:

$$\hat{r}_{ui} = \frac{\sum_{j \in N(i)} sim(i,j) \cdot r_{uj}}{\sum_{j \in N(i)} |sim(i,j)|} \tag{4}$$

Here, $\hat{r}_{ui}$ represents the predicted rating of user u for item i, $N(i)$ is the set of items similar to item i, r_{uj} is the actual rating of user u for item j, and $sim(i,j)$ is the similarity between item i and item j.

2.2. Model-Based Collaborative Filtering

2.2.1 Singlular Value Decomposition (SVD)

SVD decomposes any matrix into singular vectors and singular values. If the reader has previous experience with machine learning, particularly with dimensionality reduction, they would find traditional use of SVD in Principal Component Analysis (PCA). Simply put, SVD is equivalent to PCA after mean centering, i.e. shifting all data points so that their mean is on the origin [10].

Formally, SVD is decomposition of a matrix R into the product of three matrices:

$$R_{m*n} \approx U_{m*m} D_{m*n} V^t_{n*n} \tag{5}$$

Where R_{m*n} denotes the utility matrix with n equal to the number of e.g. users and m number exposed items. U_{m*m} is a left singular orthogonal matrix, representing the relationship between users and latent factors [3]. D_{m*n} is a diagonal matrix (with positive real values) describing the strength of each latent factor. V^t_{n*n} is a right singular orthogonal matrix, indicating the similarity between items and latent factors.

The general goal of SVD (and other matrix factorization methods) is to decompose the matrix R with all missing r_{ij} and multiply its components $U_{m*m} D_{m*n} V^t_{n*n}$ once again. As a result, there are no missing values r_{ij} and it is possible to recommend each user items they have not seen or purchased yet.

2.2.2 Non-negative Matrix Factorization (NMF)

Non-negative Matrix Factorization (NMF) is very similar to SVD, which is also a matrix factorization technique. The main difference between is that NMF decomposes a non-negative data matrix into the product of two lower-rank non-negative matrices. Unlike SVD, which allows for negative elements in its factorization, NMF imposes non-negativity constraints, leading to a parts-based, additive representation of the data [7]. This property makes NMF particularly suitable for applications like recommender systems, where the utility matrix contains non-negative values.

Formally, NMF approximates the utility matrix R as the product of two non-negative matrices:

$$R_{m \times n} \approx W_{m \times k} H_{k \times n} \tag{6}$$

Where $R_{m \times n}$ is the utility matrix with m users and n items. $W_{m \times k}$ is the non-negative user-factor matrix, representing the association between users and latent factors. $H_{k \times n}$ is the non-negative factor-item matrix, representing the association between latent factors and items. k is the number of latent factors, typically chosen such that $k \ll \min(m,n)$.

The objective of NMF is to find W and H that minimize the difference between R and their product, usually measured using the Frobenius norm:

$$\min_{W,,H}, |R - WH|_F^2 \quad \text{s.t.} \quad W \geq 0, , H \geq 0 \tag{7}$$

By iteratively updating W and H while enforcing non-negativity, NMF uncovers latent structures within the data that are inherently interpretable due to their additive nature [10].

In the context of recommender systems, the factor matrices W and H capture user preferences and item attributes in a low-dimensional space. Once the matrices are learned, the missing entries r_{ij} in R can be approximated by the product $(WH)_{ij}$, allowing for predictions of how users might rate items they have not yet interacted with.

The non-negativity constraint ensures that all latent factors contribute positively, which aligns with real-world scenarios where negative contributions may not make sense (e.g., a user cannot have a negative preference for an item they have not rated).

2.2.3 K-Nearest Neighbors with Means (KNNWithMeans)

K-Nearest Neighbors with Means (KNNWithMeans) is a neighborhood-based collaborative filtering method that enhances the traditional K-Nearest Neighbors (KNN) algorithm by incorporating user and item mean ratings to adjust for individual rating biases [2]. This adjustment helps to mitigate the effects of users who consistently rate items higher or lower than the average, leading to more accurate and personalized recommendations.

In the context of recommender systems, KNNWithMeans predicts the rating $\hat{r}_{ui}$ that a user u would give to an item i by considering the ratings of the k most similar users (neighbors) to u . The prediction formula is given by:

$$\hat{r}_{ui} = \bar{r}_u + \frac{\sum v \in N_k(u) suv(r_{vi} - \bar{r}_v)}{\sum v \in N_k(u)|s_{uv}|} \tag{8}$$

Where $\bar{r}_u$ is the average rating given by user u. $\bar{r}_v$ is the average rating given by neighbor user v. $N_k(u)$ denotes the set of k nearest neighbors to user u. s_{uv} is the similarity between users u and v. r_{vi} is the rating that neighbor user v has given to item i.

The similarity s_{uv} between users can be computed using measures such as the Pearson correlation coefficient or cosine similarity, based on their ratings of common items [9].

By centering the ratings around each user's mean ($r_{vi} - \bar{r}_v$), KNNWithMeans accounts for the individual user's rating scale and tendencies. This adjustment helps in comparing user preferences more effectively, as it normalizes the differences in rating behaviors across users.

Similarly, for item-based collaborative filtering, the prediction formula adjusts for item mean ratings:

$$\hat{r}ui = \bar{r}i + \frac{\sum j \in N_k(i) sij(r_{uj} - \bar{r}j)}{\sum j \in N_k(i)|s_{ij}|} \tag{9}$$

Where $\bar{r}_i$ is the average rating received by item i. $\bar{r}_j$ is the average rating received by neighbor item j . $N_k(i)$ denotes the set of k nearest neighbors to item i . s_{ij} is the similarity between items i and j. r_{uj} is the rating that user u has given to item j.

The KNNWithMeans algorithm improves prediction accuracy by combining the simplicity of neighborhood-based methods with the effectiveness of mean-centering techniques [6]. It is particularly useful in datasets where rating scales vary significantly among users or items.

2.2.4 Deep Structured Semantic Model (DSSM)

The Deep Structured Semantic Model (DSSM), Also known as Dual-Tower Model, is a neural network model designed to improve query-document matching, particularly in search engines. Unlike traditional keyword-based matching, DSSM leverages deep learning to map queries and documents into a shared low-dimensional semantic space. This allows it to capture semantic similarities even if the keywords between the query and document don't match exactly.

DSSM works by using a deep neural network (DNN) to learn semantic features of queries and documents. First, each query and document is represented as a high-dimensional term vector, which encodes the occurrence of words in the text. These vectors are then projected through the DNN into a low-dimensional semantic space, where they are represented as "semantic feature vectors." The relevance between a query and a document is calculated by measuring the cosine similarity between their semantic vectors in this space.

The goal of DSSM is to maximize the conditional likelihood of clicked documents given a query. The model optimizes this using the following objective function:

$$P(d|q) = \frac{\exp(\gamma \cdot sim(q, d))}{\sum_{d' \in D} \exp(\gamma \cdot sim(q, d'))} \tag{10}$$

where $P(d|q)$ is the probability that document d is clicked given the query q, γ is a smoothing factor, D is the set of candidate documents, and $sim(q|d)$ denotes the cosine similarity between the query and document in the semantic space.

The main process involves three steps. First, the model converts text features into vectors, using techniques like word hashing to reduce dimensionality and computational

complexity. Then, these features are passed through multiple layers of the DNN to produce low-dimensional semantic feature vectors. Finally, it calculates similarity scores for ranking. By learning from click data, the DSSM enhances the relevance of documents that users are likely to find useful, providing a more effective solution for large-scale query-document matching tasks

2.3. Evaluation Metrics

2.3.1 Mean Absolute Error (MAE)

MAE (Mean Absolute Error) measures the average absolute difference between the predicted values and the ground-truth values. MAE is less sensitive to outliers compared to RMSE, making it ore robust when large errors are not common. The closer MAE is to 0, the better the prediction quality of the algorithm. The formula is:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{11}$$

Here, n is the total number of samples, y_i is the actual rating of the i-th sample, and $\hat{y}_i$ is the predicted rating of the i-th sample.

2.3.2 Root Mean Square Error (RMSE)

RMSE (Root Mean Square Error) measures the square root of the average squared difference between predicted and ground-truth values. RMSE penalizes big deviations between predicted and ground-truth ratings more strongly than MAE, making it more sensitive to outliers. Like MAE, the closer RMSE is to 0, the better the prediction quality of the algorithm. The formula is:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{12}$$

Here, n is the total number of samples, y_i is the actual rating of the i-th sample, and $\hat{y}_i$ is the predicted rating of the i-th sample.

3. Experiments

3.1. Data Preparation

3.1.1 Dataset Introduction

In recommender systems, datasets are commonly consisting of user-item interaction data, such as user clicks, ratings, purchases, etc.

To compare several different recommendation algorithms (ItemCF, UserCF, SVD, NMF, KNNWithMeans, DSSM), we require another type of traditional data type, a rating table, which, unlike clicks or purchases that cannot directly and intuitively reflect user preferences, provides a quantifiable measure of user inclination towards items. Rating data is relatively easier to process, making it suitable for our experimental purposes. Based on our experimental objectives, we selected two popular rating table datasets: **MovieLens-20M** dataset and amazon **Book-Crossing** dataset.

In general, the **MovieLens-20M** dataset includes:

- A ratings table with 4 columns: `userId`, `movieId`, `rating`, `timestamp`, containing 20,000,263 rating records, with ratings ranging from 0.5 to 5.
- A movies information table with 3 columns: `movieId`, `title`, and `genres`, where `genres` represents the movie's categories, stored in the format `genre1|genre2|...`.

The **Book-Crossing** dataset includes:

- A ratings table with 3 columns: `userId`, `book-ISBN`, `rating`, containing 1,149,780 rating records, with ratings ranging from 1 to 10.
- A user information table with 3 columns: `userId`, `Location`, `Age`.
- A book information table with 5 valid columns: `book-ISBN`, `title`, `author`, `year-of-Publication`, `Publisher`.

To reduce computational load, we needed to downsize the datasets by performing negative sampling based on user activity (i.e., users with more ratings) and item popularity (i.e., items with more ratings). We selected relatively active users and the rating data of the relatively popular items they interacted with.

This preprocessing ensures that the datasets are manageable for computational purposes while retaining sufficient data to evaluate and compare the performance of the selected recommendation algorithms.

3.1.2 Data Overview

Based on above criterion, we extracted subsets for the two datasets respectively, some basic statistical results are listed as in Tab. 1.

From the statistical results, we found that there existed a difference in the rating range between the two data sets, which, rating in **MovieLens-20M** ranged from 0.5 to 1, while that in **Book-Crossing** ranged from 1 to 10. Thus, in order to ensure the training and evaluation on the both datasets are consistent, we needed to apply a ***MinMaxScaler*** to uniform the ratings into a same range of from 0.5 to 5.

	MovieLens-20M	Book-Crossing
MAX_{rating}	5	10
MIN_{rating}	0.5	1
$MEAN_{rating}$	3.28	7.84
$\#_{records}$	3,794,270	124,710
$\#_{users}$	3,000	2,996
$\#_{items}$	25,511	25,000

Table 1. Statistical description of data subsets

At the same time, in order to better make the training and testing steps of the model independent, we follow the conventional train-test-split method, that is, using the random division method, we randomly select 80% of the entire rating table data set as the training set and the remaining 20% as the test set, and ensure that all our subsequent model are trained and tuned on the training set. Similarly, the following visualizations and Exploratory Data Analyses (EDA) in this "Data Overview" chapter are also based on the data from the training set.

In the process of processing data, we notice a phenomenon of data sparsity. This is because in an ideal state, we hope that the user-item historical interaction data based on model training can cover the ratings of all users to all items, that is, for n users and m items, $n * m$ rating data are expected to support a "perfect" model training. However, the amount of rating data that can be obtained in reality is often much smaller than this ideal value, which leads to the sparsity problem of user-item rating matrix, which will significantly affect the performance of the model in some algorithms, especially those collaborative filtering algorithms, as these methods rely on sufficient user-item interactions to make accurate predictions [1]. Because high levels of sparsity can lead to unreliable similarity computations and decreased model performance, particularly in capturing user preferences and item characteristics.

To analyze the extent of sparsity and its potential effects on our models, we conduct an EDA by visualizing the user-item rating matrices of our datasets to more intuitively observe the sparsity of the data. The raw rating data are transformed into matrices where rows represent users, columns represent items, and each cell r_{ui} contains the rating given by user u to item i. Value $0s$ in these matrices indicate that the user has not interacted with (rated) the item.

In the heatmaps: **Missing ratings** (i.e. cells with value $0s$) are depicted using a cool color, such as blue, indicating low interaction frequency.**Existing ratings:** (i.e. cells with ratings) are shown in warmer colors, with the color intensity proportional to the rating value—the higher the rating, the warmer the color (e.g., transitioning from blue to red).

This color gradient effectively highlights areas of high and low user-item interactions within the datasets. A predominance of cool colors (blue regions) signifies high sparsity, suggesting that most users have rated only a limited number of items.

From Fig. 1, we can see that both datasets are holding relatively large sparsity in the user-item rating matrix. By diving into deeper insights of the 2 heatmaps, we conclude that although the **MovieLens-20M** dataset is relatively large, with relatively sufficient records of ratings, the overall matrix is quite sparse. Most of the heatmap is blue, indicating that the majority of user-item pairs have no ratings. In contrast, the **Book-Crossing** dataset has significantly fewer reviews, which also leads to a more significant phenomenon of value $0s$ in the heatmap.

Through the preliminary conclusions obtained from above EDA work, we can carry out subsequent feature construction, processing and selection, model training, evaluation and comparison work in a more targeted manner.

3.1.3 Feature Engineering

For different algorithms and models, we use different feature engineering strategies.

First, for the **memory-based CF** model, the similarity calculation used to characterize product features and user preferences is a very important step. In actual industrial processes, this part of the work often needs to be calculated and stored in an offline platform from the user's historical behavior data obtained regularly. In real-time recommendations, it is only necessary to call the pre-calculated similarity between items or the similarity between users to calculate the estimated interest score, thereby achieving fast recommendations.

Therefore, we carried out the similarity calculation required in this step during the feature engineering process. Specifically, in order to implement **itemCF** and **userCF**, we need to calculate the item similarity matrix and the user similarity matrix respectively. The similarity calculation method adopts the cosine similarity calculation method shown in the previous mentioned Eq. (1) and Eq. (3). At the same time, in order to reduce the pressure on computer memory during calculation, we designed a mechanism for batch calculation of similarity based on matrix calculation, so that the similarity calculation process can be faster and more stable. In this part of feature engineering, we expected a input of the user-item rating matrix (with $n * m$ dimension , n is the number of users, m is the number of items), which will output the item similarity matrix (with $m * m$ dimension, a symmetric matrix) and the user similarity matrix (with $n * n$ dimension, a symmetric matrix).

Then, for the **model-based CF** model, we also generate different features for different types of algorithms. For **SVD**, **NFM** and **KNNWithMeans**, the model input are user's rating towards item, which is align with our raw data.

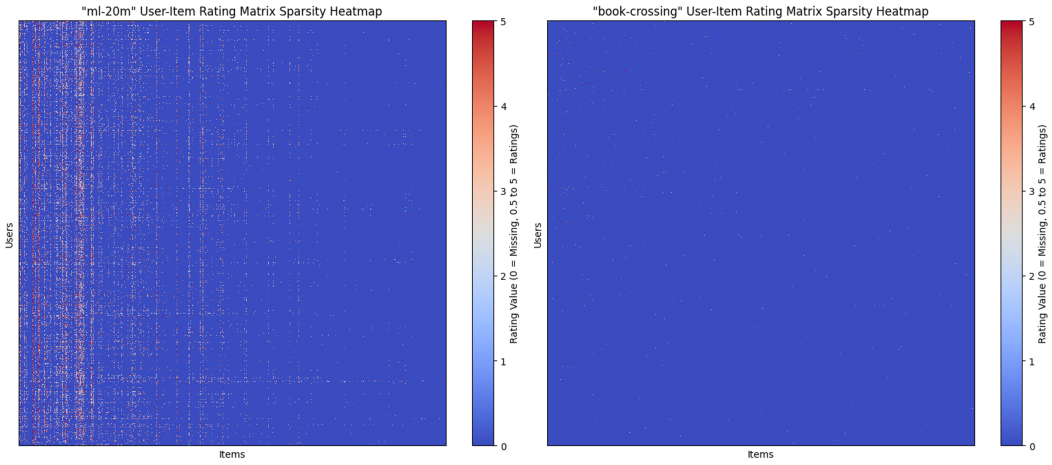

Figure 1. Sparsity heatmaps for both datasets

So we just need to load our data (with ratings already be scaled by ***MinMaxScaler***) with the pre-built python package ***surprise*** and identify the target of variable ratings.

Lastly, for neural-network-based **DSSM** model, we can generate more features to help building a more "information-rich" embeddings for both user side and item side according to some domain knowledge. In general, **DSSM** can capture information from 3 types of input data to initialize user and item embeddings: sparse features (or discrete features), dense features (continuous features), and sequence features (variable-length features). This inspires us to make better use of **external** data about users and items to construct features other than only depending on the rating table data. More specifically, we can combine external information about users and items to enhance initialization of user and item embedding representation. For example, use some text feature extraction methods (e.g. Tf-idf) to extract a feature vector of the movie's or book's title, search the user's history of interacting with items (items the user has already rated) and form a sequence record based on timestamp. So, in our experiment, we design different features for both datasets. For **MovieLens-20M** dataset, we first input user id and item id as basic sparse features. Also, in the dataset we have titles and genres of the movies, we construct a dense feature of tf-idf vector for each movie's title, limit the length to 100. And treat the genres feature as a variable-length feature, as for each movie may be consider to be in different genres. Also, we have the timestamp of user rating in this dataset, so we can generate a sequence for each rating record, containing what movies the user has already rated before current rating, which is also a variable-length feature. Additionally, we made some simple calculation on the user's mean rating and item's mean rating to get an average performance of each user and item, and this mean ratings are treated as dense features. For **Book-Crossing**, we also introduce user id and item id into the model input. Moreover, we have some population feature about the users and some book related features about the items. To be brief, the detailed features for each dataset is listed as suggest in Tab. 2.

3.2. Experiment Process

3.2.1 User-based CF, Item-based CF

Input

As mentioned in Data Preparation, since the original data set is too large, we finally selected the top 3,000 users who rated the most items, their rated items and corresponding ratings as the data set for our experiment. These users will be referred to as active users below. Due to the reduction of the data set, we use the collaborative filtering algorithm in the "Sorting" stage of the recommender system instead of the "Recall" stage, where it is used more commonly. Therefore, the experiment finally get a predicted rating set and a list of recommended items, which will be explained in detail in the section Output of this chapter.

Based on the results of Data Preparation, the following data is obtained: the complete set of active users' ratings on items, the training rating set and the rating test set obtained by dividing the complete set into a certain proportion, the user similarity matrix and the item similarity matrix calculated based on the complete set.

In the experiment, we load the training rating set, user similarity matrix and item similarity matrix as input, process them and use matrix dot multiplication to calculate the predicted rating. The detailed experimental design will be introduced in the section Algorithm Design of this chapter.

Algorithm Design

Load data: The training rating set and the similarity matrix files in parquet format are loaded by data frame format.

MovieLens-20M	Book-Crossing
'rating': target,	**'rating': target**,
'userId': sparse feature,	'userId': sparse feature,
'itemId': sparse feature,	'itemId': sparse feature,
'item mean rating': dense feature,	'item mean rating': dense feature,
'user mean rating': dense feature,	'user mean rating': dense feature,
'user hist': sequence feature,	'author': sparse feature,
'genres': sequence feature,	'year': sparse feature,
'year': sparse feature,	'publisher': sparse feature,
'title tfidf vector': dense feature,	'is erlier': sparse feature,
'is erlier': sparse feature,	'is 80s': sparse feature,
'is 80s': sparse feature,	'is 90s': sparse feature,
'is 90s': sparse feature,	'is 00s': sparse feature,
'is 00s': sparse feature,	'is latest': sparse feature,
'is latest': sparse feature,	'book age': dense feature,
'is comedy': sparse feature,	'location': dense feature,
'is romance': sparse feature,	'user age': dense feature,
'is action': sparse feature	'country': sparse feature,
	'generation': sparse feature,
	'book-user-age crossing':dense feature

Table 2. Features for DSSM

Reconstruct the similarity matrix: Considering that we use matrix dot multiplication to obtain the final predicted rating matrix, we need to ensure that the column index of the training rating matrix and the row index of the similarity matrix must match. Therefore, we retrieve all the user IDs and item IDs in the training rating matrix, sort them and save them into two arrays respectively, adjust the arrangement of rows and columns in the similarity matrix according to the order of these two arrays, and reconstruct the similarity matrix.

Execute Index mapping: Since the user IDs and item IDs involved in the training rating matrix and similarity matrix are all from the original data set, although the size of the data set used in the experiment is reduced, the value range of the user ID and item ID is not be reduced. When constructing a sparse matrix, due to the characteristics of its row and column indexes, the index values of the sparse matrix constructed with the original user ID and item ID will be very large. In order to facilitate subsequent matrix dot multiplication, it is necessary to construct an effective sparse matrix that meets the expected size. Therefore, two index mapping methods and corresponding reverse mapping methods are designed for the user ID and item ID respectively. Index mapping methods are used to convert the user ID and item ID into consecutive numbers starting from 0.

Convert to sparse matrix: Considering that even though the dataset of our experiment has been reduced compared to the original dataset, its data size is still huge. In order to facilitate subsequent matrix dot multiplication, we convert both the training rating matrix and the similarity matrix into coordinated sparse matrices and perform matrix dot multiplication in the form of row-compressed sparse matrices.

Predicted rating: The predicted rating matrix is obtained by dot multiplying the training rating sparse matrix by the similarity sparse matrix. Please note that in the user-based collaborative filtering algorithm, the training rating sparse matrix needs to be transposed first so that its rows represent items and its columns represent users, and then the dot product is used for calculation.

Execute Index reverse mapping: Convert the obtained prediction rating matrix into data frame form, and reverse the user IDs in the row indexes and and item IDs in the column indexes of the prediction ratings matrix to the original user IDs and item IDs through the index reverse mapping method. Please note that in the user-based collaborative filtering algorithm, the prediction rating matrix needs to be transposed first, so that its rows represent users and columns represent items, and then executed form transformation and index reverse mapping. Standardize predicted rating matrix: The obtained prediction rating matrix is standardized by being divided by the sum of the similarities between the user or item and all other users or items in the similarity matrix.

Save predicted rating matrix: Obtain the final predicted rating matrix and save it locally.

Generate recommendation list: Since we use the collaborative filtering algorithm in the "Sorting" stage of the recommender system, based on the obtained predicted rating

set, we can use a mask to cover some ratings which could be corresponded to an original true rating made by the same user with respect to the same item in the complete set of ratings. The processed predicted rating set could be seen as the pure predicted rating set of all users for items they have not contacted. Based on the processed predicted rating set, sort the predicted rating, select the top 100 rated items as recommended items, generate a recommendation list, and save it locally.

Output

Based on the results of the matrix dot multiplication, we can obtain a complete predicted rating matrix and save it locally in CSV format, which can be as the input for Experiment Evaluation phase.

Based on the predicted rating matrix , we can generate a recommendation list in dictionary format, with the key being the user ID and the value being an array of the item IDs of the corresponding recommended items. The recommendation list is saved locally in the pkl format, which could be regarded as the expected output in the "Sorting" stage of a recommender system usually.

Evaluation

MAE and RMSE are used as evaluation indicators in our experiment. For the collaborative filtering algorithm in our experiment, the evaluation process is as follows:

Load data: load the locally saved predicted ranting matrix and test rating set into data frame format.

Compute errors: define the error and square error arrays, find the corresponding predicted rating in the predicted rating matrix according to the user ID and item ID of each rating in the test rating set, calculate the error and square error of each pair of ratings, append them to the corresponding array, calculate the average of the values stored in the error array, which is MAE, and calculate the square root of the average of the values in the square error array, which is RMSE.

Output and save results: Print the calculated MAE and RMSE values to the console and save them locally in txt format.

Implementation

Python is used as programming language in the Collaborative Filtering experiment, and Python libraries such as numpy, pandas, pickle, and scipy are called to execute reading files, processing data, performing calculations, saving results and other operations. The experimental using collaborative filtering algorithm is implemented in the local device. The device is Lenovo-Y9000P, whose processor is 11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz

3.2.2 SVD, NMF, KNNWithMeans

For training **SVD**, **NMF**, **KNNWithMeans** models, the *surprise* package in python provides us with a convenient api to implement these recommender systems using collaborative filtering techniques. In addition, it provides several model evaluation metreics and allows for cross-validation to support hyperparameter tuning.

To optimize the performance of the models, we used the GridSearchCV function from the Surprise library. It is similar to the GridSearchCV function in *Scikit-learn* package, which enables an exhaustive search over specified hyperparameter values for an estimator, evaluating each combination using cross-validation. For each algorithm (SVD, NMF, KNNWithMeans), we defined a parameter grid and performed hyperparameter tuning using Root Mean Square Error (RMSE) and Mean Absolute Error (MAE) as the evaluation metrics.

GridSearchCV systematically explores different parameter combinations and returns the best-performing model configuration based on the evaluation criteria. This ensures that we identify the optimal set of parameters for each algorithm.

To perform GridSearchCV, we predefined the hyperparameter space for each algorithm as follows:

For the **Singular Value Decomposition (SVD)** model, we varied the number of latent factors with 3 levels $[50, 100, 150]$, learning rate with 2 levels $[0.005, 0.01]$, regularization with 2 levels $[0.02, 0.1]$, and the number of training epochs with 2 levels $[20, 30]$. For **Non-negative Matrix Factorization (NMF)** model, we tuned the number of latent factors for 3 levels $[15, 20, 25]$,epochs for 2 levels $[50, 100]$. For **KNNWithMeans** model, we varied the number of neighbors k for 3 levels $[10, 20, 30]$, the similarity calculation for 2 metrics $[\text{"}cosine\text{"}, \text{"}pearson\text{"}]$, and whether the algorithm is user-based or item-based (2 levels). The hyperparmeter space is set same for both **MovieLens-20M** and **Book-Crossing** datasets.

3.2.3 DSSM

In this experiment, the DSSM code is structured to develop a recommender system applied to the MovieLens and Book-Crossing datasets. The data preprocessing includes tailored handling of discrete, continuous, and variable-length features, converting them into formats compatible with deep learning and specific to recommendation tasks.

For discrete variables, the preprocessing step encodes these categorical values into unique integers, creating an indexed representation of each category. Continuous variables are normalized, often using techniques like min-max scaling, to ensure that they fall within a specific range and contribute proportionally within the model. For variable-length features, typically used for sequential data such as user interactions or item histories, the data is transformed into fixed-length sequences by padding. This ensures consistency in sequence length, allowing the DSSM to process

sequences as uniform input batches without losing critical ordering information.

After preprocessing, the DSSM is trained on these transformed inputs. The model learns to map user preferences and item attributes into a shared semantic space, optimizing the similarity between relevant user-item pairs. During evaluation, metrics such as MAE and RMSE are used to assess the accuracy of the recommendation rankings, allowing for performance tracking on both datasets. This approach ensures that the model is well-equipped to handle diverse data types while producing meaningful recommendations for both movies and books.

3.3. Results

3.3.1 User-based CF, Item-based CF

Four sets of evaluation metrics are obtained after the experiment of User-based CF and Item-based CF, which correspond to the evaluation results of rating prediction for the **MovieLens-20M** and **Book-Crossing** datasets using the **User-based CF** algorithm and the **Item-based CF** algorithm. The results are shown in Tab. 3.

As the result suggest, for the **MovieLens-20M** dataset, the **User-based CF** algorithm performs better with an RMSE of 2.58 and an MAE of 2.35, while the **Item-based CF** Algorithm obtained an RMSE of 2.94 and an MAE of 2.75. However, for the **Book-Crossing** dataset, there is not an obvious difference of prediction effect of these two algorithms, where the **User-based CF** algorithm retrieved an RMSE of 3.98 and an MAE of 3.88, and the **Item-based CF** Algorithm retireved an RMSE of 3.96 and an MAE of 3.86.

In conclusion, although the number of users and items obtained from the processed MovieLens-20m dataset and the processed Book-Crossing dataset are similar as mentioned in Data Preparation, the number of ratings in the former is much larger than that in the latter, which means that the user-item interaction matrix obtained from the former is much denser than that of the latter. Therefore, whether using the **User-based CF** algorithm or the **Item-based CF** algorithm, the prediction effect for the **MovieLens-20M** dataset will be better than that of the **Book-Crossing** dataset.

In the ideal condition, the prediction effect of Item-based CF is better than that of User-based CF, because the number of items is often less than the number of users, and user behavior may vary greatly , which makes the item similarity matrix denser than the user similarity matrix usually. However, in our experiment of User-based CF and Item-based CF, we have screened out active users when preprocessing the MovieLens-20m dataset, which means that the user similarity matrix is denser than the item similarity matrix based on this dataset, so the **User-based CF** algorithm has better prediction effect than the **Item-based CF** algorithm for the **MovieLens-20M** dataset. Whereas, although we also screened out active users when preprocessing the Book-Crossing dataset, the overall amount of ratings obtained is much smaller than the amount of rating data obtained from the processed MovieLens-20m dataset, which makes that the obtained user similarity matrix and item similarity matrix are still very sparse. Therefore, the prediction effect of the **User-based CF** algorithm and the **Item-based CF** algorithm is not much different for the **Book-Crossing** dataset.

Model	**User-based CF**	
Dataset	test MAE	test RMSE
ml-20m	**2.35**	**2.59**
book-crossing	3.88	3.98
Model	**Item-based CF**	
Dataset	test MAE	test RMSE
ml-20m	2.75	2.94
book-crossing	**3.86**	**3.96**

Table 3. Test results for User-based CF, Item-based CF

3.3.2 SVD, NMF, KNNWithMeans

After GridSearchCV, best parameters and test results for **SVD**, **NMF**, **KNNWithMeans** of best parameters are listed in Tab. 4.

As the results suggest, for the **MovieLens-20M** dataset, the **SVD** model performs the best with an RMSE of 0.56 and an MAE of 0.73, using the parameters: 50 latent factors (n_factors), 30 training epochs (n_epochs), a learning rate of 0.005 (lr_all), and a regularization factor of 0.1 (reg_all). Similarly, on the **Book-Crossing** dataset, the **SVD** model also delivers relatively good results, with an RMSE of 0.58 and an MAE of 0.76.

The **NMF** model also shows a realtively good performance, particularly on the **MovieLens-20M** dataset, achieving an RMSE of 0.60 and an MAE of 0.78 with 25 latent factors and 100 epochs. However, its performance declines on the **Book-Crossing** dataset, where the RMSE increases to 0.69 and the MAE to 0.91.

The tuned **KNNWithMeans** model, using a Pearson similarity metric and user-based collaborative filtering, performed the best on both datasets. For **MovieLens-20M** dataset, it achieves an RMSE of 0.61 and an MAE of 0.80, while for **Book-Crossing** dataset, it returns an RMSE of 0.63 and an MAE of 0.84.

In conclusion, the best model to predict recommendation for **MovieLens-20M** data set is the **SVD** model with parameters of $n\_factors = 50, n\_epochs = 20, lr\_all =$

$0.005, reg\_all = 0.02$, reaching a lowest RMSE (0.56) and MAE (0.73). For **Book-Crossing** dataset, best model is also **SVD**, whose best parameters change to $n\_factors = 50, n\_epochs = 30, lr\_all = 0.005, reg\_all = 0.1$, reaching RMSE(0.58) and MAE (0.76). Results suggest that all models are more effective at modeling user-item interactions based on denser datasets. And among these 3 algorithms, **SVD** performs the best while also taking the most time to train. The performance of **NMF** is close to **SVD** on **MovieLens-20M** but deteriorates significantly on the **Book-Crossing** dataset, highlighting the limitations of **NMF** in handling realatively sparse data. Finally, **KNNWithMeans** offers a simpler and faster approach to recommendation as it provides outcomes the most quickly in these 3 models, although it does not match the precision of **SVD** across both datasets.

Model	**SVD**	
Dataset	test MAE	test RMSE
ml-20m	**0.73**	**0.56**
book-crossing	**0.76**	**0.58**
Model	**NFM**	
Dataset	test MAE	test RMSE
ml-20m	0.78	0.60
book-crossing	0.91	0.69
Model	**KNNWithMeans**	
Dataset	test MAE	test RMSE
ml-20m	0.80	0.61
book-crossing	0.84	0.63

Table 4. Test results for SVD, NFM, KNNWithMeans

3.3.3 DSSM

The experimental results demonstrate that the DSSM model achieved relatively consistent performance across both the ml-20m and book-crossing datasets. Specifically: On the ml-20m dataset, the model achieved a MAE of 0.68 and RMSE of 0.86, while on the book-crossing dataset, it obtained a MAE of 0.71 and RMSE of 0.82. Both sets of metrics indicate good predictive accuracy of the model.

Comparing the metrics, the model shows slightly better MAE performance on the ml-20m dataset compared to book-crossing (0.68 vs 0.71), indicating smaller average prediction errors. Interestingly, the RMSE on ml-20m is higher than book-crossing (0.86 vs 0.82), suggesting that there might be some samples with larger prediction deviations in the ml-20m dataset, leading to a higher root mean square error.

Overall, these test results indicate that the DSSM model demonstrates robust performance across two different datasets, with prediction errors maintained within an acceptable range.

Model	**DSSM**	
Dataset	test MAE	test RMSE
ml-20m	**0.68**	**0.86**
book-crossing	**0.71**	**0.82**

Table 5. Test results for DSSM

4. Conclusion and Discussion

4.1. Conclusion

This study conducted a comprehensive evaluation of six recommendation algorithms across two datasets—MovieLens-20M and Book-Crossing. The performance of each algorithm was measured using Mean Absolute Error (MAE) and Root Mean Square Error (RMSE), and the results are visualized 2.

Our findings indicate that model-based approaches, particularly SVD, generally outperform memory-based techniques in terms of prediction accuracy, especially in dense datasets like MovieLens-20M. Conversely, memory-based methods such as UserCF tend to perform less accurately in sparse datasets, as demonstrated by the Book-Crossing data. DSSM showed potential for handling complex user-item interactions but requires additional computational resources and detailed feature engineering to achieve optimal results. These findings align with our initial expectations: memory-based collaborative filtering methods are highly susceptible to data sparsity and exhibit poorer performance on sparse datasets. In contrast, model-based methods, particularly those utilizing matrix factorization and latent vector representations, demonstrate enhanced accuracy and robustness against sparsity, albeit at the cost of increased computational complexity.

4.2. Drawbacks and Prospects

4.2.1 Drawbacks

Despite the valuable insights gained, several limitations in our study warrant discussion:

- **Dataset Limitations:** The study focused on explicit rating datasets, which provide clear insights into user preferences. However, in real-world business scenarios, explicit ratings are seldom available. Users

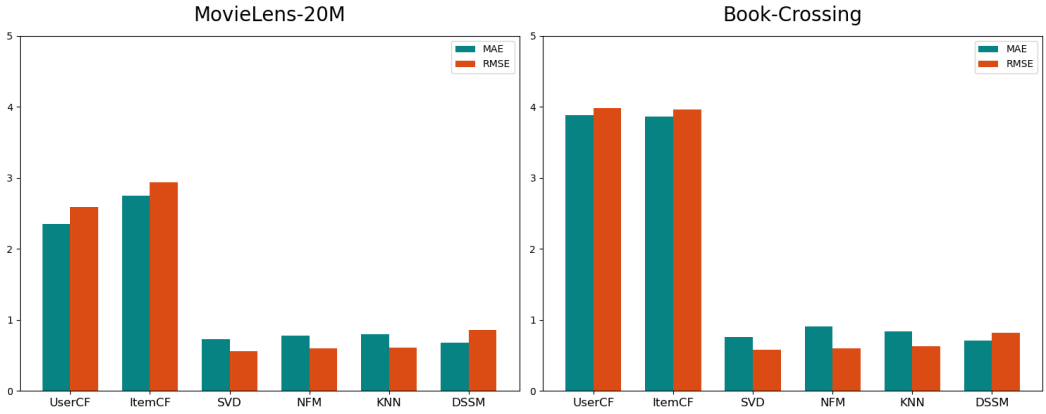

Figure 2. Test results on 2 datasets

more commonly express their preferences through implicit feedback mechanisms such as clicks, views, purchases, and other interaction behaviors. Reliance on explicit ratings limits the applicability of our findings to practical systems where implicit feedback is predominant.

- **Evaluation Metrics:** The use of MAE and RMSE provides a quantitative assessment of prediction accuracy but does not fully capture the effectiveness of recommender systems in achieving business objectives like increasing click-through rates or sales. These offline metrics may not correlate directly with user satisfaction or business performance indicators.

- **Sampling and Process Limitations:** Due to computational constraints, experiments were conducted on subsets of the datasets. This approach does not reflect the scale at which commercial recommender systems operate, where millions of users and items are involved. Additionally, the study did not implement the multi-stage recommendation process typically used in industry, which includes stages such as candidate generation (recall), coarse ranking, fine ranking, and re-ranking. Each stage serves distinct purposes and is crucial for handling large-scale data efficiently.

4.2.2 Future Prospects

Building upon the insights and acknowledging the limitations, several avenues for future enhancements emerge:

- **Generalization to Implicit Feedback Data:** Future work should aim to extend these algorithms to handle implicit feedback data, which is more representative of real-world user interactions. This involves developing models capable of interpreting and leveraging various user behaviors to infer preferences, thereby enhancing the relevance and personalization of recommendations.

- **Adoption of more Business-Relevant Evaluation Metrics:** Incorporating evaluation metrics that align with business goals—such as precision, recall, click-through rate (CTR), diversity, and novelty—would provide more actionable insights into the recommender system's performance, which can better reflect user engagement and satisfaction. Also, in real-world implementation, effectiveness of a recommendation strategy is needed to be tested in practical experiments, which may requires companies to conduct A/B Testing on their customers and analysis the experiment results.

- **Scalable Recommendation Frameworks:** Designing recommendation processes that are scalable to large datasets is essential. Implementing multi-stage pipelines with dedicated recall, ranking, and re-ranking phases can improve efficiency and effectiveness. Fast recall mechanisms can quickly narrow down the vast item pool, while advanced ranking models can provide more accurate predictions of user preferences.

- **Advanced Modeling Techniques:** Exploring advanced machine learning and deep learning techniques, such as graph neural networks, attention mechanisms, and reinforcement learning, could further enhance the capability of recommender systems. Hybrid models that combine collaborative filtering with content-based approaches may also mitigate the limitations of individual methods.

By acknowledging these drawbacks and focusing on future improvements, we can develop more robust and effective recommender systems. This work lays a foundational understanding for future research and practical applications,

contributing valuable insights to the ongoing development of advanced recommender systems.

References

[1] Charu C. Aggarwal. *Recommender Systems: The Textbook*. Springer International Publishing, 2016. 6

[2] Robert M. Bell and Yehuda Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, 2007. 4

[3] Avrim Blum, John Hopcroft, and Ravindran Kannan. *Foundations of Data Science*. Cambridge University Press, 2020. 3

[4] Jonathan L Herlocker, Joseph A Konstan, Al Borchers, and John Riedl. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 230–237, 1999. 2

[5] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 2333–2338, 2013. 2

[6] Yehuda Koren. Factor in the neighbors: Scalable and accurate collaborative filtering. *ACM Trans. Knowl. Discov. Data*, 2010. 2, 4

[7] Daniel D. Lee and H. Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 1999. 3

[8] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295, 2001. 2

[9] Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*, 2009. 4

[10] Johan A. K. Suykens, Marco Signoretto, and Andreas Argyriou. The why and how of nonnegative matrix factorization. In *Regularization, Optimization, Kernels, and Support Vector Machines*, page 525, 2014. 3, 4

[11] Lyle H Ungar and Dean P Foster. Clustering methods for collaborative filtering. In *AAAI workshop on recommendation systems*, volume 1, pages 114–129. Menlo Park, CA, 1998. 2