Technical University Berlin

A Hybrid Approach to Recommender Systems based on Matrix Factorization

Diploma Thesis

at Department for Agent Technologies and Telecommunications
Prof. Dr.-Ing. habil. Sahin Albayrak
Faculty IV - Electrical Engineering and Computer Science
Technical University Berlin

presented by

Stephan Spiegel

Supervisor:	Prof. Dr.-Ing. habil. Sahin Albayrak,
	Dr.-Ing. Stefan Fricke
Advisor:	Dipl.-Inform. Jérôme Kunegis

Stephan Spiegel
Student ID: 301132
Düsseldorfer Straße 67
10719 Berlin

Declaration of Authorship

I certify that the work presented here is, to the best of my knowledge and belief, original and the result of my own investigations, except as acknowledged, and has not been submitted, either in part or whole, for a degree at this or any other University.

Date:

Name:

Signature:

Abstract

Due to the huge amount of information available online, the need of personalization and filtering systems is growing permanently. Recommendation systems constitute a specific type of information filtering technique that attempt to present items according to the interest expressed by a user. Commonly online recommender are employed for e-commerce applications or customer adapted websites.
In general, there exist two basic types of recommendation techniques, namely content-based filtering and collaborative filtering. Whereas content-based filtering methods examine items previously favored by the actual user, collaborative filtering computes recommendations based on the information about similar items or users. In our work we combine both techniques into a hybrid approach, where supplementary content features are employed to improve the accuracy of collaborative filtering.
For the development of our hybrid recommender we utilized the well-known *MovieLens* rating data as well as the *IMDB* online movie archive. The content information retrieved from *IMDB* is converted into a notation that is useable for our hybrid approach. Rating and content data are both normalized separately, before the combined information is utilized by our recommendation algorithm. In order to reduce the computational effort of our hybrid model, we furthermore factorize the extended rating matrix by means of singular value decomposition.
A prototype system of our novel hybrid recommender was implemented in *MATLAB* programming language. By means of various experiments, we could demonstrate that the extracted content features are beneficial to our underlying rating prediction algorithm. In addition, we discover a way to reveal latent feature relations, which can be used to generate more individual and accurate recommendations.

Keywords:
Hybrid Recommendation System, Collaborative Filtering, Content-Based Filtering, Feature Retrieval, Matrix Factorization, Rating Normalization, Latent Feature Relations

Zusammenfassung

Aufgrund der gewaltigen Informationsflut im Internet, werden immer mehr Personalisierungs- und Filtersysteme benötigt. Empfehlungssysteme stellen eine spezielle Form von Informationsfilter dar, welche den dargestellten Inhalt an die Präferenzen des Nutzers anpassen. Gewöhnlich werden Empfehlungssystem für Internetversandhäuser oder anwenderbezogene Webseiten benutzt.
Im Allgemeinen existieren zwei Grundformen von Empfehlungssystemen, namentlich Inhaltsbasiertes und Kollaboratives Filtern. Während Inhaltsbasiertes Filtern Artikel untersucht die vom aktuellen Nutzer zuvor betrachtet wurden, berechnet Kollaboratives Filtern die Empfehlungen beruhend auf ähnliche Artikel oder Nutzer. In unserer Arbeit fügen wir beide Techniken zusammen, wobei zusätzliche Artikelbeschreibungen die Genauigkeit der Empfehlungen verbessern sollen.
Für die Entwicklung unseres gemischten Ansatzes haben wir *MovieLens* Bewertungen sowie *IMDB* Beschreibungen verwendet. Artikelbeschreibungen die von dem *IMDB* Archiv bezogen wurden, mussten zuerst in ein für unseren Ansatz passendes Format gebracht werden. Bewertung und Beschreibungen werde beide getrennt normalisiert, bevor die kombinierten Daten durch unser Empfehlungssystem ausgewertet werden können. Um die Rechenzeit für unseren gemischten Ansatz zu minimieren, wird die erweiterte Bewertungsmatrix anhand der Singulärwerte zerlegt.
Ein Prototyp unseres neuartigen Ansatzes wurde in der Programmiersprache *MATLAB* umgesetzt. Durch verschiedene Experimente konnten wir nachweisen, dass die entnommenen Beschreibungsmerkmale einen positiven Einfluss auf die Voraussage von fehlenden Bewertungen haben. Des Weiteren haben wir einen Weg gefunden um verborgene Merkmalsbeziehungen aufzudecken, welche für die Erstellung von individuelleren und zuverlässigeren Empfehlungen eingesetzt werden können.

Acknowledgement

First of all I would like to thank my supervisor Jérôme Kunegis for supporting me in all questions that occurred during the realization of my thesis. Furthermore, I appreciate the useful advices provided by colleagues of the *DAI* laboratory.
In addition, I want to express gratitude to all my fellow students and friends who always lend a sympathetic ear. I am grateful for their moral support, which was an essential motivation for keep going with my research.

Contents

Chapter 1

Introduction

1.1 Motivation

During the last decade the amount of information available online increased exponentially. Recommender system address the problem of filtering information that is likely of interest to individual users. Typically, users profiles are employed to predict ratings for items that have not been considered [1]. Depending on the application domain, items can be web pages, movies or any other products found on a web store. Probably one of the most famous online recommender systems is *Amazon*[1], which suggests books and other articles to their customers.

Although many different approaches to recommender systems have been developed within the past few years, the interest in this area still remains high. This is due to growing demand on practical applications, which are able to provide personalized recommendation and deal with information overload [1, 2]. Another accelerator for the research in the field of recommender systems is the *Netflix Prize* competition[2] started in October 2006. The competition is held by *Netflix*, an online DVD rental service, and seeks to improve the accuracy of predictions about how much someone is going to like a movie based on their preferences. Many scientist, students, engineers and enthusiasts were attracted by the free accessible large-scale dataset and the public announced $1,000,000 prize money for the winner of the competition.

Recommender systems are usually classified according to their approach to rating estimation. In general we can distinguish between content-based and collaborative recommender systems. Content-based recommendations [3] are typically based on item similarity to objects the user preferred in the past. In contrast, collaborative recommen-

[1] http://www.amazon.com

[2] http://www.netflixprize.com/

dation systems [4] depend on the ratings given by individuals with similar taste and preference. However, both techniques exhibit specific drawbacks.

Some of the central problems concerning content-based recommender systems are limited content analysis, overspecialization and the *new user problem* [2]. Content-based techniques mostly analyze item features that were automatically extracted by information retrieval methods. While there exist sophisticated algorithms to parse textual documents, feature extraction might be much harder to apply to multimedia data or items with heterogeneous features. Furthermore, recommendations based on content-based techniques tend to overspecialize, because only items with a high similarity to those already rated will be suggested to the individual user. Another problem with content-based recommenders is that a user first has to rate a sufficient number of items before the system is able to make accurate recommendations.

Unlike content-based recommendation methods, collaborative recommender systems make predictions based on items previously rated by other users. Nonetheless, collaborative recommender systems exhibit the *new user problem* and first have to learn user preferences to make reliable recommendations. Beside the new user *cold start problem* collaborative approaches also exhibit the *new item problem*, which means that a new item needs to be rated by a sufficient number of users in order be recommended accurately by the system. Moreover, the success of a collaborative recommender system depends on the amount of rating information available. Typically the number of ratings obtained is very small compared to the number of ratings that need to be predicted. In other words, the user-item matrix is usually very sparse, which consequently leads to poor recommendations [2].

Beside recommendation precision, computation efficiency is a key consideration in all fields of computer science [5]. Usually a recommender needs to deal with millions of users and items, computing rating estimations in an instant or even in real time. Under the restrictions of memory and time consumption many prediction algorithms quickly reach their limit of possible manageable data volume. In order to handle large-scale datasets, further improvements on information representation and recommendation modeling need to be done.

Without doubt, it is impossible to tackle all mentioned problems at once. Recommender systems constitute an own independent research field, containing dozens of sub-domains and interdisciplinary aspects that are worth to be studied. In the next section we localize the exact problem that we want to investigate within this thesis.

1.2 Purpose

The main aim of this research work is to find a general way to make recommendation methods more effective in a broader range of applications. Although our experiments merely focus on one specific dataset, we desire to develop a universal model that can be applied to any other problem domain.

For our research we want to investigate the well-known *MovieLens* dataset [6][3], which contains 100,000 anonymous ratings (1-5) of approximately 1682 movies made by 943 users who joined *MovieLens* in 2000. Unknown ratings can be estimated in many different ways using methods from machine learning, approximation theory, and various heuristics [2].

Typically collaborative recommendation techniques, like the popular *Nearest Neighborhood* approach [4], are employed to compute personalized recommendations for particular users. In order to improve the precision of recommendations, we plan to develop a hybrid system that not only utilize obtained user ratings but also incorporate contextual information of corresponding items. Our novel approach will be compared to established recommendation methods to analyze the influence of different supporting content features on the final rating estimation.

Due to the fact that the *MovieLens* dataset does not contain any content features, we need to extract these supplementary information from an external source. The *Internet Movie Database (IMDB)* [7] is by far the most extensive collection of movie information available nowadays. Movie features can be retrieved from the *IMDB* website[4] or accessed directly from a copy of the data on the local system. Part of our work will be to find suitable relations between the entries of both databases and to interconnect appropriate item features.

Another major objective of our research is to scrutinize the effect of dimension reduction on recommendation precision. One possible technique to lower the dimension of the obtained user-item feature information is matrix factorization. There exist several examples [8, 9] that matrix factorization can lead to higher recommendation accuracy and further decrease computational runtime.

Having defined a set of problems we want to investigate, the next section will discuss the further proceeding of our work.

[3] http://movielens.umn.edu/login

[4] http://www.imdb.com/

1.3 Approach

To overcome the limitations of content-based and collaborative recommender system, we decided to follow a hybrid approach [10, 11] that combines both methods. As stated earlier, our goal is to construct a unified model that incorporates user ratings from the *MovieLens* dataset as well as movie features from the *IMDB* corpus.
Two-part hybrid recommender systems are quite successful, but under certain domain and data characteristics different hybrids might achieve unlike results. For our intended movie recommendation system we decided to use the feature combination strategy [10], making use of a contributing recommender that provides supplementary features for the actual recommender. To be more specific, content features are utilized to strengthen collaborative recommendation.
Instead of employing the retrieved content features to interpolate missing ratings [11], new features are attached to the sparse user-item matrix right away. The general idea behind this approach is that the expanded matrix gives more precise information about item-item similarities. Due to the fact that we are going to employ the *item-based Nearest Neighborhood algorithm* [4] for our examination, these item similarities will have a significant influence on the final recommendation results.
Possible content features for our movie items are genre, release date, place of production and many more [7]. Since we do not know which content features have a positive or negative effect on our rating prediction, we plan to assign discrete weights to each single feature. Whereas, high feature weights imply a high importance and vice versa. Stepwise experiments will reveal the optimal weight and parameter configuration for our hybrid recommendation system.
Extending the original user-item rating data with additional content features yields an inflated mixed feature matrix. Depending on the number of added content features the updated matrix can become quite large. As pointed out before, we want to apply matrix factorization techniques to handle these large-scale matrices, which are typical for collaborative recommender systems. Factorization methods like the *Singular Value Decomposition (SVD)* are able to compute low-rank approximations, which can be used to efficiently calculate item similarities [8]. In turn this speeds up the determination of the closest item peers, which pose a computational expansive step within the *Nearest Neighborhood* algorithm.
In order to evaluate our proposed system we need to split the original rating information into a training and test set. By means of these two disjoined datasets we are able to measure the difference between predicted and actual values. Whereas the obtained

training set is utilized to predict all ratings contained in the test set. Fortunately, the *GroupLens Research Project*[5] already provides several different training and test samples, which exhibit perfect properties for validation.

One standard measure for the accuracy of recommender systems is the *Root Mean Squared Error (RMSE)* [4, 8], which exactly expresses how precise a constructed heuristic model estimated missing ratings. By means of the *RMSE* measure we are able to compare our own approach with other acknowledged recommender systems. Of course an appropriate comparison of different systems is only possible if one and the same dataset is employed for all conducted experiments.

Another crucial point for the success of our novel hybrid recommender system is an elaborated normalization method. This is due to the fact that rating information usually exhibit large user and item effects [8]. Some user tend to give higher ratings than others, and likewise some items receive higher ratings than others. Our research seeks to automatically learn normalization parameters using *Linear Regression* techniques. However, original rating matrix and attached content information have to be processed separately, because their items lie in different range.

[5] http://www.grouplens.org/

Chapter 2

State of the Art

2.1 Collaborative Filtering

Collaborative Filtering (CF) is the process of evaluating information using the opinion of other people [4]. Typically, predictions about user interests are made by collecting taste information from many other similar users. Thereby it is assumed that those individuals agreed in the past tend to agree again in the future[1]. Often CF systems need to process huge amounts of information, including large-scale datasets such as in electronic commerce and web applications. Within the last decade CF has been improved continually and finally became one of the most prominent personalization techniques in the field of recommendation systems.

Originally CF was inspired by the nature of human being, and the fact that people share opinions with other people. For example, when several of your good friends like the same book, you might decide that you also should read it. Similarly, if many of them found it boring, you might decide to spend your money elsewhere [4]. This is due to the fact that taste is a sociological concept and not only a personal subject.

Nowadays computers and the internet allow us to consider the opinions of large interconnected communities with thousands of members [4]. Individuals can profit from a community, in that they get access to the knowledge of other users and their experience about diverse items. Furthermore, this information can help individuals to develop their own personalized view or to make a decision regarding the items that where rated. To be more specific, people use CF systems for finding new items they might like, getting recommendations on particular items and connecting to other users with same interests.

[1] http://en.wikipedia.org/wiki/Collaborative_Filtering

The most popular non-probabilistic CF approach is the *Nearest Neighborhood (NNH)* algorithm. In general there exist two basic NNH techniques, namely user-based nearest neighborhood and item-based nearest neighborhood [4].
User-based methods first look who shared the same rating pattern with the active user, and then use the ratings of the like-minded users to calculate the predictions. To estimate the rating for a yet unconsidered item of the active user, the ratings of the nearest neighbors about this specific item are averaged. In order to generate more accurate predictions, rating values of neighbor are assigned with weights according their similarity to the target user. Generally, user similarity is calculated by means of the *Pearson Correlation Coefficient*, which compares ratings for items rated by both target user and neighbor.
Item-based nearest neighbor algorithms are the transpose of user-based algorithms, and generate predictions based on similarities between items [4]. For a user-item pair *(u,i)* the prediction is composed of a weighted sum of the user *u's* ratings for the item most similar to *i*. The general item-based prediction algorithm has been formalized in the following equation.

$$pred(u,i) = \frac{\sum_{j \in ratedItems(u)} itemSim(i,j) \cdot r_{ui}}{\sum_{j \in ratedItems(u)} |itemSim(i,j)|}$$

There exist several variations for calculating the similarity between two items. The *Person Correlation Coefficient* alias *Adjusted-Cosine Similarity* [4, 12] is considered the most accurate similarity metric among all.

$$itemSim(i,j) = \frac{\sum_{u \subset ratedBoth(i,j)} (r_{ui} - \overline{r}_i)(r_{uj} - \overline{r}_j)}{\sqrt{\sum_{u \subset ratedBoth(i,j)} (r_{ui} - \overline{r}_i)^2} \sqrt{\sum_{u \subset ratedBoth(i,j)} (r_{uj} - \overline{r}_j)^2}}$$

The major drawback of this approach is the high memory usage for the generated item similarity matrix *($|items|^2$)*. However, the size of the model can be substantially reduced by just considering the correlations with more than a minimum number of co-ratings, or even further by only retaining the top *n* correlations for each item [4].

Although these measures increase memory and computing time, less correlations leads to more difficult prediction. Nonetheless, item pair with few co-ratings are easily to dominate prediction, and negatively affect prediction.

Without doubt, one of most popular applications in the field of collaborative filtering is *Amazon's* recommendation system[2]. *Amazon* uses recommendations for marketing campaigns and personal adaptation of its homepage. Customers have the possibility to receive individual suggestion on *Amazon* products, based on the articles the purchased previously. For the reason that ordinary CF algorithms cannot scale *Amazon's* massive datasets *(∼30M users)*, they developed their own *Item-To-Item Collaborative Filtering method* [12]. The main difference to traditional item-based CF techniques is that the *Amazon* algorithm prunes items which have no common customers or belong to unlike product catalogs. However, in order to provide realtime recommendations, the expensive item-similarity matrix is computed offline.

Figure 2.1 presents the interaction of a user with an online collaborative recommender system through a web interface. In order to suggest products to a user, web server and recommender need to communicate with each other. Usually, the web server application forwards user feedback to the recommender, and receives personalized recommendations in return. User ratings and item correlations are both stored on the recommender platform to ensure realtime results.

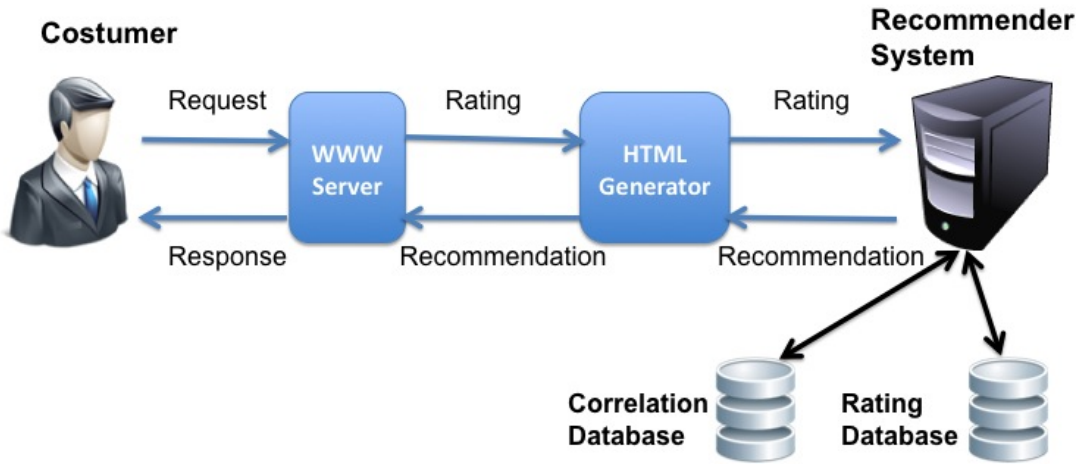


Figure 2.1: Collaborative Recommender System Architecture [13]

[2]Introduction to Amazon's Recommendation System
http://www.amazon.com/gp/help/customer/display.html?nodeId=13316081

2.2 Content-Based Recommendation

Content-Based recommender systems make suggestions upon item features and user interest profiles [3]. Typically, personalized profiles are created automatically through user feedback, and describe the type of items a person likes. In order to determine what items to recommend, collected user information is compared against content features of the items to examine.

Usually, items are stored in database tables, whereas each row describes one item and each column represents a specific item feature. Depending on the domain, features can be represented either by boolean values or by a set of restricted values *(i.e.* $x \in \mathbb{N}$*)*. For example, imagine we want to analyze a set of newspaper articles about different kind of topics. While a boolean value could indicate whether a word is contained in an article or not, an integer value could express the number of times a word appears.

Instead of straightforward word occurrences, sophisticated text analyzing techniques associate numeric weights to each word, which express the relevance of the single terms. In general, term weights are a combination of *Term Frequency (TF)* and *Inverse Document Frequency (IDF)*, considering relative document length as well as composition of the document collection *(TF*IDF approach)* [14].

As mentioned earlier, most content-based recommender systems store user interests in personalized profiles. Typically, user profiles contain descriptions about the items an individual likes most or information about user interactions with the recommendation system. Interactions may include saving user queries as well as capturing user activities related to particular items [1]. The history of user interactions can facilitate the user returning to viewed items, but also can assist the recommender system to filter out item that were already considered by the user. In addition, user history may serve as training data for machine learning algorithms, which employ the gathered information to generate a more sophisticated user model [3].

User information might be provided explicitly by the individual person or gathered implicitly by a software agent. Explicit user information collection basically relies on personal input by the user [1]. A common feedback technique is the one that allows users to express their opinions by selecting a value of range. However, filling out forms or clicking checkboxes places a burden on the user. Profiles might be imprecise, because the user is not willing to spend a lot of time providing personal information or the user information is already outdated. In contrast, implicit feedback does not require any additional intervention by the user during the process of constructing profiles. Moreover, it automatically updates as the user interacts with the system [1]. Thus, systems that

collect implicit feedback are more likely to be used in practice.
To sum up, *Content-based Filtering (CBF)* uses the assumption that items with similar objective features will be rated similarly [3]. The challenge is to extract those item features that are most predictive. Once a user profile has been constructed form the items a user expressed interest in, then new item can be compared to the profile to make recommendations. In contrast, *Collaborative Filtering (CF)* assumes that people with similar tastes will rate things similarly [4]. Where CF needs ratings for an item in order to predict for it, content-based filtering can predict relevance for items without rating. On the other hand, for many domains it is quite difficult to obtain content information (e.g movies and music). As a matter of principle, content-based recommender are limited to the information they can access. Another main drawback of content-based filtering approaches is that they tend to overspecialize, because solely items that match the content features in the user profile are recommended.
One example for an early CBF system is *WebMate* [15], an agent that assist users to effectively browse and search the Web. *WebMate* employs multiple profile vectors to keep track of user interests in different domains. In order to learn from users *WebMate* deploy use of a *http proxy* that monitors all user browsing and search activities. Visited web pages need to be pre-processed before they can be utilized to build the user model. Pre-processing includes deleting of *stop-words*, term *stemming* and *TF-IDF* keyword weighting [14]. Finally, *WebMate* uses the originated profiles to compile a personalized newspaper, by comparing the weighted keyword vectors with previously unseen web documents. To be sure, new documents are pre-processed as well.

2.3 Hybrid Recommender Systems

The last two sections discussed the strengths and weaknesses of *Collaborative Filtering (CF)* and *Content-based Filtering (CBF)*. In order to achieve better recommendation results, researchers combined both techniques to build *Hybrid Recommender* systems, which seek to inherit vantages and eliminate disadvantages [10, 16].
In general, hybrid recommenders are systems that combine multiple recommendation techniques together to achieve a synergy between them. Although there exist a number of recommendation approaches that are practical to merge *(i.e. Collaborative, Content-based, Demographic and Knowledge-based Recommender)*, our work will mainly focus on the combination of CF and CBF techniques.
Depending on the domain and data characteristics, different types of combinations might produce dissimilar outputs. The following list describes several hybridization

techniques that come into consideration to merge CF and CBF recommenders [10].

Weighted: Perhaps the most straightforward architecture for a hybrid system is a weighted one. Given items are scored separately by both incorporated recommender, whereas the final output is a linear combination of the intermediate results. Typically, empiric means are used to determine the best weights for each component. Note that content-based recommenders are able to make prediction on any item, but collaborative recommender can only score an item if there are peer users who have rated it.

Mixed: In many domains it is infeasible to receive an item score by both recommenders, because either rating matrix or content space are too sparse. Mixed hybridization techniques generate an independent set of recommendations for each component, and join the ranked candidates before being shown to the user. However, merging the predicted items of both recommenders makes it difficult to evaluate the improvement about the individual components.

Switching: Some hybrid systems consist of more than two recommendation components with different underlying CF and/or CBF approaches. Often recommenders are ordered, and if the first one cannot produce a recommendation with high confidence, then the next one is tried, and so on. On the contrary, other switching hybrids might select single recommenders according to the type of user of item. However, this method assumes that some reliable switching criterion is available.

Feature Combination: Systems that follow the feature combination approach only employ one recommendation component, which is supported by a second passive component. Instead of processing the features of the contributing component separately, they are injected into the algorithm of the actual recommender.

Feature Augmentation: The strategy of feature augmentation is similar in some ways to feature combination. But instead of using raw features from the contributing domain, feature augmentation hybrid's support their actual recommender with features passed through the contributing recommender. Usually, feature augmentation recommender are employed when there is a well-engineered primary component that require additional knowledge sources. Due to the fact that most applications expect recommendations in realtime, augmentation is usually done offline. In general, feature augmentation hybrids are superior to feature combina-

tion methods, because they add a smaller number of features to the primary recommender. Melville, Mooney and Nagarajan [17] developed a feature argumentation recommender that incorporates content-based and collaborative methods to predict new items of interest for a user. Their *Content-Boosted Collaborative Filtering (CBCF)* algorithm learns a content-based model over the training data to generate ratings for unrated items. The generated dense rating matrix is then used for collaborative recommendation by the actual recommender. CBCF overcomes disadvantages of both CF and CBF methods, and significantly improved the predictions of the recommender system.

Cascade: The concept of a cascade hybrids is akin to feature augmentation techniques. However, cascade models make candidate selection exclusively with the primary recommender, and employ the secondary recommender simply to refine item scores. For example, items that were equally scored by the main component might be re-ranked employing the secondary component.

Meta-Level: This kind of hybrids employ a model learned by the contributing recommender as input for the actual one. Although the general schematic of meta-level hybrids reminds on feature augmentation techniques, there exists a significant difference between both approaches. Instead of supplying the actual recommender with additional features, a meta-level contributing recommender provides a completely new recommendation space. However, it is not always necessarily feasible to produce a model that fits the recommendation logic of the primary component.

There exist a few surveys that compare different hybridization techniques using the same data. *Burke* [10] compared the full scope of hybrid design utilizing the user profile data of the well-known *Entree* recommendation system[3]. Analyzed recommendation methods include content-based filtering, knowledge-based retrieval as well as two different collaborative algorithms. To measure the performance of examined hybrid combinations the *average rank of the correct recommendations (ARC)* is computed. Between all discussed hybridization techniques, the feature augmentation hybrids showed the best performance. Especially, if the content-based recommender contributed to the collaborative one [10].

[3]Data contains a record of user interactions with the Entree restaurant recommendation system http://kdd.ics.uci.edu/databases/entree/entree.html

2.4 Matrix Factorization

Recommender systems often need to deal with large-size user and product data, and are furthermore required to produce high quality recommendations with in a split second. Collaborative filtering is the most successful recommendation technology in the field of E-commerce. Most collaborative recommender employ the nearest neighborhood *(NNH)* algorithm [13] in combination with *Pearson correlation* (or *cosine similarity*) to compute rating predictions.

However, due to sparse rating information, *NNH* approaches usually experience difficulties in finding exact matches and consequently produce recommendations with poor accuracy [13]. Furthermore the computational complexity of *NNH* algorithms tends to grow with both the number of user and the number of products, whereby recommender system suffer serious scalability problems.

To overcome these weaknesses the KDD[4] research community explored alternative recommendation algorithms. In many recent approaches *Latent Feature Indexing (LSI)* is applied to reduce the dimensionality of the user-item rating matrix [18]. LSI has been widely used in *Information Retrieval (IR)* to extend document modeling techniques and to reveal semantic relationships between terms, assuming there is a hidden structure in using the terms in a collection of documents [14]. In contrast, collaborative recommenders try to capture relationships among user-user or item-item pairs. By reducing the dimensionality of the user or rather item space, we can increase density and thereby find more ratings.

Usually, LSI employs *Singular Value Decomposition (SVD)* as its underlying matrix factorization algorithm [13]. SVD is an established matrix factorization techniques with several applications in signal processing and statistics. Suppose *R* is an $m \times n$ matrix with rank *r* whose entries are either real number or complex numbers. Then there exists a *Singular Value Decomposition* of the form:

$$R = U \cdot S \cdot V'$$

where *U* and *V* are orthogonal matrices of size $m \times r$ and $n \times r$ respectively; and S is a diagonal matrix of size $r \times r$ with all singular values of *R* as its diagonal entries [13]. This lower rank approximation of the original rating matrix *R* is able

[4]Knowledge Discover in Databases - KDD.org

to capture latent relationships between users and items, which allows a collaborative recommendation systems to predict user-item ratings straightaway. Furthermore, this low-dimensional representation of the user-item space reduces the computational complexity of neighborhood algorithms. In fact, it is possible to reduce the original user-item space to a designated dimension *k*, in which matrix S only contains the *k* largest singular values $(k < r)$. Accordingly, the closest *rank-k* approximation to *R* is formulated as:

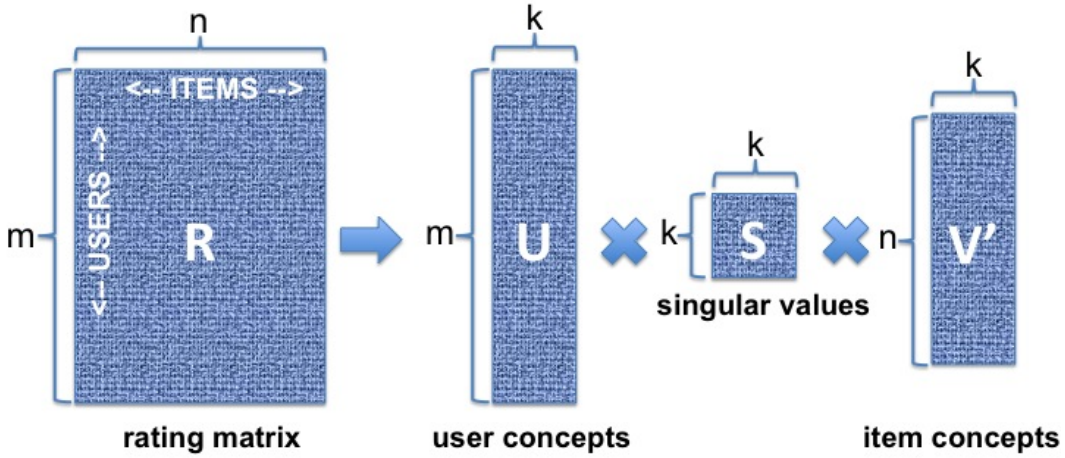


Figure 2.2: Factorization of Rating Matrix

Depending on the value *k* collaborative filtering will produce quite different prediction or rather recommendation results. Generally, *k* should be large enough to capture all latent relationships and small enough to avoid over-fitting.
In practice, recommender systems need to handle large amounts of user and item data, and are required to provide real-time recommendations. For that reason, decomposition of the user-item matrix and computation of similarity values between user-user or rather item-item pair is usually done offline. Matrix factorization using an ordinary SVD algorithm requires a time in order of $O(m+n)^3$ [13].
Several studies on collaborative recommender systems show that the singular value decomposition has the potential to improve improve prediction accuracy [19, 8, 20]. Especially in case of data sparsity, SVD-based algorithms often outperform ordinary nearest-neighbor collaborative filtering techniques. Although, computing the SVD is expensive, dimension reduction techniques persuade in terms of memory consumption and online performance.

2.5 Data Normalization

Typically rating information exhibit large user and item effects, which need to be adjusted before processing to avoid over-fitting the observed data [8]. For example, some users tend to give higher ratings than other, and some items generally receive higher ratings than others. A possible way to clean collaborative filtering data ($R_{m\times k}$) is to shrink the user-item ratings towards the baseline default:

$$\begin{aligned}\tilde{r}_{ui} &= r_{ui} - \alpha\overline{r} - \beta\overline{r}_u - \gamma\overline{r}_i \\ &= r_{ui} - \alpha(\frac{1}{m\cdot k}\sum_{u,i} r_{u,i}) - \beta(\frac{1}{m}\sum_{u} r_{u,i}) - \gamma(\frac{1}{k}\sum_{i} r_{u,i})\end{aligned}$$

where a combination of global effects is subtracted from the original rating. In which order those effects are removed does not influence the final result [21]. The optimal parameter setting can be determined by machine learning techniques or statistical methods (i.e. regression analysis).

Beside user and item effects, there also exist other factors that are likely to influence rating data [22]. For example, the rating received for an item usually depends on the date when the feedback was given. Over time, items go out of fashion and users may change their taste. In general, global effects cause much of the data variability and mask the fundamental relationship between ratings. Several studies [8, 23, 22] showed that after removing global effects, correlations among items become smaller and hence estimation accuracy is improved.

Note that it is necessary to add the previously subtracted effects back on the predicted ratings:

$$\widehat{pred}(u,i) = pred(u,i) + \alpha\overline{r} + \beta\overline{r}_u + \gamma\overline{r}_i$$

2.6 Linear Regression

Linear regression *(LR)* is one of the most popular regression models, which finds application in many different sciences (e.g. biology, finance and computer science). Typically *LR* is employed to predict what value will occur for a quantity of interest when other related variables take given values. In other words, regression analysis models the relationship between one or more independent variables and another dependent variable. This fact can be expressed in the form:

$$Y_i = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \ldots + \beta_p X_{ip} + \varepsilon_i, \qquad (i = 1, \ldots, n)$$

where $X_{i1} \ldots X_{ip}$ are independent variables, Y_i is the dependent variable, and ε_i is an error residual. Typically regression models contain a set of *n* samples, which can also be considered as a system of equations. In order to find an approximated solution for the system of equations, linear regression usually tries to minimize the sum of the squared error residuals ($\sum_{i=1}^{n} \varepsilon_i^2$). In literature, this approach is widely known as *least square estimation* or *least square problem*.
Recently, several studies on collaborative filtering employed regression models to either learn normalized ratings or estimate item similarities [8, 23, 24]. In all cases regression analysis helped to improved prediction accuracy and led to a lower number of system parameters, which makes it a suitable extension for conventional rating estimation.

2.7 Summary

The amount of information available online is growing rapidly. Although this information is essential for many of our daily operations, its enormous quantity makes it challenging and time consuming to retrieve specific data [1]. For this reason, there is a great demand in software applications that address the information overload and support individuals in their decision making. It is necessary to improve performance and accuracy of existing recommendation techniques, and to find new ways in automatic information filtering. *"The Web, they say, is leaving the era of search and entering one of discovery "*[5].

[5] http://recsys.acm.org/ ACM Conference on Recommender Systems

Chapter 3

Problem Analysis

3.1 Challenge

As mentioned previously, we aim to develop a hybrid recommender that is able to achieve higher prediction accuracy than ordinary single component systems. In order to improve system performance, we are going to utilize both rating information and content features of the observed corpus. In our study we want to examine the well-known *MovieLens* [6] rating data in combination with the corresponding *IDMB* [7] movie features. Although our research merely concerns movie data, we seek to design an universal model that also can be deployed for other domains.

Generally collaborative recommender systems can be employed either to predict how much a user will like an item or to recommend a list of items to a user [13]. However, candidate selection for a *top-N* recommendation is often based on previously generated rating predictions, which implies that both techniques principally employ the same underlying algorithm (i.e. neighborhood-based or statistical methods). Be that as it may, our own research mainly deals with the prediction of unknown user-item ratings rather than item recommendation.

Before we can make any design decisions regarding our hybrid recommender, we first need to analyze all system constraints. In the following sections we want to discuss all problems that come along with each single system component.

3.2 Rating Information

For our research we merely employ rating data[1] from the *MovieLens* recommender system[2] to evaluate the accuracy of our prediction algorithm. *MovieLens* debuted in 1997 and by now contains more than 10,000 movies rated by approximately 70,000 users. There exist several different *MovieLens* datasets that were anonymized and published for research issues *(i.e. 100K, 1M, 10M)*. In order to allow fast program changes and a straightforward system development, we decided to investigate the *100K MovieLens* dataset, which contains ratings from 943 users given on 1682 movie items. The data was collected through the *MovieLens* web site during the seven-month period from September 19th, 1997 through April 22nd, 1998. Rating information is provided as a text file in the following format:

user id	item id	rating	timestamp
196	242	3	881250949
186	302	3	891717742
22	377	1	878887116
244	51	2	880606923

Table 3.1: Extract of Rating Data

Although there exist several approaches [25] that successfully employ the time factor for collaborative filtering, we are mainly interested in the triple $< userID, itemID, rating >$. This data can easily be converted into a two dimensional rating matrix, whereas the first dimension spans the number of users and the second dimension spans the number of items ($M_{943\times1682}$). As before, ratings are represented as numerical values. However, the resulting matrix *M* is extremely sparse, because only 100,000 ratings were observed and many of the user-item entries are just empty fields ($\emptyset$). Furthermore, due to the fact that some of the observed data need to be reserved for testing, only part of the available user-item ratings can be employed for the training algorithm. The *GroupLens Research Group* provide a 90.57% to 9.43% split for training and user data *(training ratio* $t \approx 0.9$*)*, which gives us 90,570 ratings for the baseline algorithm and another 9,430 ratings for evaluation. Based on this numbers we can calculate the fill rate of our matrix *M*:

[1] MovieLens Datasets: http://www.grouplens.org/node/73
[2] MovieLens Recommendation Website: http://www.movielens.org

$$fillRate = \frac{numRatings * 100\%}{numUsers * numItems} = \frac{90570 * 100\%}{1682 * 943} \approx 5.7101\%$$

That means, not even 6% of all entries in the rating matrix are filled. In order to get a better picture of the rating information, we want to illustrate the sparsity pattern of the training data in Figure 3.1. All colored dots represent known rating values and white areas depict unknown rating information, whereas users and items were ordered according their amount of ratings.

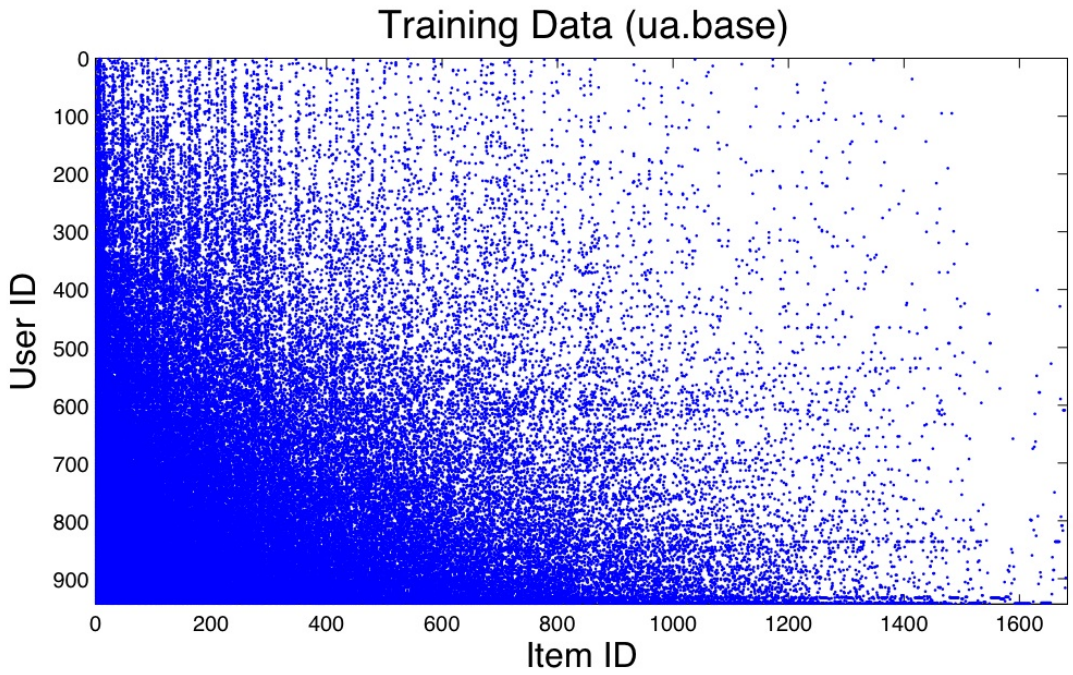


Figure 3.1: Sparsity Pattern of MovieLens Data

The rating information in Figure 3.1 might seem slightly denser than it is in reality. This is due to the low resolution and the slightly enlarged point size for the known ratings plotted. However, Figure 3.1 reveals that most users only rated a small portion of the items available, and most items just received modest user feedback. Note that users as well as items of the original rating matrix were reordered according to their amount of given or rather received ratings.

Figure 3.2 illustrates the rating distribution for both users and items. Obviously the amount of ratings increases slowly with a decreasing number of users or rather items, which exactly express the *power law* (also known as 80-20 rule).

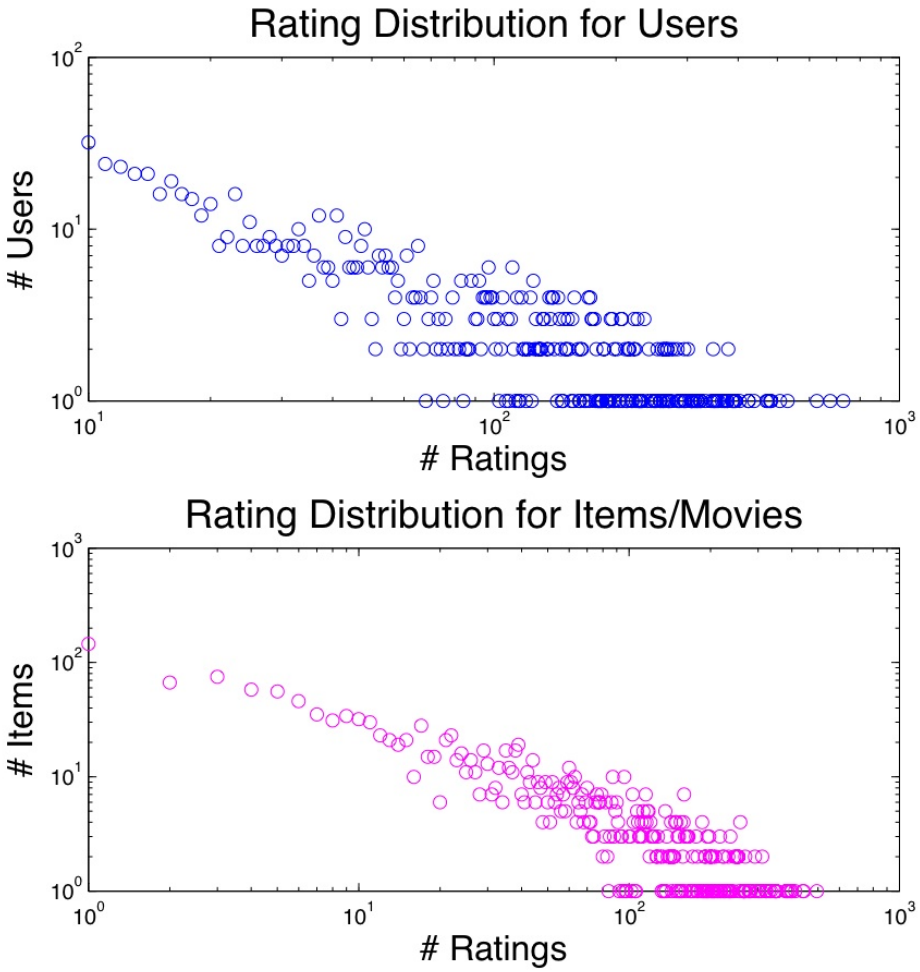


Figure 3.2: Rating Distribution of Training Data (Logarithmic Scale)

In compliance with the *GroupLens Research Project* and our own verification, each user in the investigated dataset gave at least 20 ratings. With exactly 10 ratings per user in the test set and the remaining ratings in the training set. However, this is not correct for the number of ratings per item. We found that many items just received a very small number of ratings, or even just obtained one rating by a single user. Since we want to discover nearest neighbors based on item similarity, the lack of rating information in item space might get us into a predicament. Hence, we are going to implement both user-based and item-based *NNH* approach to compare their prediction accuracy.

3.3 Content Features

Beside rating information the *MovieLens* dataset furthermore contains demographic information about users and feature information about movie items (see Figure 4.2). For our research we are mainly interested in additional item features, which help to give a more precise item description. Some sample records of the *MovieLens* item file are illustrated in the following table:

movie id	movie title	release date	genre00	...	genre18
53	Natural Born Killers (1994)	01-Jan-1994	1	...	0
54	Outbreak (1995)	01-Jan-1995	0	...	1
55	Professional, The (1994)	01-Jan-1994	1	...	0
56	Pulp Fiction (1994)	01-Jan-1994	1	...	1

Table 3.2: Extract of *MovieLens* Item File

The last 19 fields represent different genres, whereas a 1 indicates the movie is of that genre and a 0 indicates it is not. It is possible that movies can be in several genres at once. For instance the movie *Pulp Fiction* belongs to the categories *Crime* and *Drama*. The movie IDs are those used for the rating data (Figure 3.1). However, we had to find out that several movie titles in the *MovieLens* item file appear twice. Hence, it is not feasible to allocate each movie title with an explicit movie ID. This can be problematic when we are going to retrieve supplementary item feature from the *IMDB* by searching for titles that are redundant. Fortunately, the number of redundant titles is quite small and its influence on the prediction accuracy will be insignificant.

In general movie features can be retrieved from the *Internet Movie Database (IMDB)* in several ways. We decided to create a copy of the *IMDB* data files on our local system to avoid performance loss due to unreliable network connections. Each of the examined *IMDB* files contains information about an independent item feature. The single files do not stay in any relationship to each other and all have dissimilar formats for their data records. As a consequence, we need to parse each single feature file in a different way. In order to overcome this hurdle we have to build our own parser that can handle all necessary file formats.

Despite that, we have to decide which movie features to use as support for the obtained rating information. Among the bulk of features some seem to more promising than others. For our further investigation we selected the following *MovieLens* and *IDBM* item feature as candidates:

MOVIELENS	
Release Date	Genre
IMDB	
Actor	Actress
Country	Director
Keywords	Producer
Production Company	Production Designer

Table 3.3: Selected Movie Features

Beside the selected features in Figure 5.1, IMDB comprise much more movie attributes. When choosing features we always need to have in mind that they should help to identify neighbors and distinguish unlike items. Hence, at best a single feature is able to group a medium-sized number of items, instead of categorizing all or nothing. What actual benefit these additional movie information has on the prediction accuracy will be determined by tests on the system performance, under usage of single features or rather feature combinations. However, before the unstructured content information of the selected movie feature can be employed for the prediction algorithm of our hybrid recommender, it need to be transformed into an appropriate target format.
In general, content-based recommenders seek to capture a user's interests in an individual profile [1]. This kind of profile typically contains information about the items a user likes. Furthermore, content-based recommenders collect information about items of likely interest. Information about items is often modeled in the same way like user interests, so that user profile and new items can be compared against each other easily. In case of textual content, many systems employ keyword vectors, with interest weights assigned to each term *(TFIDF scheme)* [14]. Two keyword vectors can be checked for correlation by a distance measure like cosine similarity [3].
However, content-based recommendation systems cannot make reliable recommendations if the content does not contain enough information to identify the items a user likes or rather dislikes. Another drawback of such systems is that they usually tend to over-fit and a user seldom receives recommendations that are unexpected or fortuitous. Moreover, textual content exhibit the significant problem of semantic meaning, where several different terms can stand for one and the same concept [14].
We are looking for a way to utilize our selected content features without inherit all discussed disadvantages. *Chapter 4* will explain our actual approach in more detail.

3.4 Hybridization Technique

As mentioned earlier, a well-though-out hybridization approach is critical for the success of our two component recommender system. There exist numerous methods to combine collaborative filtering recommender with content-based techniques, but probably not all of them will lead to same prediction accuracy. In *Section 2.3* we already have discussed several different types of hybrid recommender systems. However, these possible hybrid combinations are **not** applicable in each situation or for each kind of underlying recommender [10].
Due to the fact that collaborative filtering is a well-established way of rating prediction, we want to employ this technique for our actual recommender. In addition, content features of the contributing recommender will support the main recommendation component. The following figure displays designated hybrid architecture:

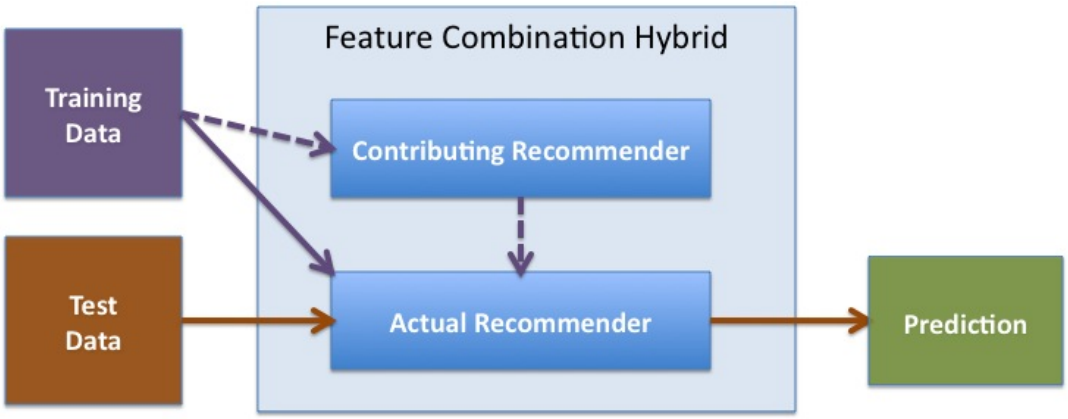


Figure 3.3: Hybrid Recommender System [10]

Content information forwarded by the contributing recommender should be in an appropriate data format to be processed by the actual collaborative recommender. Information loss through data transformation should be avoided.

3.5 Scalability

Building an elaborated recommender system means to achieve high prediction accuracy, but also implies an extremely flexible architecture that is scalable to data of different size. Although our research mainly deals with the comparatively small *100K MovieLens* corpus, we aim to design a hybrid recommender system which is able to handle large data amounts as well.

The major problem with large-scale data are memory management and computational effort. Commonly, the more data to be processed the higher the memory utilization and the more CPU time is needed. Imagine we would investigate the *Netflix* dataset with *100M* ratings, then it would already require about *700MByte*[3] memory space to simply store all entries. To say nothing of what happens when we try to employ any prediction algorithms on such an oversized corpus. For instance, the k-Nearest-Neighbor algorithm applied on a rating matrix $R_{m \times n}$ already has an approximate complexity of $O(mn)$ for similarity calculation only.

In order to avoid high computational complexity and excessive memory utilization, we are going to reduce the dimension of the rating data with singular value decomposition. To factorize the rating matrix into low dimensional segments seems to be a reasonable way to address the discussed problems. *Chapter 4* will further analyze the application of *SVD* for our hybrid recommendation approach.

3.6 Summary

This chapter identified some major problems concerning the combination of rating information and content features. Characteristics and specifics of both utilized data sources were examined in more detail. Moreover we discussed a possible hybridization technique to unify *MovieLens* ratings and *IMDB* content features in a single model. In addition time and memory constraints of real world recommendation systems were briefly introduced. The following chapters deal with the technical solution and the testing of our hybrid recommender system.

[3] $100M * (UserIndex + ItemIndex + Rating) = 100M * (32bit + 16bit + 8bit) \approx 700MBybte$

Chapter 4

Solution

4.1 System Architecture

Before we introduce each single component of our hybrid recommender, we first want to give a rough overview of the system architecture as a whole. *Figure 4.1* shows the main elements of our design, and illustrates the flow of information through the system.

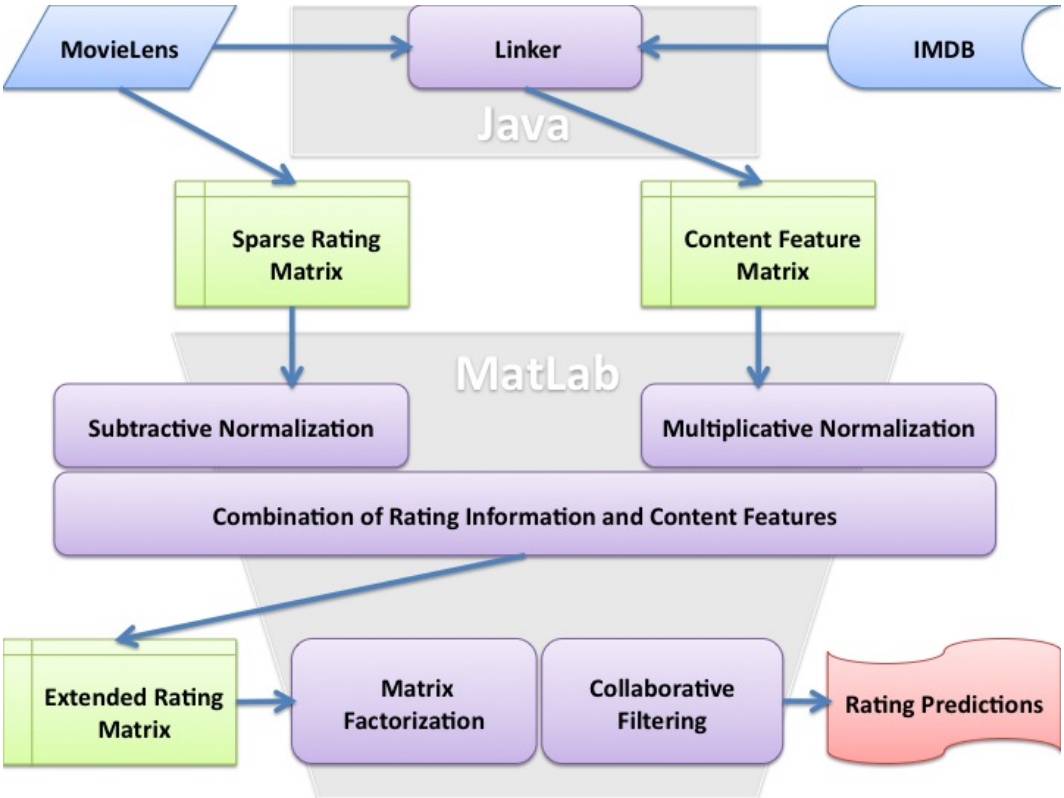


Figure 4.1: System Overview of Hybrid Recommender

As we can see in *Figure 4.1*, the *MovieLens* and *IMDB* data are joined through a *Linker* written in Java programming language. Retrieved content features and obtained rating information are normalized separately, before they are combined to a single rating matrix. This extended matrix is then passed through the factorization process to reduce the dimension of the data. Finally collaborative filtering is applied on the reduced data to predict the missing ratings. The following sections of this chapter will explain each step of our rating estimation approach systematically.

4.2 Data Linking

4.2.1 Entity Relation

The following diagram (Figure 4.2) shows all relationships among the data entities within the MovieLens [6] and IDMB [7] corpus. Furthermore it illustrates how both datasets are interconnected.

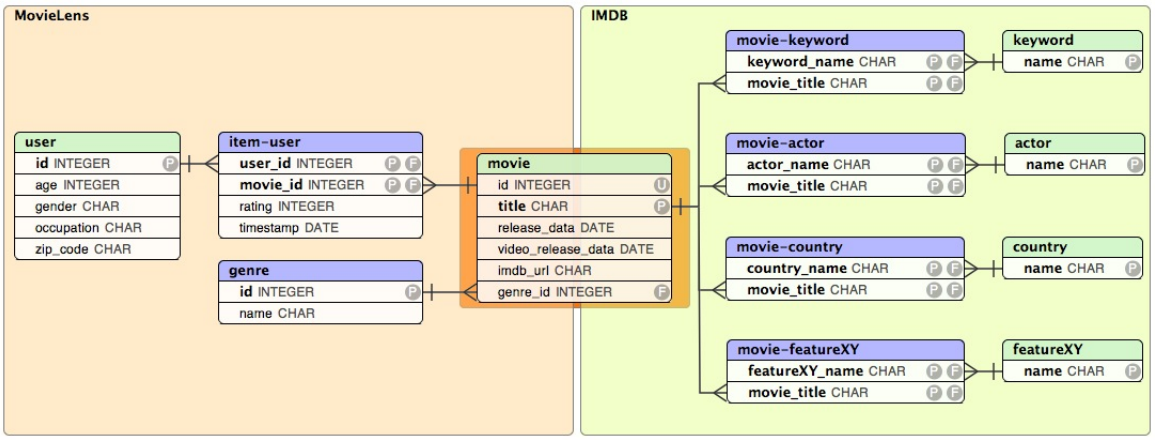


Figure 4.2: ER-Diagram of MovieLens and IMDB Dataset

Obviously the movie entity acts as a central interface between both examined datasets. All other entities are arranged around the interface in star-shape. Figure 4.2 furthermore expose that we can deduce supplementary movie features from a given user, or contrariwise infer user features from a designated movie. These additional features are the key point of our research, and will be utilized to improve accuracy of conventional collaborative rating prediction.

4.2.2 Bipartite Graph

Due to the fact that there does not exist any complex interdependencies between the data entities in Figure 4.2, the model can also be described as a bipartite graph. A possible presentation of the graph is illustrated in Figure 4.3, where only a small part of the entire network is shown.

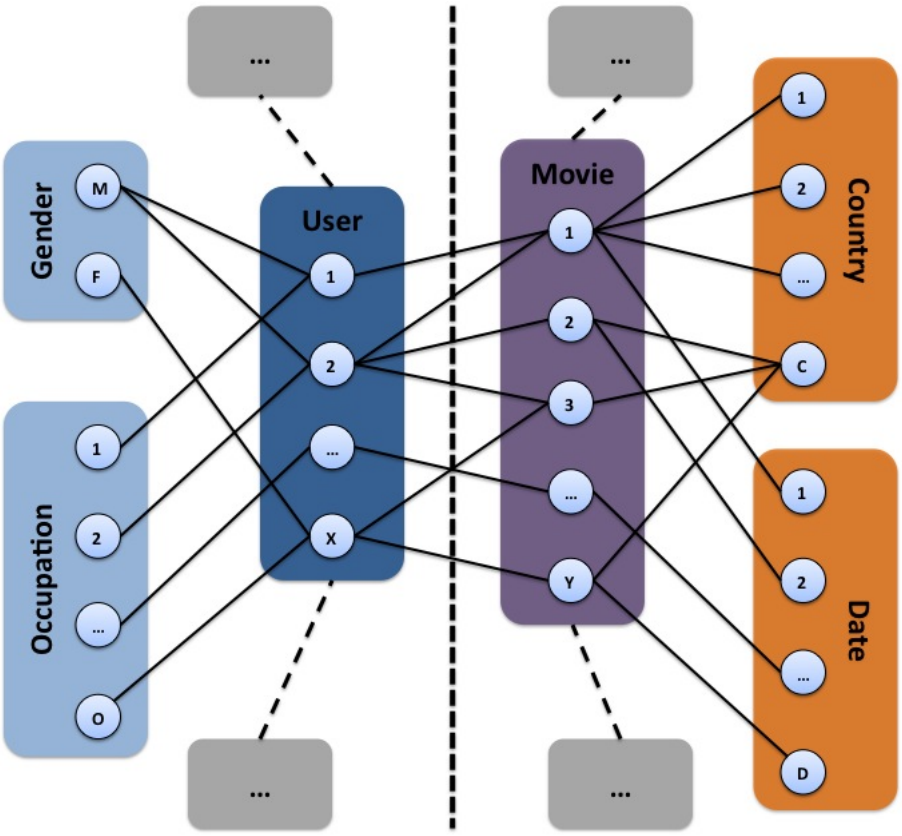


Figure 4.3: Bipartite Graph of Feature Relations

In Figure 4.3, blue points represent single items, also called *nodes*. They are interconnected with each other through *edges*, whereupon edges always link nodes of **unlike** entities *(eg. user <–> movie)*. In general there are three types of entity relations, namely *m:n*, *1:n* and *1:1*. For instance, a user normally only belongs to one gender, but many users can be either male or female *(1:n)*. However, a user can like many movies, and a movie can be enjoyed by many users *(m:n)*.
The simple network structure allows a straightforward data analyzation in the rating prediction step, whereas relations are captured in matrix format.

4.2.3 Feature Matrices

In order to get a better understanding of how to transform an entity relation into a feature matrix, we want to give an concrete example for the correlation between movies and countries.

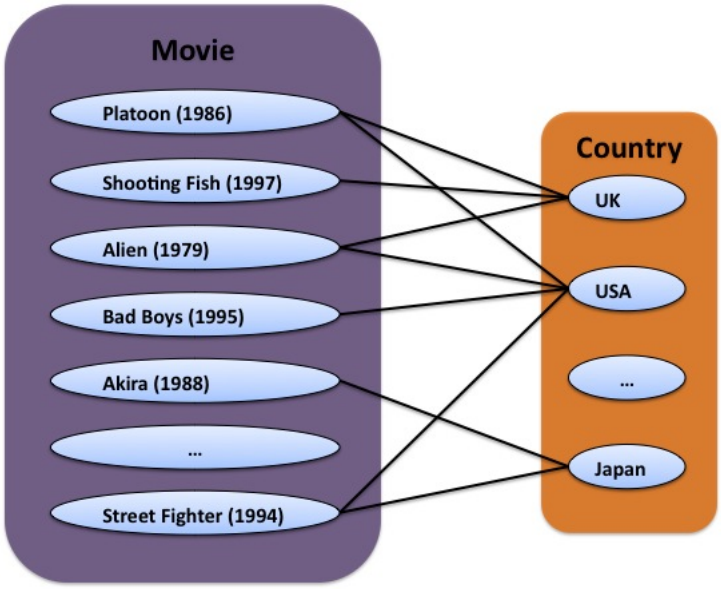


Figure 4.4: Snapshot of Movie-Country Relation

The item relations shown in Figure 4.4 are an extract of the actual *MovieLens* and *IMDB* dataset. In order to make this information usable for data mining techniques, we need to transform it into an appropriate format. Figure 4.5 illustrate the same item relations in a matrix notation, which can be utilized by collaborative filtering methods.

		Movie						
		PL	JA	AL	BB	AK	...	SF
Country	UK	1	1	1	0	0	0	0
	USA	1	0	1	1	1	0	1
	...	0	0	0	0	0	0	0
	Japan	0	0	0	0	1	0	1

PL .. Platoon
JA .. Jade
AL .. Alien
BB .. Bad Boys
AK .. Akira
SF .. Street Fighter

Figure 4.5: Extract of Movie-Country Feature Matrix

4.2.4 Feature Combination

In our approach we are mainly interested in the *Movie* and *User* entities, and their relations to any other available features. Possible movie features are actor, country and genre, as well as users that gave ratings on these items. From the perspective of a user we have the features gender, age and occupation, plus items that where rated by these users. Information about all existing relationships between different entities can be directly obtained from the ER-Diagram shown in Figure 4.2. Our goal is to combine the original rating matrix with all extracted feature information in a single model. Figure 4.6 illustrates such an extended matrix, where all entity relations already were transformed into matrix notation.

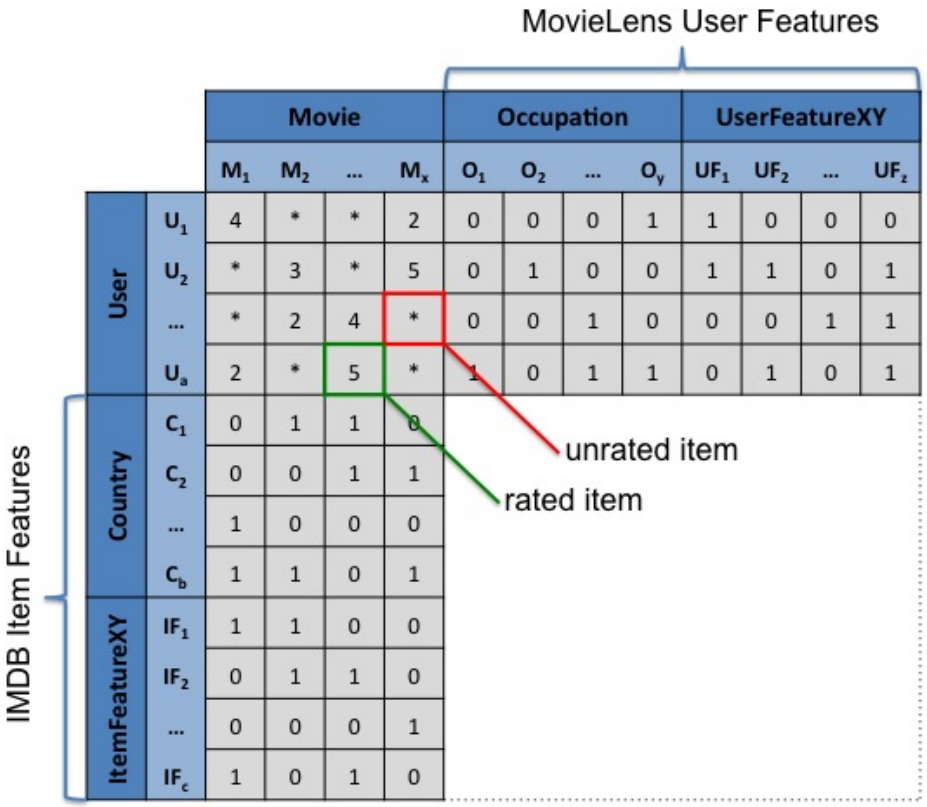


Figure 4.6: 2D-Matrix with User and Movie Features

After constructing an extended matrix like in Figure 4.6, we can apply collaborative filtering techniques to estimate missing user-item ratings. Inflating the original rating matrix with content-based features is expected to improve the performance for rating prediction. Experiments will show which features perform best (see Chapter 6).

4.3 Normalization

When we compare the items contained in our original *MovieLens* ratings matrix with the entries of our generated feature matrices, we will notice that both exhibit a different range of values. Where movie ratings can range from 1 to 5 *(zero if non-rated)*, content-features are either existent or not *(1 or 0)*. Hence, rating matrix and feature matrices both need to be normalized in a different way.

4.3.1 Subtractive Normalization

As mentioned earlier, *user-item* ratings usually exhibit different kinds of global effects. For instance, some users always tend to give higher ratings on items than other users, and some items at an average receive more positive user feedback than other items. In order to compute accurate rating predictions, global effects need to be removed from our data before applying any collaborative filtering techniques.
Typically a weighted combination of user-, item- and overall-average rating values is subtracted from the original entries to remove individual user preferences and item popularity effects. Figure 4.7 and the subsequent equations describe the principle of subtractive normalization.

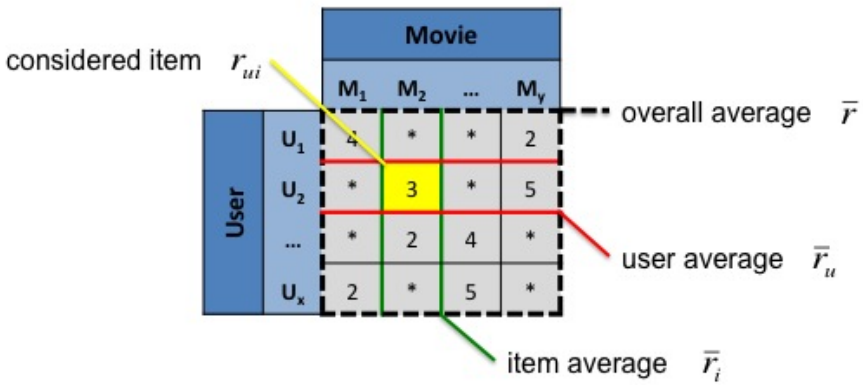


Figure 4.7: Subtractive Normalization of User-Item Rating Matrix

$$\begin{aligned}
\tilde{r}_{2,2} &= r_{2,2} - \alpha\overline{r}_{overall} - \beta\overline{r}_{user2} - \gamma\overline{r}_{item2} && (\text{let } \alpha, \beta, \gamma = \frac{1}{3}) \\
&= 3 - \frac{1}{3} \cdot \frac{(4+2)+(3+5)+(2+4)+(2+5)}{8} - \frac{1}{3} \cdot \frac{(3+5)}{2} - \frac{1}{3} \cdot \frac{(3+2)}{2} \\
&= 3 - \frac{1}{3} \cdot 3.375 - \frac{1}{3} \cdot 4 - \frac{1}{3} \cdot 2.5 \approx -0.389
\end{aligned}$$

In our above example we weighted all subtraction effects equally $(\alpha, \beta, \gamma = \frac{1}{3})$, whereas the weights sum up to one $(\alpha + \beta + \gamma = 1)$. The final normalized result for the rating entry $r_{2,2}$ of our example matrix is a negative value $(\tilde{r}_{2,2} = -0.389)$, which in turn can cause negative item or rather user similarities. This matter of fact need to be considered in the rating prediction step (see Section 4.5).
Typically all normalized rating have a value around zero $(-1 \leq \tilde{r}_{ui} \leq 1)$, which enables a comparison of the single ratings entries. The general equation for the subtractive normalization is formulated in the following:

$$\tilde{r}_{u,i} = r_{u,i} - \alpha\overline{r} - \beta\overline{r}_u - \gamma\overline{r}_i$$

with

$$\overline{r} = \frac{\sum_x \sum_y r_{xy}}{\#ratings(r)} \qquad \overline{r}_u = \frac{\sum_y r_{uy}}{\#ratings(u)} \qquad \overline{r}_i = \frac{\sum_x r_{xi}}{\#ratings(i)}$$

The parameters α, β and γ determine the influence of the observed effects on the final normalization result. Our purpose is to find a single parameter configuration which gives optimal normalization results for all entries of our original rating matrix. We decided to employ the well-known linear regression approach to solve the system of normalization equations. Mathematically the problem can be formulated as:

$$r_{u,i} = \alpha\overline{r} + \beta\overline{r}_u + \gamma\overline{r}_i + \epsilon_{u,i}$$

The residual $\epsilon_{u,i}$ is often characterized as disturbance term or error variable. One of the most popular methods to decrease the error residual is *Least Squares* (linear regression model), where the independent variables (α, β, γ) are chosen in a way that the sum of the squared residuals is minimized $(\sum_{i=1}^{n} E_i^2)$. The entire system of equations that need to be solved by the linear regression approach can be expressed in following way:

$$\begin{aligned} r_{1,1} &= \alpha\overline{r} + \beta\overline{r}_{u1} + \gamma\overline{r}_{i1} + \epsilon_{1,1} \\ r_{1,2} &= \alpha\overline{r} + \beta\overline{r}_{u1} + \gamma\overline{r}_{i2} + \epsilon_{1,2} \\ \dots &= \dots \\ r_{x,y} &= \alpha\overline{r} + \beta\overline{r}_{ux} + \gamma\overline{r}_{iy} + \epsilon_{x,y} \end{aligned}$$

4.3.2 Multiplicative Normalization

Our generated feature matrices need to be treated differently than the original *MovieLens* ratings matrix, because features entries are either 0 or 1, and therefore the row/column mean average values $(\overline{r}_u \ \& \ \overline{r}_i)$ are always 1. If we would employ subtraction to normalize the feature entries, all final rescaled values would be the same $(\tilde{r}_{u,i} = r_{u,i} - (\overline{r} + \overline{r}_u + \overline{r}_i) = r_{u,i} - (1 + 1 + 1)$. Hence, we need to find another general method that can be applied to all feature matrices.

Contrary to the subtractive method, multiplicative normalization rescales the feature matrix $F_{m\times n}$ based on row and column length $(||F_{rowX}|| \ \& \ ||F_{colY}||)$. The main idea behind the multiplicative technique is to normalize all matrix entries in a way that the sum of each row/column adds up to 1. How to approximate such a matrix is precisely explained in the following equations and Figure 4.8.

$$FN_{m\times n} = ROW_{m\times m}^{-1/2} \times F_{m\times n} \times COL_{n\times n}^{-1/2}$$

where

$$ROW_{x,x}^{-1/2} = \frac{1}{\sqrt{\sum_n F_{x,n}}} \quad \text{and} \quad COL_{y,y}^{-1/2} = \frac{1}{\sqrt{\sum_m F_{m,y}}} \tag{4.1}$$

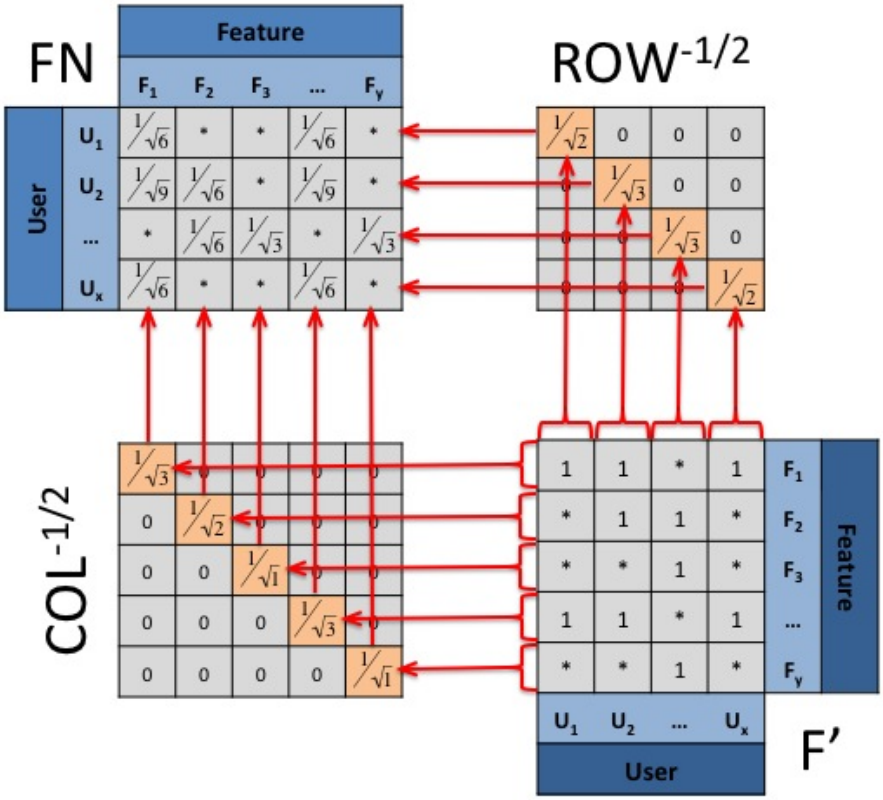


Figure 4.8: Multiplicative Normalization of User-Feature Matrix

For the purpose of better illustration, the initial feature matrix in Figure 4.8 is transposed (F') and placed in the lower right corner. The example matrix contains *'s instead of 0's, because nonexistent features do not need to be considered. In the main diagonals of the *ROW* and *COL* matrix, we find the multipliers for the single feature entries. These multipliers originate from the corresponding rows or rather columns of the initial feature matrix (refer to Equation 4.1). In addition, all off-diagonal elements of both multiplication matrices are filled with zeros.

To get a better understanding of multiplicative normalization we want to have a look at the regularization of the concrete matrix entry $F_{2,2}$. User/row number two (F_{row2}) exhibits three nonzero elements, wherefore the respective row multiplier is three to the power of minus one half ($ROW_{2,2}^{-1/2} = \frac{1}{\sqrt{1+1+1}} = \frac{1}{\sqrt{3}}$). Furthermore the second column (F_{col2}) exhibits two nonzero elements, which gives two to the power of minus one half as column multiplier ($COL_{2,2}^{-1/2} = \frac{1}{\sqrt{1+1}} = \frac{1}{\sqrt{2}}$). Finally both multipliers are combined in one equation:

$$FN_{2,2} = ROW_{2,2}^{-1/2} \cdot F_{2,2} \cdot COL_{2,2}^{-1/2} = \frac{1}{\sqrt{3}} \cdot \frac{1}{1} \cdot \frac{1}{\sqrt{2}} = \frac{1}{\sqrt{6}} \approx 0.41$$

4.4 Matrix Factorization

Beside analyzing diverse movie and user features, our research is also concerned with matrix factorization, and its impact on prediction accuracy. Typically matrix factorization techniques are employed to reduce the dimension of the item space and/or to retrieve latent relations between items of the observed dataset. In our work we want to investigate the well-known *Singular Value Decomposition (SVD)* method, which factorizes the original rating matrix into three low-dimensional matrices containing the left-singular vectors, the singular values and right-singular vectors respectively ($R = U \cdot S \cdot V'$; refer to Section 2.4 and Figure 2.2). The resulting matrices can be utilized for rating prediction in several different ways.
Due to the fact that the new generated matrices can be considered as an approximation of the original rating matrix, they can be employed to directly estimate missing rating values. For a particular matrix entry $r_{u,i}$, this is done in the following way:

$$\hat{r}_{u,i} = X(u) \cdot Y(i) \qquad (X = U_k S_k^{1/2} \ \ \& \ \ Y = S_k^{1/2} V_k')$$

The compound matrices X and Y represent user and item concepts respectively. Unknown user-item ratings are predicted by computing the dot-product between the appropriate user and item concept. Note that all estimated ratings need to be de-normalized by adding back the user and item effects.
Instead of multiplying the compound matrices to directly estimate missing ratings, they also can be considered separately for the user-oriented or rather user-oriented collaborative filtering approach. Due to the fact that the user/item vectors of the compound matrix X/Y are much shorter than in the original rating matrix *(reduced to length k)*, computing the distance *(cosine-similarity)* between different neighbors *(users/items)* is less computational expensive. Although the reduced user/item vectors contain less values, the information they hold is more pregnant. We believe that if the original rating matrix was inflated with additional feature information before factorization, user-concept (X) or rather item-concept (Y) matrix will have higher explanatory power. To what extent the reduced user/item vectors help to improve prediction accuracy will be determined through experiments (see Chapter 6).

4.5 Collaborative Filtering

Neighborhood-based algorithms (like *k*-NN) have a long tradition in collaborative filtering systems [4] and also constitute an important core element in our own approach. This section will give an insight into operation method and mathematical concept of item-oriented as well as user-oriented design. In order to get a better understanding understanding of the *kNN* algorithm, we want to give the following example:

	Forrest Gump	Pulp Fiction	Toy Story	Star Wars	
Amy	3	-	4	2	
Marc	4	5	-	4	
Sue	-	4	2	3	
Paul	3	5	2	-	

Table 4.1: Example Rating Matrix

All users in *Table 4.1* have rated three out of four movies. Likewise each movie received ratings from three different users. Although our example rating matrix is quite small and has a simple structure, it should be enough data to explain the principles of the neighborhood approach.

4.5.1 User-Oriented Method

Lets assume that *Amy* wants to watch the movie *Pulp Fiction*, but she does not know if she would like it or not. Intuitively *Amy* would first ask some friends of her about their opinion. *Marc*, *Sue* and *Paul* all have seen the movie *Pulp Fiction* before. However, probably all of them would give different suggestions to *Amy*, because each has an unlike taste about movies. In real life it is often challenging to find out to which degree we should trust other people's opinion. Most times we listen to people that have similar thoughts than we have, which is exactly the principle of the user-oriented nearest neighbor algorithm. In *Table 4.2* we can see the complete *user-user* distance matrix, which was calculated based on the cosine-similarity ($sim[vecA, vecB] = \frac{vecA \cdot vecB}{|vecA| \cdot |vecB|}$) of the user/row vectors.

	Amy	Marc	Sue	Paul
Amy (A)	-	0.4919	0.4828	0.5121
Marc (M)	0.4919	-	0.7871	0.7950
Sue (S)	0.4828	0.7871	-	0.7230
Paul (P)	0.5121	0.7950	0.7230	-

Table 4.2: User-User Similarity Matrix

Obviously the user similarity matrix in *Table 4.2* is symmetric. For instance, the similarity between *Amy* and *Marc* is the same like between *Marc* and *Amy*. We are able to safe computational time by only calculating the upper triangular of the similarity matrix. In addition, we do not need to consider the main diagonal, because a user cannot have a distance to itself. Some applications furthermore prune the similarity matrix by only calculating those entries needed for rating prediction. In our case we just need the row of user *Amy* for further computations.
To get the final prediction for *Pulp Fiction (PF)* we have to compute the mean of all user ratings on this movie. Note that all ratings are first multiplied with their according user similarity value before they are summed up. This computation can be formulated in the following way [4]:

$$pred(u,i) = \frac{\sum_{n \subset neighbors(u)} userSim(u,n) \cdot r_{ni}}{\sum_{n \subset neighbors(u)} |userSim(u,n)|}$$

$$pred(A,PF) = \frac{uSim(A,M) \cdot r_{M,PF} + uSim(A,S) \cdot r_{S,PF} + uSim(A,P) \cdot r_{P,PF}}{|uSim(A,M)| + |uSim(A,S)| + |uSim(A,P)|}$$

$$pred(A,PF) = \frac{0.4919 \cdot 5 + 0.4828 \cdot 4 + 0.5121 \cdot 5}{|0.4919| + |0.4828| + |0.5121|} \approx 4.6763$$

Usually an user has many neighbors, which can be consulted for rating prediction. However, most neighborhood-based recommender systems limit the number of accounted neighbors, because neighbors based on a small number of overlapping items tend to be a bad predictor [11, 2]. Hence, the well-known *k-nearest-neighbor* algorithm only consider the *k*-top similar users or rather items.

4.5.2 Item-Oriented Method

User-oriented methods predict unknown ratings based on information of like minded users [8]. Although this approach was most common in early *CF* system, recently item-oriented methods became more and more popular. Those methods make predictions using known given by the same user on similar items. In general, item-oriented methods are more reasonable, because users are more familiar with items previously preferred than with potentially like minded users.
In many cases [13, 26, 27] item-oriented collaborative filtering recommendation systems show better scalability than ordinary user-oriented methods and furthermore improve prediction accuracy. For this reason, we decided to focus on the item-oriented method for our own approach. In the following we want to illustrate the algorithm of item-oriented rating prediction. Again, we first need to compute the similarities of the neighborhood items, before their ratings can be employed for estimation.

	Forrest Gump	Pulp Fiction	Toy Story	Star Wars
Forrest Gump (FG)	-	0.7389	0.6301	0.7006
Pulp Fiction (PF)	0.7389	-	0.4523	0.7314
Toy Story (TS)	0.6301	0.4523	-	0.5307
Star Wars (SW)	0.7006	0.7314	0.5307	-

Table 4.3: Item-Item Similarity Matrix

$$pred(u,i) = \frac{\sum_{j \in ratedItems(u)} itemSim(i,j) \cdot r_{uj}}{\sum_{j \in ratedItems(u)} |itemSim(i,j)|}$$

$$pred(A,PF) = \frac{iSim(PF,FG) \cdot r_{A,FG} + iSim(PF,TS) \cdot r_{A,TS} + iSim(PF,SW) \cdot r_{A,SW}}{|iSim(PF,FG)| + |iSim(PF,TS)| + |iSim(PF,SW)|}$$

$$pred(A,PF) = \frac{0.7389 \cdot 3 + 0.4523 \cdot 4 + 0.7314 \cdot 2}{|0.7389| + |0.4523| + |0.7314|} \approx 2.8548$$

Employing the item-oriented approach, the final rating prediction for our tuple *<Amy,PulpFiction>* is quite different to what we calculated before. Whether user-oriented or item-oriented strategy performs better for our own recommender, will be evaluated through extensive system tests.

4.6 Summary

Many recommendation systems combine collaborative and content-based filtering techniques to improve rating prediction. However, our approach is special in that we unify user-item ratings and content features in a single model/matrix *(Feature Combination)*, which is exploited by nearest neighborhood techniques *(NNH-algorithm)* subsequently. Moreover our research gives attention to both user and item features, where the impact of single and joined features is analyzed (see Chapter 6).

In order to decrease the computational runtime and memory usage of our system implementation, we furthermore employ matrix factorization *(SVD)* on the hybrid model. Although the resulting low-dimensional matrices are just an approximation of the original rating matrix $(R \approx U \cdot S \cdot V')$, they are less sparse and reveal hidden user (U) or rather item (V') relations. Beside that, we expect the injected features to have a positive influence on the explanatory power of the decomposition, because additional features contribute to a more precise differentiation of the single users and items respectively.

The main purpose of our research is to find out which features are beneficial to rating prediction, and in what extend matrix factorization is advantageous to our approach. Furthermore we face the challenge to determine the optimal parameter setting of our constructed recommender system for the observed *MovieLens* dataset. In Chapter 6 we are going to evaluate different variants of our hybrid solution. However, before we first want to introduce our implementation in Chapter 5.

Chapter 5

Implementation

In this chapter were are going to introduce the implementation of our previously formulated hybrid recommendation approach within a concrete development environment. Specifics that emerge during system implementation does not affect the applicability of our solution proposed before (refer to Chapter 4).
Our implementation can be divided into two main components, whereas the first one is written in *Java* programming language and the second one is realized with *MATLAB* logic (see Figure 4.1). Content features extracted by our *Java* program are passed to the *MATLAB* component, which employ these information for rating prediction.
The first section of this chapter will discuss feature retrieval by means of our developed *Java* tool, and furthermore give reasons for feature selections. In the second section we are going to conceive different implementation of the underlying rating prediction algorithm. All formulated implementation variants will be analyzed in terms of prediction accuracy and computational complexity in Chapter 6. In the end we will explain our novel approach to feature relation extraction, whereas latent associations between user and item features are retrieved.

5.1 Feature Retrieval

In order to utilize supplementary user/item features for our rating prediction algorithm, we developed a Java tool, which converts the corresponding *MovieLens* and *IMDB* information into valuable feature matrices. According to the introduced hybrid approach (refer to Chapter 4), the obtained feature matrices are combined with our original rating matrix subsequently. In the following we are going to talk about how to retrieve features, and which features to select for further processing.

5.1.1 Java Program

Our java program is divided into two different source packages, whereas one package contains all the classes and methods to extract the item features, and the other one contains all functionality to retrieve the user features. Even though the source code of both packages basically comprise the same operations, parsing item and user feature files exhibit specifics. However, in this section we are going to explain feature retrieval only from the perspective of movie items. To be more precise, we want to discuss the construction of the *Movie-Country* feature matrix (also see Appendix 7.2).
No matter what item feature to extract, our program always first parse the movie file ('item.u') to gain those items that need to be consider further on. The movie file provided by *MovieLens* has the following format: <*Movie ID, Movie Title, Release Date, Video Release Date, IMDB URL, Genre00, ..., Genre18*>. In case we want to extract the *Movie-Country* item relations, we just require information about *'Movie ID'* and *'Movie Title'*. Whereas the *'Movie ID'* attribute explicitly identify a movie within the original user-item rating matrix, and the *'Movie Title'* attribute represents an interface to the *IMDB* movie content data source. Our program stores both attributes in a *Java Hash Map* data structure (<*Movie ID, Movie Title*>), so that they can be associated later on.
In the next step we need to parse the corresponding *IMDB* feature file ('countries.list'), whose format looks like: <*Movie Title, Country*>. Due to the fact that we do not know what possible values the attribute country can take, our program first needs to skim through the document to find all occurring features. Same like the movie title, each feature is assigned with an ID and is stored in a *Java Hash Map* data structure (<*Country ID, Country Name*>). After we have determined the number of movies and countries, a feature matrix of appropriate size can be constructed ($\#movies \times \#countries$; or more general $\#items \times \#features$).

Since our features matrix has a relatively low number of nonzero elements, we better use a specific storage format for sparse matrices that deal with the available memory in a more economical way [28]. For our implementation we employ the third party tool *MTJ (Matrix Toolkit for Java)*[1], which is a comprehensive collection of matrix data structures and associated computations (least square methods, matrix decomposition, etc.). *MTJ* is available as free Java library, and can be legally referenced for our own project. To store our sparse feature matrix, we use the *MTJ-FlexCompRowMatrix* class, which puts the information row-wise into sparse arrays *(Java Sparse Array - JSA)*. This data structure can be imagined in the following way:

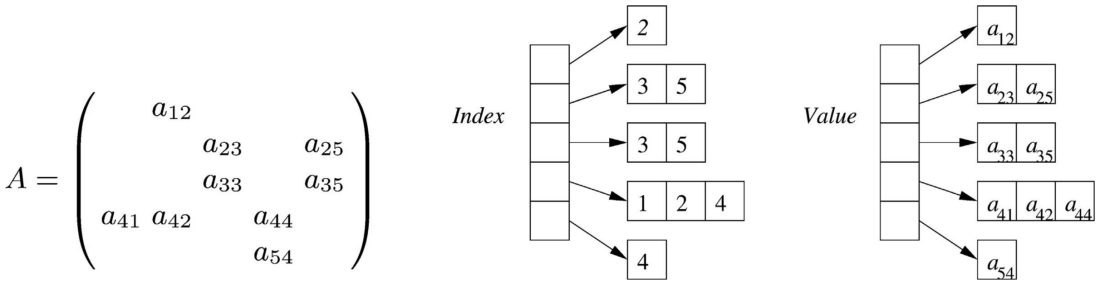


Figure 5.1: Example of Java Sparse Array (JSA) [28]

Our feature matrix is initially empty, and then filled stepwise by parsing through each line of the *IMDB* item-feature file. In case of our observed country feature this is the same document then examined before ('countries.list'), but commonly both files are different for other features. For each line of the *IMDB* item-features file our program checks whether or not the current movie title appear in the *MovieLens* item file. If so, our feature matrix gets an entry with value one at the position for the current movie-country pair $(M_{CountryID,MovieID} = 1)$.

Due to the fact that some features do not appear for the items of the *MovieLens* dataset, a couple of rows within our feature matrix might be empty. These empty rows can be deleted, because they do not contribute to distinguish or rather describe our movie items. The information contained in the remaining rows is exported to a simple text file, which is taken as input by our *MATLAB* program for further processing.

A simplified version of the source code of the above described Java program can be found in Appendix 7.2.

[1] Official MJT Website - http://ressim.berlios.de/

5.1.2 Feature Selection

Besides retrieving user/item features, our Java program furthermore calculate the sparsity of the individual feature matrices. We found that some of the generated feature matrices are not beneficial to our hybrid recommendation approach, because they are too sparse to precisely describe the corresponding users and items respectively. We believe that each user/item feature vectors of our generated features matrices should contain at least two nonzero entries (better more), which group or rather classify the particular users/items. The following table shows the dimension, number of entries and resulting fillrate of all constructed feature matrices:

Matrix	Dimension	#Entries	Fillrate in %
Movie-Actor	22240×1682	32542	.0869
Movie-Actress	10942×1682	14676	.0797
Movie-Country	42×1682	1683	2.3922
Movie-Date	72×1682	1663	1.3881
Movie-Director	861×1682	1322	.0913
Movie-Genre	19×1682	2863	9.0525
Movie-Keyword	12121×1682	52363	.2568
Movie-Producer	9793×1682	6533	.1028
Movie-Production-Company	1515×1682	3359	.1343
Movie-Production-Designer	511×1682	1027	.1195
User-Ages	61×943	943	1.6393
User-Gender	2×943	943	50.0000
User-Occupation	21×943	943	4.7619
User-Zipcode	795×943	943	.1258

Table 5.1: Sparsity of Feature Matrices

As we can see in Table 5.1, some of the user/item features matrices contain less than 1% nonzero elements, which is definitely not enough to increase the explanatory power of our original ratings matrix. We decide to exclusively select these features matrixes that exhibit a fillrate higher than the 1% limit. For our item features these are the *Movie-Country*, *Movie-Date* and *Movie-Genre* matrices. Additionally, we selected the *User-Ages*, *User-Gender* and *User-Occupation* matrices as user features for the further processing by our hybrid recommendation system.
We believe that the *User-Zipcode* matrix would be applicable as well, if all zip-codes would be grouped according to their respective states. However, this would go beyond the scope of our research.

5.2 Rating Prediction

Our program for rating prediction was completely written in the *MATLAB* programming language. The following sections will describe all parts of our recommendation system and introduce the discrete prediction variants we implemented.

5.2.1 Data Input

Our *MovieLens* training and test data, as well as our extracted feature information, are all expressed in three column notation. Where the first column describes the row index, the second column describes the column index and the third column express the value of the appropriate matrix entry ($[i, j, v]$; also refer to Table 3.1). However, for further examination of the data we necessarily need to transform this information into matrix format. *MATLAB* offers several different ways to convert an ASCII file into a sparse matrix. The following source code example illustrates how to load and convert the data:

```
1 M = load('rating.data');
2 Rows = M(:,1);
3 Cols = M(:,2);
4 Vals = M(:,3);
5 MS = sparse(Rows,Cols,Vals,m,n));
6 save -append IO_File MS;
```

With the command *sparse()* we are able to create a $m \times n$ matrix with space allocated for the number of nonzero elements only. This representation allows us to apply all *MATLAB* build-in arithmetic, logical, and indexing operations. The *save* command stores computed results in a *IO-File*, which can be loaded again for subsequent operations in other system modules.

5.2.2 Rating Interpolation

One of the most straightforward methods to rating prediction is interpolation, where missing values of the ratings matrix are estimated based on information already known. In our own interpolation approach we intend to predict all missing ratings regarding their user average ($\overline{r}_u$) and item average ($\overline{r}_i$) value, as well as the overall matrix average ($\overline{r}$) value (refer to Section 4.3.1 or Figure 4.7 respectively).

First of all we employ linear regression to find the parameter configuration that gives the best possible approximation of our known ratings ($r_{ui} = \alpha\overline{r} + \beta\overline{r}_u + \gamma\overline{r}_i + \epsilon_{ui}$). Based on the determined normalization parameters we can then interpolate the unknown user-item ratings ($\hat{r}_{ui} = \alpha\overline{r} + \beta\overline{r}_u + \gamma\overline{r}_i$) (also refer to Section 2.6 and 4.3.1). In our *MATLAB* implementation this looks like:

```
1  R = load('rating.data');
2  R1 = R(:,1);  R2 = R(:,2);  R3 = R(:,3);
3
4  load IO_File userAvgArray itemAvgArray overallAvg;
5  U = userAvgArray(R1);
6  I = itemAvgArray(R2);
7  O = ones(numel(R3),1)*overallAvg;
8
9  X = [U I O] \ R3;
10 a = X(1); b = X(2); c = X(3);
11
12 predRat(u,i) = a*userAvgArray(u) + b*itemAvgArray(i) + c*overallAvg;
```

In our above source code extract, the average values for user and item ratings, as well as the average value for the overall matrix, were already pre-calculated and are loaded into the program from the *IO-File*. Afterwards our program computes the single columns for the system of equations *(U, I and O)*, which is then solved by linear regression ($[UIO] \backslash R3$). As mentioned before, the determined parameters ($\alpha\ \beta\ \gamma$) can then be employed to predict any unknown user-item rating (see Line 12).
Besides predicting missing ratings, the learned parameters can furthermore applied to regularize known ratings (refer to Section 4.3.1). Therefore, we will use the calculated parameter configuration for the normalization and de-normalization part of all following rating prediction approaches as well.

5.2.3 Singular Value Decomposition (SVD)

For the following *SVD* recommendation approach we assume that all values within our original rating matrix *R* were already normalized. The main difficulty concerning our matrix factorization strategy is, that we have to find the optimal setting for the parameter *k*, which determines the dimension of the resulting matrices (refer to Section 2.4 and 4.4, as well as Figure 2.2). Unfortunately there does not exist any elegant regression model that is able to find the optimal dimension automatically, rather we have to determine *k*

by multiple test runs. In the following we illustrate a shortened version of the matrix factorization in *MATLAB* code:

```
function r = factorizeMatrix(k)
load IO_File R;

[U,S,V] = svds(R,k);

X = U*sqrt(S);
Y = sqrt(S)*V';

predRat(u,i) = dot(X(u,:),Y(:,i));
```

Our above matrix factorization program loads the normalized rating matrix *R* from the *IO-File*, whereas the value of parameter *k* for the current iteration is retrieved as input at function call. Both variables serve as input parameters for the *MATLAB* build-in singular value decomposition method for sparse matrixes (*svds(R,k)*), which calculate the *k* largest singular values and singular vectors of *R* ($U_{m\times k} \cdot S_{k\times k} \cdot V'_{n\times k} = \tilde{R}_k$). Where the left singular vectors contained in *U* describe the latent user concepts and the right singular values in *V'* represent the hidden item concepts respectively.
In order to predict unknown user-item ratings, our program at first compute the two compound matrices *X* and *Y*, which are constructed from the factorization output ($X = U_k S_k^{1/2}$ & $Y = S_k^{1/2} V'_k$). These compound matrices are then employed to estimate missing ratings by calculating the dot-product between the appropriate user and item concept ($\hat{r}_{ui} = X(u) \cdot Y(i)$). As mentioned earlier, the precision of the estimated ratings strongly depends on the dimension of the decomposed matrices.

5.2.4 SVD with Content Features

In this section we want to extend the previously discussed *SVD* approach by inflating our original ratings matrix with additional content features. As preconditions we assume that the normalized rating matrix and all generated feature matrices were already imported into our program, and are stored as variables in the default *IO-File*. However, the feature values still need to be regularized by multiplicative normalization (refer to Section 4.3.2 and Figure 4.8). A simplified version of our implementation for the *SVD* approach with *Content-Features* is illustrated in the following:

```
1  function r = contentFeatures(w)
2  load IO_File k R F ROW COL;
3
4  numFeats = numel(FeatMatrix(:,1));
5  numItems = numel(FeatMatrix(1,:));
6
7  for x = 1:numFeats
8    for y = 1:numItems
9      if F(x,y) ~= 0
10       F(x,y) = (ROW^(-0.5))(x) * F(x,y) * (COL^(-0.5))(y);
11     end
12   end
13 end
14
15 FW = F*w;
16 RFW = [R;FW];
17
18 [U,S,V] = svds(RFW,k);
19 ...
```

First of all the dimension parameter k, as well as the rating matrix R and feature matrix F are loaded into our program. Furthermore, the matrices ROW and COL (refer to Section 4.3.2), whose main diagonals contain the number of nonzero elements appearing in the rows or rather columns of F ($ROW_{ff} = \sum_i F_{fi}$ & $COL_{ii} = \sum_f F_{fi}$), are read from the *IO-File*. Having these variables in memory, our program is able to normalize all feature values ($\dot{F}_{m\times n} = ROW^{-1/2}_{m\times m} \times F_{m\times n} \times COL^{-1/2}_{n\times n}$).
Moreover, the above function retrieves the parameter w, which acts as weight for the currently processed content feature. Every single value of our observed feature matrix is multiplied with the current weight ($\ddot{F} = \dot{F} \cdot w$; see Line 15), whereas w changes in each iteration of our program. In general, the higher the retrieved weight the higher the influence of the current feature on our prediction results. Our aim is to find the optimal weight configuration for each examined user or rather item feature.
In the next step the weighted content features are combined with our original rating matrix. Depending on the feature type, the normalized and weighted feature matrix is either attached towards the existent item or user vectors ($RF = [R; F]$ or $RF = [R, F]$; also refer to Section 4.2.4 and Figure 4.6). In case that item and user features are processed simultaneously, we need to complete the lower right part of our extended matrix with zero values to keep the designated rectangular form ($RF_{(m+x)\times(n+y)} = [R_{m\times n}, UF_{m\times y}; IF_{n\times x}, Zeros_{x\times y}]$; where *UF* and *IF* represent user or rather item features).
After we combined original ratings and content features, the extended matrix is utilized

for rating prediction. The estimation of unknown ratings is done in the same way as in our straightforward *SVD* approach discussed previously. First of all the extended model is factorized ($\widetilde{RF}_k = U_{m\times k} \cdot S_{k\times k} \cdot V'_{n\times k}$), after what the resulting components are joined to compound matrices ($X = U_k S_k^{1/2}$ and $Y = S_k^{1/2} V'_k$). As before, we can predict any missing rating value by simply multiplying the appropriate row and column vector of the compound matrices ($\hat{r}_{ui} = X(u) \cdot Y(i)$). Note that all estimated ratings need to be de-normalized by adding back the user and item effects ($\tilde{r}_{u,i} = \hat{r}_{u,i} + \alpha\overline{r} + \beta\overline{r}_u + \gamma\overline{r}_i$; refer to Section 2.5).

5.2.5 Nearest Neighborhood Algorithm (NNH)

Most present collaborative filtering recommendation systems employ some variation of the *Nearest Neighborhood Algorithm (NNH)*, where either similar users or items are utilized to predict unknown ratings (refer to Section 2.1 and 4.5). In this section we want discuss the item-oriented *NNH* approach by means of our own *MATLAB* implementation. For the purpose of estimate missing ratings, we first need to compute the similarity between all occurring items. Usually, item similarity values are pre-calculated and stored in an appropriate matrix. The following source code extract shows how to generate and fill such a data structure.

```
load IO_File R nbh;

numItems = numel(R(1,:));
ISM = zeros(numItems,numItems);

for x = 1:numItems
  for y = 1:numItems
    if x <= y
      break;
    else
      ISM(x,y) = cosineSim(R(:,x),R(:,y));
    end
  end
end

ISM = ISM+ISM';
[S,IX] = sort(ISM,2,'descend');
```

As we can see in the above source code example, our program initially loads the previously normalized rating matrix, and then creates another empty matrix to cache all

item similarity values ($ISM_{numItems \times numItems}$). Generally, similarities are calculated based on the well-known cosine function, which computes the included angle between both examined item vectors ($\angle(x, y) = cosineSim(x, y) = \frac{vectorX \cdot vectorY}{|vectorX| \cdot |vectorY|}$). Due to the fact that the item similarity matrix is symmetric, our own implementation merely calculates the upper triangular part, which includes all entries above the main diagonal. To complete all entries, we simply need to add the transposed similarity matrix to our previously computed values (see Line 16).
Furthermore, all rows of the complete matrix are sorted according their similarity values, because our own implementation only considers the top-*n* nearest neighbors (items) for rating prediction (see Line 17; where *IX* contains the indices of the similarity values in descend order). Which neighborhood size produces the most accurate prediction results need to be determined through testing (also refer to Chapter 6). The following *MATLAB* source code illustrates the estimation of an example user-item pair based on the previously computed similarity matrix.

```
1  function r = predictRatings(nbh)
2  load IO_File R ISM IX userU itemI;
3  sum1 = 0;
4  sum2 = 0;
5
6  for nb = 1:nbh
7    itemJ = IX(itemI,nb);
8    if itemI~=ItemJ & R(userU,itemJ)~=0
9      itemSim = ISM(itemI,itemJ);
10     sum1 = sum1 + (itemSim * R(userU,itemJ));
11     sum2 = sum2 + abs(itemSim);
12   end
13 end
14
15 pred(userU,itemI) = (sum1 / sum2);
```

In the above shown implementation the neighborhood size is determined through the parameter *nbh* received at function call. Depending on the number of neighbors our program considers more or less adjacent items for rating prediction (see Line 6). For each top-*n* nearest neighborhood item, our implementation first looks up the corresponding *ISM* matrix index, and then utilize all referenced similarity values to estimate the rating of the currently examined item ($sim_{I,XY} = ISM(item_I, IX(item_I, nb_{XY}))$).
Due to the fact that all entries within our original rating matrix were normalized, it is possible that item similarity values become negative, which in turn distorts our final prediction results. Hence, our implementation always computes the the absolute

value of the similarity values in the denominator ($pred(u,i) = \frac{\sum_{j \in nbh(i)} ISM(i,j) \cdot R(u,j)}{\sum_{j \in nbh(i)} |ISM(i,j)|}$). As might be expected all estimated ratings also need to be de-normalized. Although this section merely discussed the item-oriented nearest neighborhood approach, all introduced techniques are valid for the user-oriented strategy as well. However, the dimension of an appropriate user similarity matrix ($USM_{numUsers \times numUsers}$) would be different from the *ISM* discussed before.

5.2.6 NNH with Content Features

Although this approach employs the previously discussed standard *NNH*-algorithm as the underlying recommendation technology, it is special in that both rating information and content features are utilized for rating prediction. Our research mainly focus on this hybrid approach, because it is expected to improve prediction accuracy. This assumption is rooted in the additional information content given through the extracted user or rather item features (refer to Section 2.3 and 3.4). The following source code example illustrates how to attach an item feature to our original rating matrix.

```
1 function r = calcItemSimMatrix(w)
2 load IO_File R IF;
3
4 RFW = [R;IF*w1];
5
6 ISM(x,y) = cosineSim(RFW(:,x),RFW(:,y));
```

Due to the fact that all movie items are arranged along the x-axis of our rating matrix, all additional item features are attached to the bottom ($RIF = [R; IF]$; see Figure 4.6). On the contrary, user features would be append to the right side of our original rating matrix ($RUF = [R, UF]$).
In the next step our extended model is utilized to calculate the entries of our item or rather user similarity matrix, which is then employed to predict unknown ratings (refer to Section 5.2.5). As before, similarities are computed by comparing the according item/user vectors by virtue of cosine-similarity (see Line 6). But, the main problem concerning this hybrid approach is the increased computational complexity. Generally speaking, the longer the item/user vectors the higher the computational effort. Hence, we are going to apply *SVD* to our extended model (refer to Section 5.2.8).

5.2.7 NNH with SVD

Both the *NNH* and the *SVD* approach are commonly used strategies for rating estimation. However, we intend to combine both techniques within one single approach. Whereas the robust nearest neighborhood algorithm is employed as underlying recommender, and singular value decomposition is used to reduce computational complexity. The following source code extract shows our combined implementation.

```
1  function r = factorizeMatrix(k)
2  load IO_File R;
3
4  [U,S,V] = svds(R,k);
5  SV = sqrt(S)*V';
6
7  numItems = numel(R(1,:));
8  ISM = zeros(numItems,numItems);
9
10 for x = 1:numItems
11   for y = 1:numItems
12     if x <= y
13       break;
14     else
15       ISM(x,y) = cosineSim(SV(:,x),SV(:,y));
16     end
17   end
18 end
19
20 ISM = ISM+ISM';
21 [S,IX] = sort(ISM,2,'descend');
```

In fact, the above *MATLAB* program looks similar to our pure *NNH* approach. But instead of employing the item vectors to compute the similarity values, our current implementation rather utilize the appropriate singular vectors. Same as in our previous *SVD* approach, the singular vectors are obtained by matrix factorization ($\tilde{R}_k = U_{m\times k} \cdot S_{k\times k} \cdot V'_{n\times k}$; see Line 4). Again, the factorization output is joined into an unified compound matrix, which describes the latent item concepts ($Y = S_k^{1/2} V'_k$). The generated compound matrix is then used to compute the single entries of our similarity matrix, by simply comparing the appropriate singular vectors via cosine-similarity ($ISM(i,j) = \frac{columnY(i) \cdot columnY(j)}{\|columnY(i)\| \cdot \|columnY(j)\|}$). After our item similarity matrix was filled up and sorted (see Line 20 and 21), it can be employed to estimate the unknown values within our original rating matrix. As before, missing ratings are predicted by calculating the weighted sum of all neighborhood items (refer to Section 4.5). To be sure, estimated values need to be de-normalized, because all entries within our original

rating matrix were regularized before. Note that all concepts described in the present item-oriented approach are furthermore valid for user-oriented strategies.

5.2.8 NNH with SVD and Content Features

This section deals with our actual hybrid recommendation system, which uses rating information as well as additional content features for prediction. As described earlier, the content features are attached to our original rating matrix (refer to Section 4.2.4). Before our recommendation system can utilize the extended matrix, both ratings and features values need to be regularized via subtractive or rather multiplicative normalization. Due to the fact that the newly generated rating-feature matrix is considerably bigger in size, we additionally apply matrix factorization to reduce the dimension and to decrease computational effort. In the following example implementation the original rating matrix R is inflated with both item features (IF) and user features (UF).

```
1  function r = factorizeMatrix(w1,w2)
2  load IO_File R IF1 IF2 IF3 UF1 UF2 UF3 k;
3
4  IFW = [IF1;IF2;IF3]*w1;
5  UFW = [UF1,UF2,UF3]*w2;
6
7  dim1 = numel(IFW(:,1));
8  dim2 = numel(UFW(1,:));
9  Z = zeros(dim1,dim2);
10
11 RFW = [R,UFW;IFW,Z];
12 [U,S,V] = svds(RFW,k);
13
14 X = U*sqrt(S);
15 Y = sqrt(S)*V';
```

The above *MATLAB* program receives two parameters at function call, whereas the first one represents the item feature weight and the second one stands for the user feature weight. These weights affect the influence of the appropriate content features on the final rating prediction. Our aim is to determine the optimal weight configuration for each single feature as well as for suitable feature combinations.
In the above example implementation multiple user and item features are joined together and processed simultaneously (see Line 4 and 5). Depending on the dimension of the joined user and item matrix, our program creates an adequate *zero-matrix* (Z) to fill up the lower right part of our extended matrix ($RFW = [R, UFW; IFW, Z]$, furthermore

refer to Figure 4.6). Same as in the previously introduce approach *(NNH with SVD)*, the extended matrix is factorized with the aid of singular value decomposition in the next step (see Line 12). Again, the resulting factors are employed to build up the user or rather item compound matrix, which are then utilized by the previously introduced nearest neighborhood algorithm.
Due to the fact that the above approach exploits user and item features along with collaborative filtering techniques, we need to implement both user-oriented and item-oriented *NNH*-algorithm. That means for each unknown rating our program produces two separate prediction results, which are combined to one single estimation value. The main problem concerning this method is that we need to determine optimal ratio between user and item predictions. In the following source code extract we illustrate how to combine those ratings.

```
1  function r = combineRatings(ratio)
2  load IO_File T predItemOriented predUserOriented;
3
4  numTestElems = numel(T(:,1));
5  predCombined = zeros(numTestElems,1);
6
7  for e = 1:numTestElems
8    predCombined(e) = predItemOriented(e)*(ratio) + predUserOriented(e)*(1-ratio);
9  end
10
11 observedValues = T(:,3);
12 r = rmse(observedValues,predCombined);
```

Generally, user-oriented and item-oriented prediction values are combined in a way that the sum of their factors add up to one ($ratio + (1 - ratio) = 1$; see Line 8). The combined prediction values are then compared with the actual observed values of the test dataset T. A typical measure for the accuracy of recommender systems is the *RMSE (Root Mean Square Error)*, which returns an error value that describes the deviation of our model from the actual data (see Line 12; furthermore refer to Chapter 6).

5.3 Latent Feature Relations

Originally we didn't plan to analyze latent feature relations, but the results we found by accident were highly interesting. This approach is similar to our *SVD with Content Features* strategy (refer to Section 5.2.4), where the original rating matrix is extended with user and item features. As before, the resulting extended matrix is then factorized by singular value decomposition. Contrary to our previous approaches, we do not want to predict unknown values within our original rating matrix, but rather we purpose to estimate the lower right part of our extended matrix (Z; see Figure 4.6). The detailed implementation of our novel approach to feature relation retrieval is illustrated in the following source code extract.

```
1  load IO_File R IF UF k;
2
3  numUsers = numel(R(:,1));
4  numItems = numel(R(1,:));
5
6  numItemFeats = numel(IF(:,1));
7  numUserFeats = numel(UF(1,:));
8  Z = zeros(numItemFeats,numUserFeats);
9
10 RF = [R,UF;IF,Z];
11 [U,S,V] = svds(RF,k);
12
13 US = U*sqrt(S);
14 SV = sqrt(S)*V';
15
16 for x = 1:numItemFeats
17   for y = 1:numUserFeats
18     Z(x,y) =dot(US(x+numUsers,:),SV(:,y+numItems));
19   end
20 end
21
22 contourf(Z);
```

As we can see in the above *MATLAB* program, the entries of our sub-matrix Z are calculated by multiplying the appropriate vectors of our pre-computed item and user concept matrices ($Z(x,y) = US(x + userOffset) \cdot SV(y + itemOffset)$; also see Line 18). For the purpose of visualizing all latent feature relations, our implementation constitutes a contour plot of the information contained in our originated matrix Z. The meaning of these contour plots will be given in Chapter 6.

Chapter 6

Evaluation

6.1 Performance Measure

There are many different performance measures that can be found in literature or rather research papers. Usually various measures are employed to evaluate one and the same system, because different quality properties need to be analyzed. Due to the fact that performance measures are often quite complex and hard to interpret, they constitute an own research area. In the following we shortly introduce the most common measures used to evaluate recommendation systems like ours.

Two of the most popular performance quantifier are *RMSE* and *MAE*, which both measure how close the computed estimates are to the values actually observed. In our case estimates are the outcomes of our prediction algorithm, and actual values are given through our test dataset. For our further investigations we decided to employ the *RMSE*, because it has better statistical properties for the distributions that seem to be at work in movie ratings (according to the *Netflix* community). The RMSE is a quadratic scoring rule, which is most useful when large errors are particularly undesirable. Mathematically this can be formulated as $rmse(O, P) = \sqrt{\frac{\sum_{i=1}^{n}(O_i - P_i)^2}{n}}$; whereas O and P are vectors of observed and predicted values respectively.

Recommendation systems that suggest the top most relevant items to a user are often evaluated by dint of *F-Measure*, which is a harmonic mean of precision and recall used in the field of information retrieval ($F = \frac{2 \cdot precision \cdot recall}{precision + recall}$). However, our intention is to improve the prediction accuracy for the ratings estimated by our proposed hybrid recommender.

6.2 Rating Prediction

6.2.1 Rating Interpolation (Regression)

This approach is probably the most straightforward among all investigated rating prediction methods. Based on the item and user average values as well as overall average value of our original rating matrix $(\overline{r}_i, \overline{r}_u, \overline{r})$, we are able to establish a system of equations, which is subsequently solved by linear regression (refer to Section 4.3.1 or Figure 4.7 respectively). The learned parameter configuration (α, β, γ) is then employed to compute unknown ratings of our original user-item matrix ($\tilde{r}_{u,i} = \alpha\overline{r} + \beta\overline{r}_u + \gamma\overline{r}_i$). Depending on the dataset examined, the optimal parameter setting might vary. For our particular *MovieLens* corpus, we determined the following parameter configuration:

$$\begin{aligned} \alpha &= 0.8144 \\ \beta &= 0.8700 \\ \gamma &= -0.6845 \end{aligned}$$

Since all of our experiments investigate one and the same *MovieLens-100K* dataset, we always make use of the above parameter setting for normalization or de-normalization respectively. Note that the regularization of the item/user feature matrices is handled separately.

6.2.2 Singular Value Decomposition (SVD)

As described earlier (refer to Section 2.4 and 4.4), the *SVD* approach approximates the missing rating values based on the matrix factorization ($\hat{r}_{u,i} = (U_k S_k^{1/2})_u \cdot (S_k^{1/2} V_k')_i$). The precision of the estimated ratings is strongly dependent on the dimension of the decomposed matrices. Very short singular vectors might not have enough explanatory power to differentiate the appropriate item or rather users. In the contrary, relatively long singular vectors might lead to over-fitting. This specific circumstance is presented in the following diagram, where we analyze the prediction accuracy of our recommendation system regarding to the dimension k for factorization.

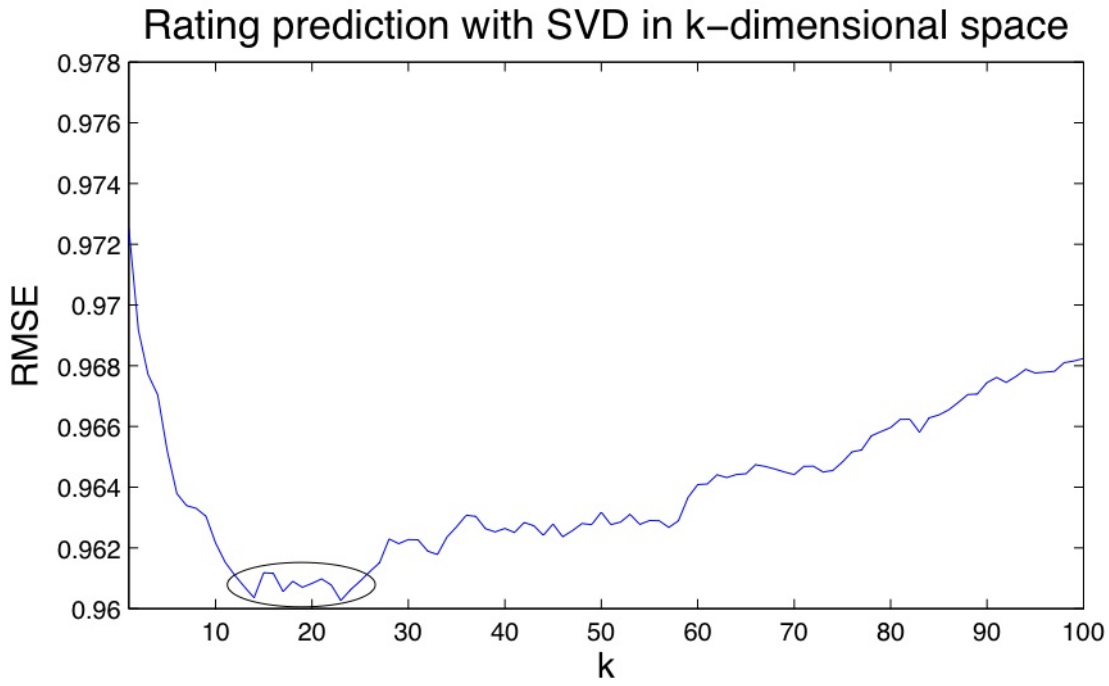


Figure 6.1: SVD Approach with Variable Dimension k

Although the above curve exhibits slight fluctuations, we can observe a general tendency. As expected, a medium k around 10 to 25 results in the highest prediction accuracy. For further research into our *SVD* approach we decided to fix the dimension k to 20, which is the mean of the global minimum area marked in the Figure 6.1.
Besides evaluating the dimension for matrix decomposition, we furthermore analyzed the influence of various item/user content features on the prediction accuracy of our recommender system (refer to Section 5.2.4). The contribution of the single content features is regulated by weights, which are multiplied with the appropriate feature matrices. Multiplying a feature matrix with an ascertained weight value, increases all matrix entries by a multiple of the specified factor. This in turn increases the influence of the feature values as comparing the item/user vectors via cosine similarity. Needless to say, if the weights are set too high, the content features outweigh the rating information. Our aim is to determine the optimal balance between both content and rating information, so that our recommendation system can achieve the highest possible prediction accuracy. Depending on the particular feature or respective feature combination, the optimal weight setting might be different.
Figure 6.3 illustrates the influence of the examined item/user content feature on the system performance, whereas the feature weights are variable.

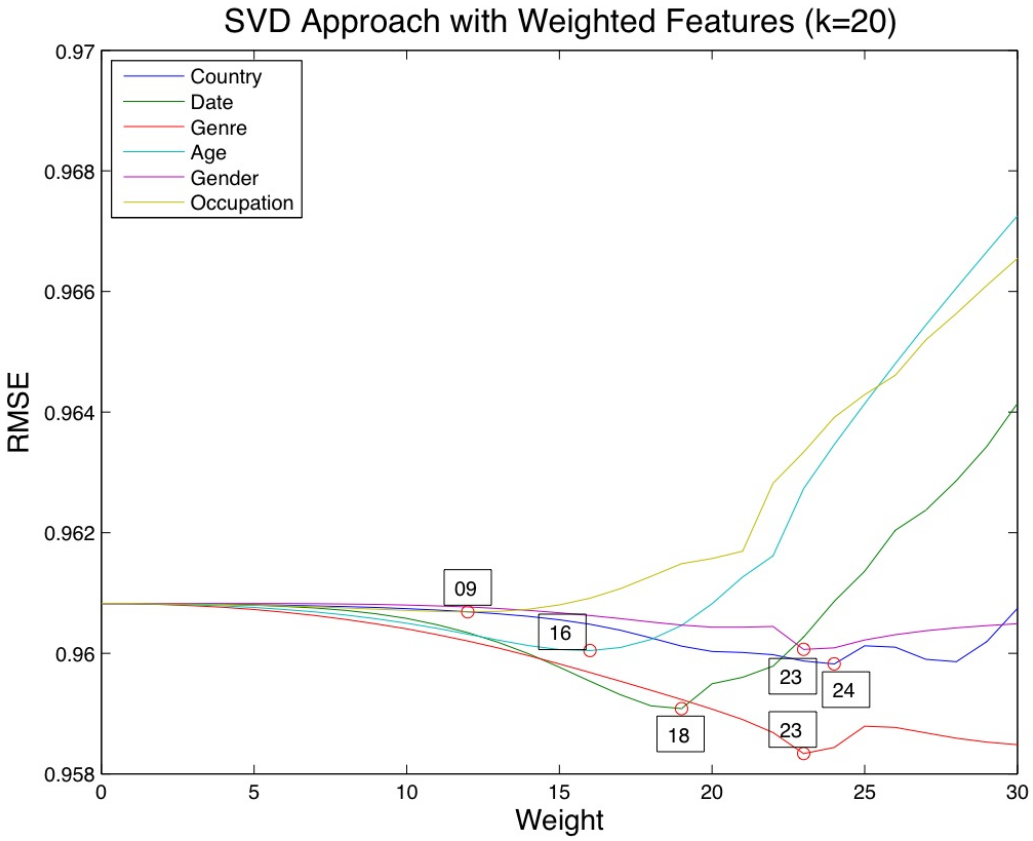


Figure 6.2: SVD Approach with Weighted Item/User Content-Features

Basically, all graphs in Figure 6.2 exhibit the same tendency. By stepwise increasing the feature weights, the prediction error (*RMSE*) of our recommender system gradually falls down until reaching the global minimum, and then continuously climbs up again (Figure 6.2 merely illustrates an extract of the entire analysed range). This characteristic gets even more prominent when different content features are processed simultaneously. Note that the minimum of the single feature graphs is indicated by a red circle and the appropriate weight label.

Furthermore, we found out that the examined item features show an better overall performance than the considered user features. In other words, the accuracy improvement made by the augmentation of item features was noticeable higher. However, the best prediction result could be reached by the combination of all item and user features. A comparison of all observed item/user features can be found in Figure 6.3. Note that the illustrated bars always represent the best weight configuration found for the appropriate features. The according item and user feature weights are displayed inside of the respective bars. In order to emphasize the foremost feature variant, we additionally plotted a horizontal bottom-line.

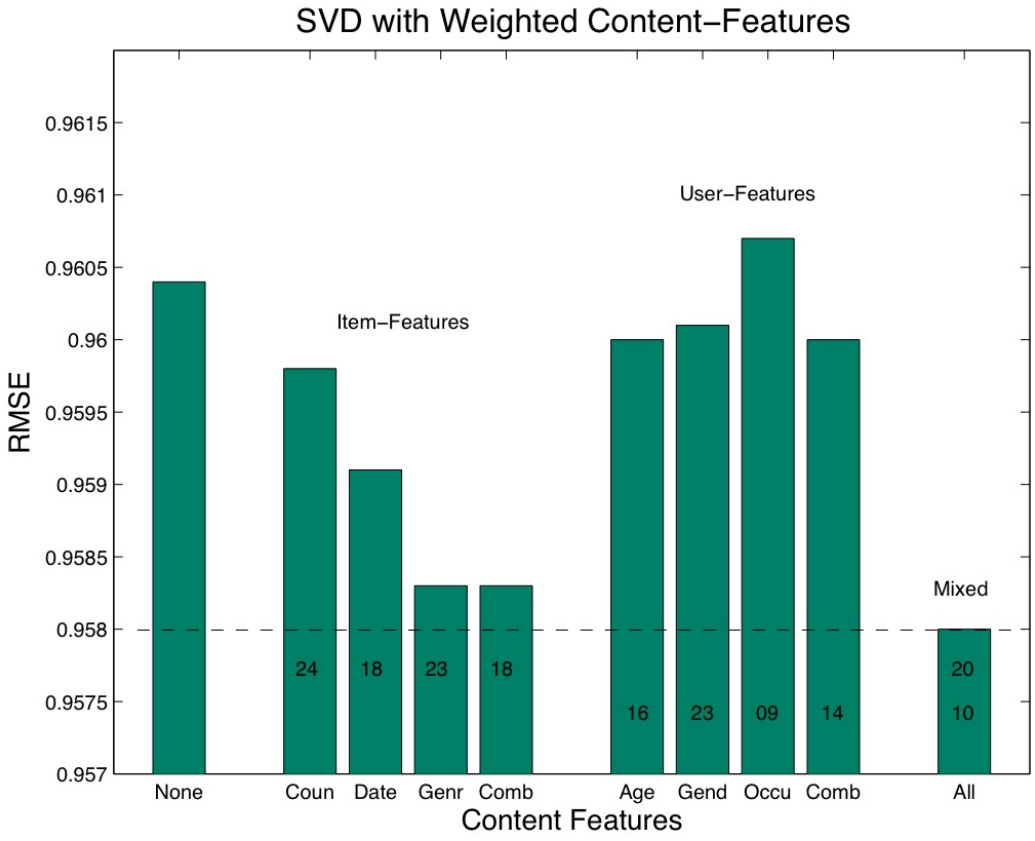


Figure 6.3: SVD Approach in Combination with Item/User Features

6.2.3 Nearest Neighborhood Algorithm (NNH)

In general, there exist two basic implementation variants of the *NNH*-algorithm, namely *item-oriented* and *user-oriented* approach (refer to Section 4.5). Whereas *item-oriented* methods utilize similar items to predict unknown ratings, and *user-oriented* techniques consider like-minded users. However, both collaborative techniques employ the same underlying mathematical approach, in which the current examined item/user rating is estimated based on the *n*-nearest neighbors. As might be expected, the predicted result is strongly dependent on the number of neighbors taken into account. In case of a relatively small neighborhood the estimated ratings might be imprecise, because the considered adjacent items/users do not contain sufficient information to make reliable predictions. Contrariwise, if the specified neighborhood is comparatively big, then it might happen that too many items/users with very low similarities are considered for rating estimation. The actual influence of the neighborhood size (nbh) on the prediction

accuracy of our recommendation system is illustrated in Figure 6.4.

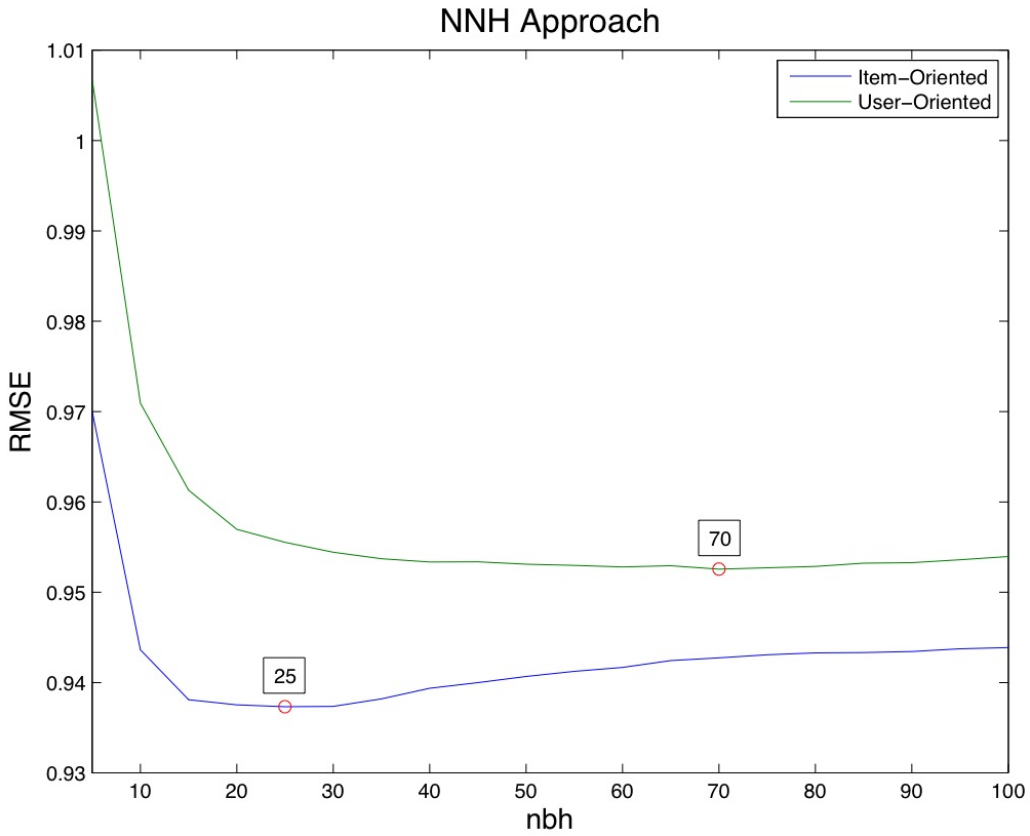


Figure 6.4: Item-Oriented and User-Oriented NNH Approach

The above diagram reveals that our *item-oriented* approach performs much better than the *user-oriented* one. This observation is conform to the general statement about *CF* systems found in most literature or rather research papers [13, 26, 27]. Maybe it can be explained by the fact that individuals are more familiar with items previously preferred than with potential like-minded users.

Furthermore, we can tell that the optimal neighborhood size determined for our *item-oriented* and *user-oriented* approach are quite different. In Figure 6.4 the optimum of both graphs is marked by a red circle and the respective neighborhood size ($I_{min} = 25$ and $U_{min} = 70$). Usually the optimal neighborhood size strongly depends on the investigated dataset. Be that as it may, the more neighbors items/users are examined the higher the computation effort of our rating prediction algorithm.

In the following we furthermore want to analyze the effect of all previously selected item and user features on our *NNH* approach. As described earlier, the content-features are first assigned with weights and then are attached to the initial item or rather user

vectors (see Figure 4.6). The optimal weight setting for the single features or feature combinations was determined by a series of experiments. Figure 6.5 illustrates the performance of our recommendation system with different content-features and variable weights. Again, the optimum of each graph is labeled by the according weight value.

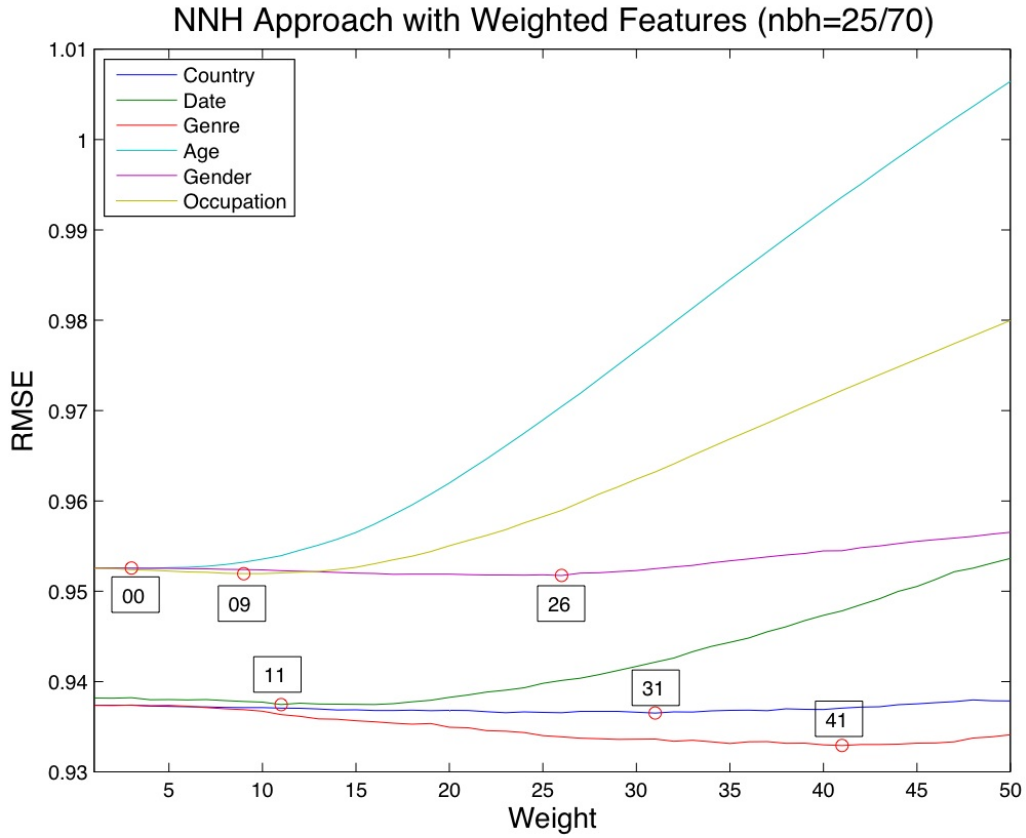


Figure 6.5: NNH Approach with Weighted Item/User Content-Features

As we can see in Figure 6.5, the optimum weight setting varies for each of the examined content-features. In general we can say that relatively high feature weights have a negative influence on the prediction accuracy, because large feature values tend to outbalance the original rating information. This characteristic is clearly observable for the *Movie-Date* as well as for the *User-Age* and *User-Occupation* feature. On the other hand, comparatively small feature weights might not produce any noticeable change for the prediction accuracy of our recommender.

Depending on the type of features processed, our recommendation system needs to switch between item-oriented and user-oriented prediction, or even has to employ both techniques simultaneously (*mixed* approach). The following diagram compares the prediction accuracy achieved by the different features or rather feature combinations.

As before, the illustrated bars represent the best weight setting determined for the appropriate features. The respective item and/or user weights are displayed inside the single bars of Figure 6.6. In order to emphasize the foremost feature variant, we additionally plotted a horizontal bottom-line.

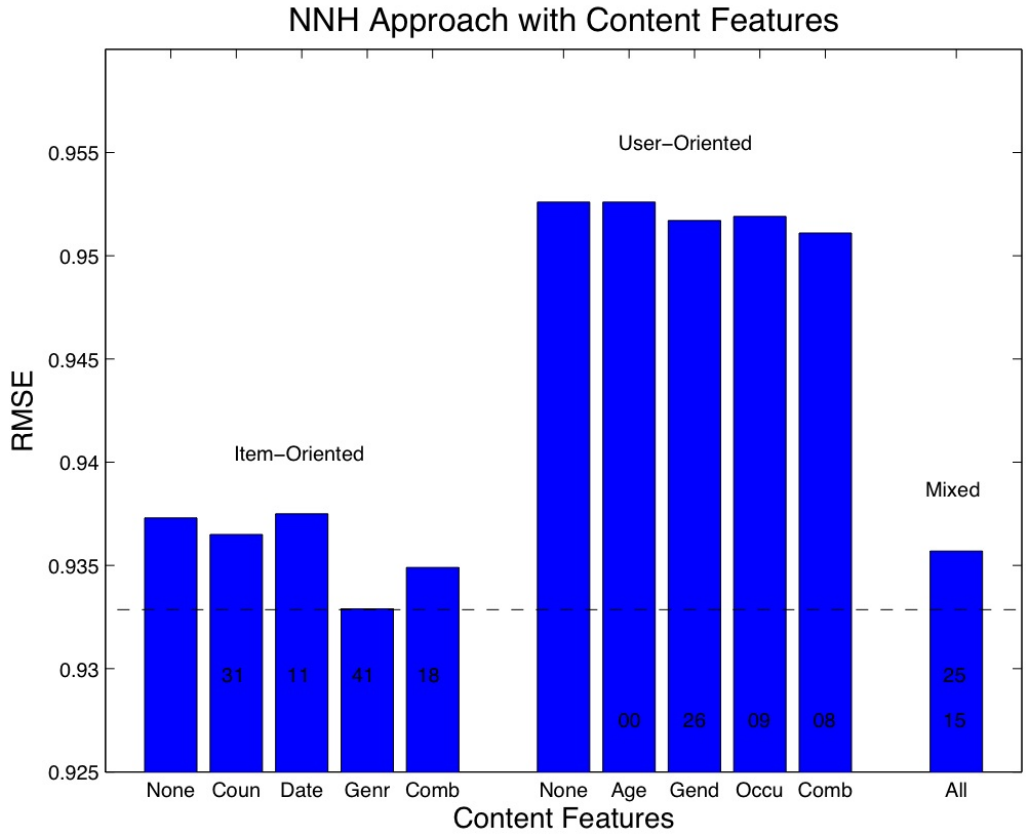


Figure 6.6: NNH Approach in Combination with Item/User Features

Once more, the examined item features performed significantly better than the considered user features. In particular the *Movie-Genre* feature seems to be beneficial to our *NNH*-approach. Surprisingly, the *mixed* feature approach is inferior to our pure item-oriented *NNH* algorithm as well. As likely as not this is due to the negative influence of the considered user content features have on the mixed model.
The main drawback concerning our feature augmented *NNH* approach is the increased computational effort. This is due to the fact that the original rating matrix vectors were extended with additional feature information, which in turn causes more operations for each item or rather user comparison. In the next section we will tackle this problem by combining our above hybrid recommendation system with the previously described matrix decomposition method.

6.2.4 NNH with SVD

Among all investigated rating prediction techniques our *NNH-SVD* approach is the most sophisticated one. In this approach we reduce the dimension of the rating/feature information by means of singular value decomposition before analyzing the data with the nearest neighbor algorithm (refer to Section 5.2.7 and 5.2.8). According to the underlying *NNH* variant, the optimal dimension for the matrix factorization can be quite different. In the following diagram we scrutinize the properties of the *item-oriented* and *user-oriented*, as well as *mixed NNH* implementation.

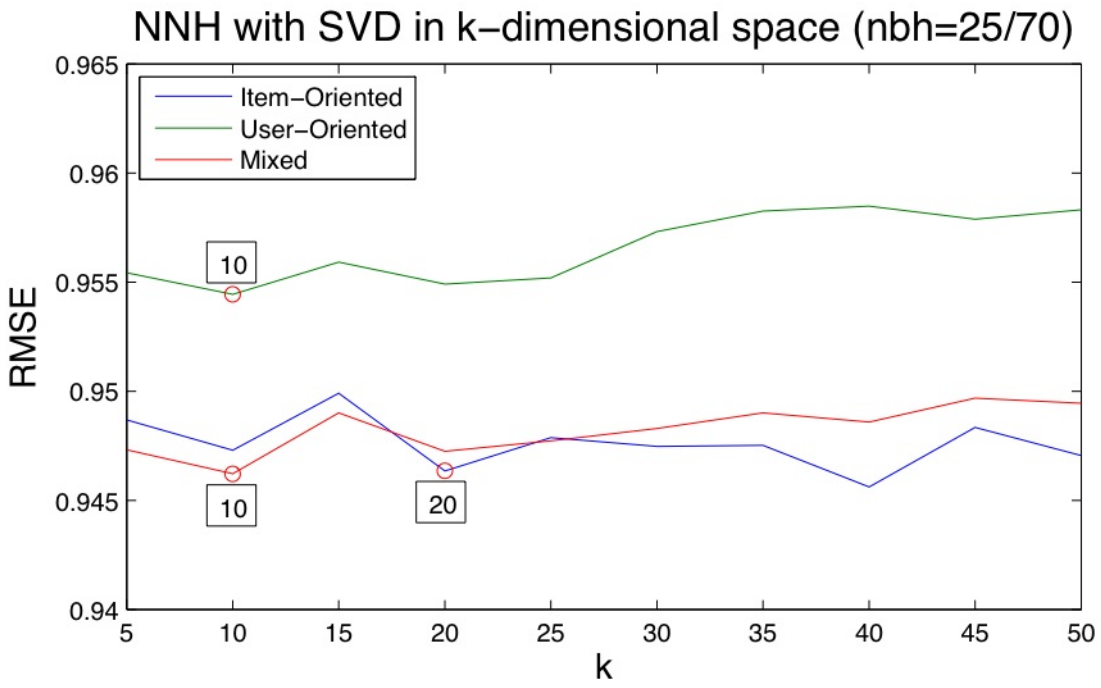


Figure 6.7: NNH Approach with SVD and Variable Dimension k

Due to the fact that the performance of the different implementation variants were analyzed in a certain interval ($k_{n+1} = k_n + 5$), the graphs in Figure 6.7 appear relatively unsteady. Nonetheless, we can identify the global optima or rather minima for each of our examined prediction methods. As we can see, a dimension of $k = 20$ is the optimal adjustment for the *item-oriented* implementation, and a dimension of $k = 10$ is the most favorable setting for the *user-oriented* as well as *mixed* approach.
In the following we furthermore want to investigate the effect of our previously selected content-features on the introduced *NNH-SVD* recommendation system. As before, the contribution of the single item and user features is regulated by weights. Figure 6.8

illustrates the prediction accuracy of our hybrid recommender with weighted features, based on the optimal neighborhood and dimension determined before.

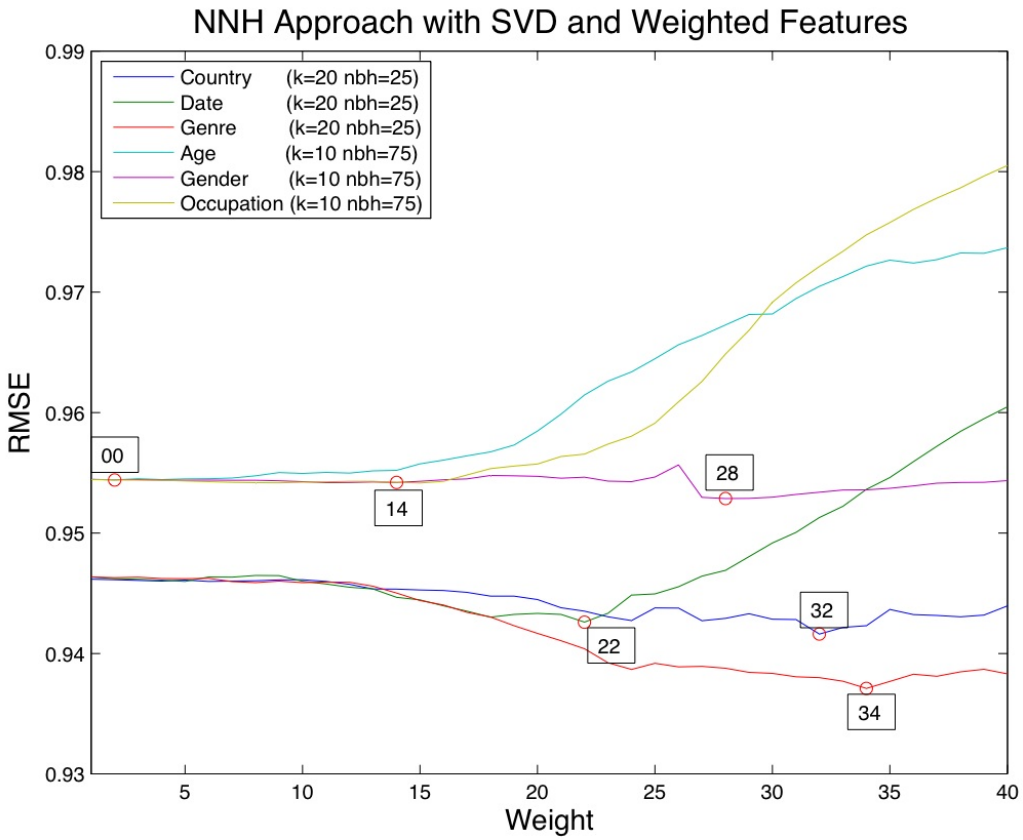


Figure 6.8: NNH Approach with SVD and Weighted Item/User Features

For each graph shown in Figure 6.8, the minimal reached prediction error is labeled by the according weight value. Just like in the pure *NNH* approach, the item-oriented implementation performed significantly better than the user-oriented one. Moreover, it is apparent that the performance improvement achieved by the weighted item features is much higher. Again, the *Movie-Genre* feature produced surpassing prediction results. However, the *User-Age* feature could not benefit our *NNH-SVD* approach, and therefore was not considered for the *mixed* feature variant. All different single feature and mixed feature variants of our *NNH-SVD* approach are compared in Figure 6.9, whereas the determined feature weights are displayed in the respective chart bars. For both *item-oriented* and *user-oriented* implementation, the combined feature variant performed best. Nevertheless, the *mixed* approach with merged item and user features, marked by the horizontal bottom-line, outperformed all other variants.

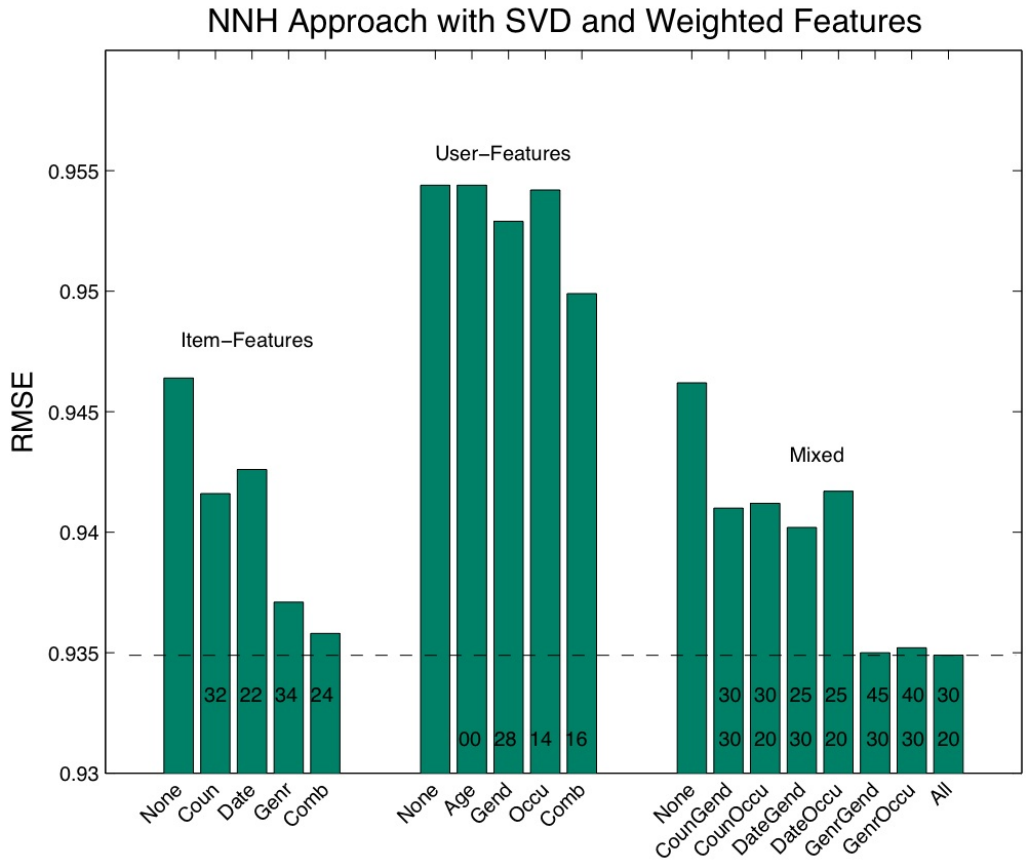


Figure 6.9: NNH-SVD Approach in Combination with Item/User Features

6.2.5 Computational Complexity

In many practical recommendation systems unknown user-item ratings necessarily need to be computed in realtime. Due to this fact, we want to analyze the computational effort of all previously discussed recommendation approaches. Figure 6.10 compares the different rating prediction variants according their system runtime, whereas time is represented on a logarithmic scale.

Even though it is not noticeable at the first sight, the computational effort of the *NNH-SVD* approach is about 500 times higher than for the pure *SVD* variant. Furthermore, the runtime of the pure *NNH* approach is another $four$ times bigger.

However, the performance and runtime of most present recommendation system are in competition, and have a mutual influence on each other. For this reason we need to treat the system with all its factors as a whole. Our hybrid *NNH-SVD* approach achieves the around the same accuracy than the pure *NNH* algorithm within less time, and therefore is superior. Compared to the pure *SVD* variation, our hybrid NNH-SVD approach has a higher runtime, but produces excellent rating predictions. Summing up, it is always a

balancing act to decide between higher accuracy or better runtime. On that account, we believe that our hybrid *NNH-SVD* approach is the happy medium.

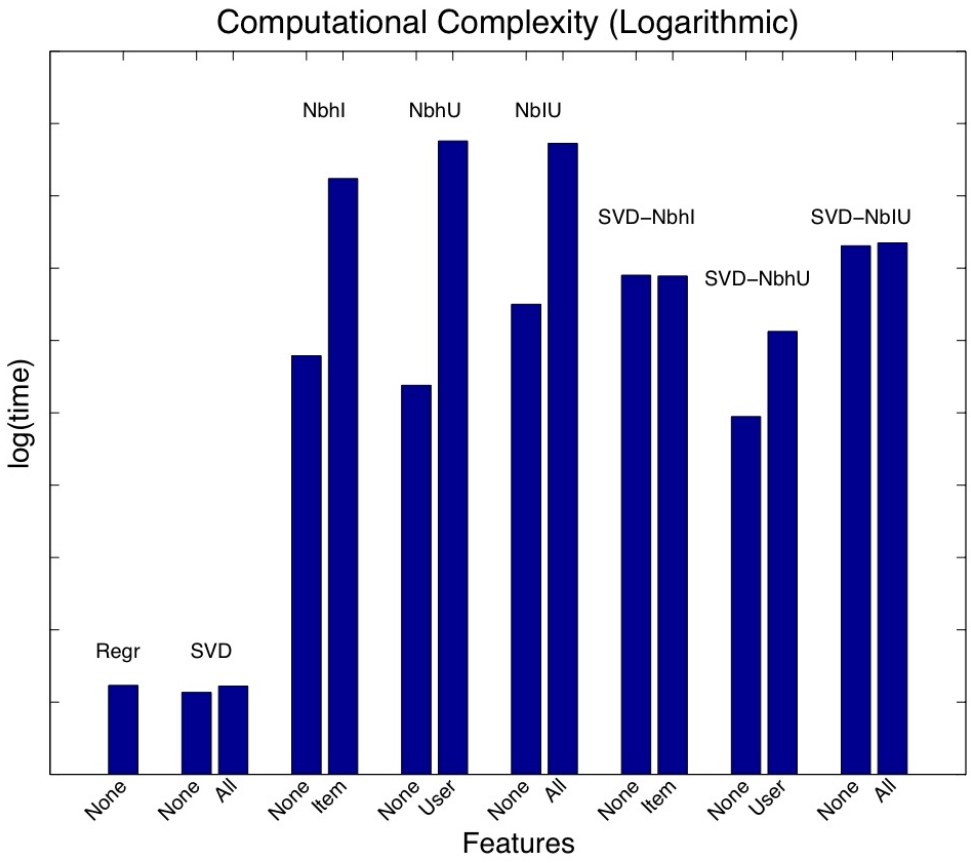


Figure 6.10: Computational Effort of Recommender Systems

6.3 Latent Feature Relations

As mentioned earlier, latent feature relations are not an object of our original research work, but they turned out to be highly interesting. Similar to the pure *SVD* approach, unknown entries of our extended rating-feature matrix are estimated by means of the factorized matrices. However, instead of predicting unknown ratings, we are interested in computing missing feature information (refer to Section 5.3). Those estimated features can help us understand the correlation between different item and user features, and shed light on hidden user preferences or latent item characteristics respectively.
Figure 6.11 reveal the correlation between *Movie-Genre* and *User-Age* feature, whereas red colored areas represent user interest and blue colored areas denote disfavor.

Everything in yellow or green tint express indifference or rather neutrality. Note that the genres in Figure 6.11 were ordered according the biggest eigenvector of the compound matrix UE, by what genres with similar properties are grouped together.

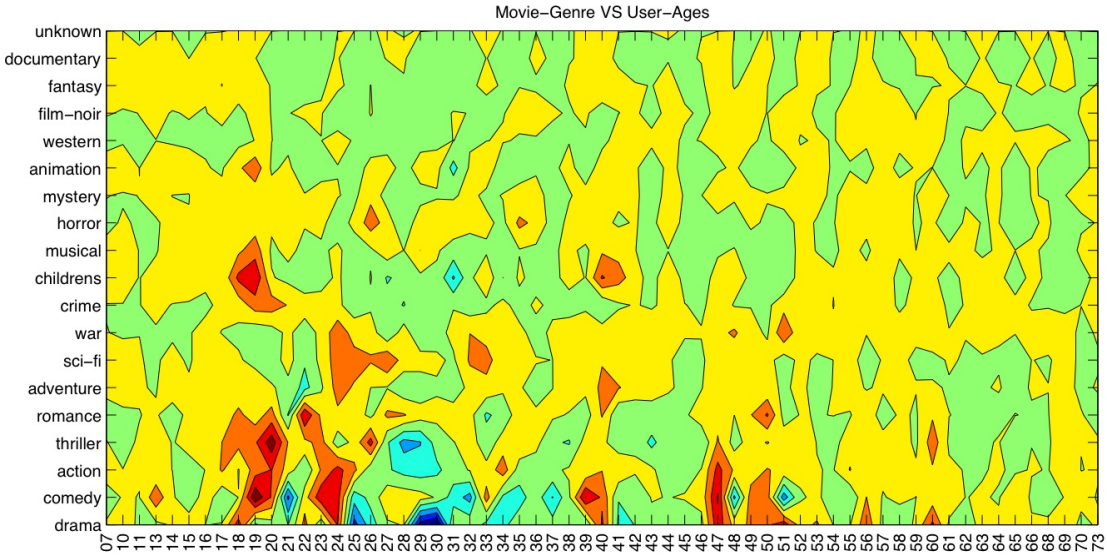


Figure 6.11: Correlation of Movie-Genre and User-Ages

From the above contour plot we can deduce that young people between the age of 18 and 24 have interest in a lot of different movie genres. Especially thrillers, comedy soaps and children's movies are popular among this age group. In contrast, user around 28 to 32 years old are more pessimistic, and express disfavor in various genres. For instance, 29 and 30 year old users obviously dislike dramas. Moreover, we can infer that people above the age of 63 are kind of indifferent to any genre. If we view our above contour plot from another perspective, we can furthermore analyze the characteristics of the particular movies categories. For example, we can deduce that the genres thriller and comedy are quite controversial, because they are either loved or hated by different age groups. Additionally, we can tell that some genres like documentary or fantasy do not enjoy great popularity at all.

In the following we are going to scrutinize the correlation between *Movie-Genre* and *User-Gender*. Figure 6.12 illustrates the contour plot of the appropriate feature values approximated by virtue of our decomposed rating-feature matrix. As before, red colored areas represent user interest and blue colored areas denote disfavor.

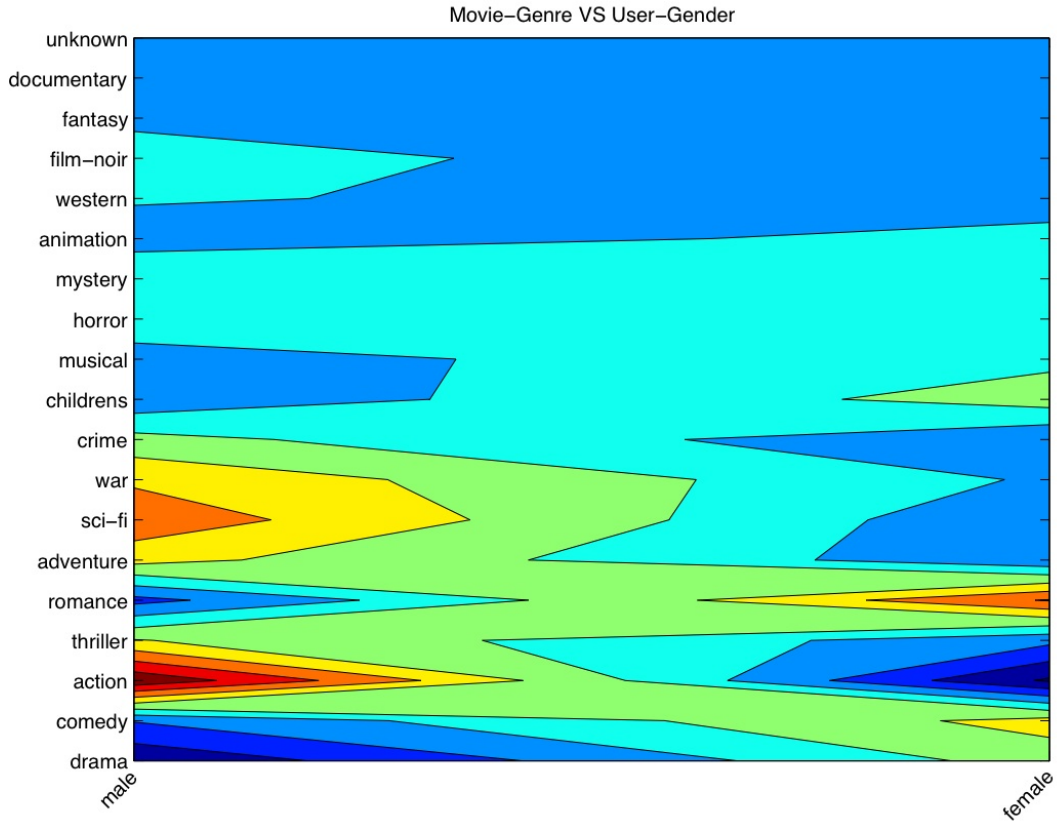


Figure 6.12: Correlation of Movie-Genre and User-Gender

Figure 6.12 does not only show interests and disfavors of males or rather females, but also reveals those genres which provoke conflicts between both genders. For instance, there exists a deep disagreement about the *action* genre. Most men favor action movies, whereas women do not like them at all. On the contrary, *romance* enjoys great popularity among women, but is often refused by men. Moreover, the *science-fiction* and *comedy* genre are controversial, and might bring some marriage to fail. Additionally, we can deduce that some genres like *documentary* or *fantasy* are relatively out-of-favor. Even though most conclusions drawn from Figure 6.12 are common knowledge, this results are based on our novel approach to retrieve latent feature relations. The matter of fact that our measurement results and the real world knowledge are consistent confirms the trustworthiness of our novel approach.

In the end, we furthermore want to investigate the correlation between *Movie-Genre* and *User-Occupation* feature. Figure 6.13 displays the hidden feature information that were estimated by means of our factorized matrices ($\tilde{Z}_{u,i} = UE_u \cdot EV_i$). Note that both genres and occupations were sorted by their respective biggest eigenvector.

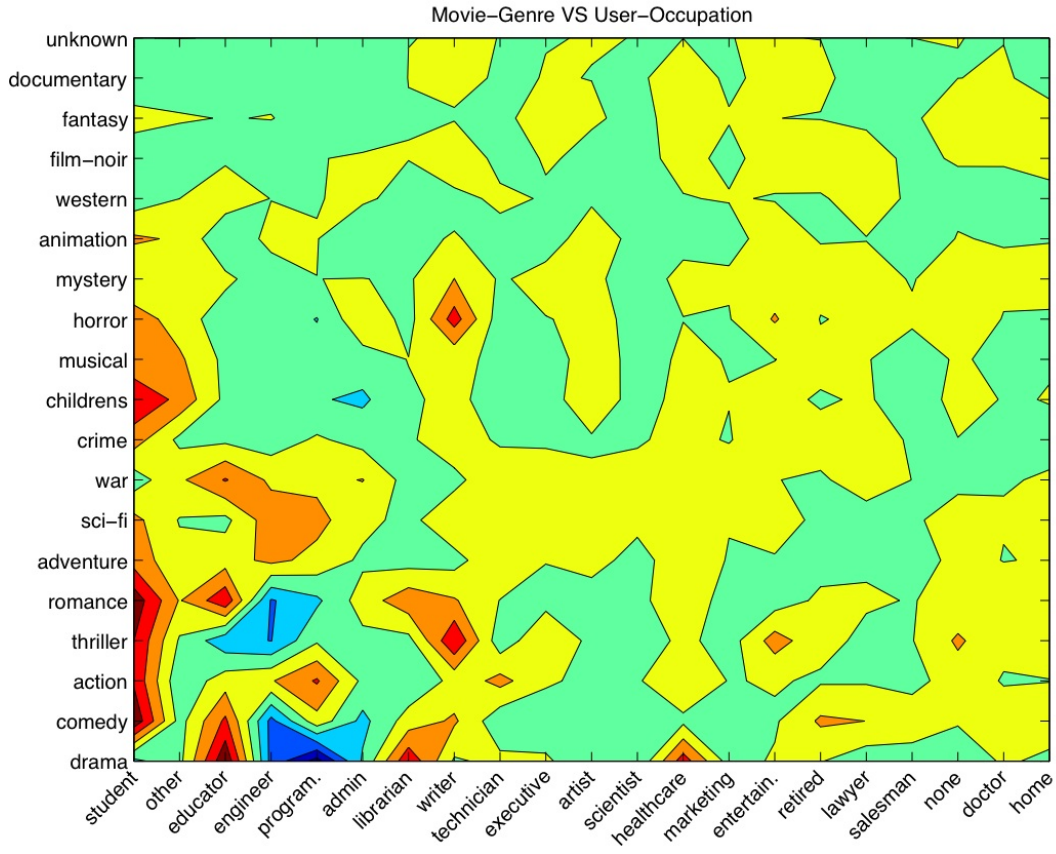


Figure 6.13: Correlation of Movie-Genre and User-Occupation

The information presented in Figure 6.13 is quite complex and allows a diversity of interpretations. First of all, based on the amount of interests and disfavors for each user group, we can deduce what kind of people possess a positive or rather negative attitude. For instance, we can reason that students and educators have a cheerful disposition, because they have interest in movies of many different genres. The other way round, we can infer that engineers and administrators are critical personalities, as they have few interests and many disfavors regarding movie genres. If we view our above contour plot from another perspective (row-wise), we can furthermore conclude controversial as well as indisputable genres. Obviously, *drama* and *romance* are rather controversial, whereas other genres like *action* and *horror* do not exhibit any disagreements among the examined occupations.

Beside the discussed feature combinations, there exist many other latent information which is worth to be retrieved. The additional knowledge about user preferences and item characteristics obtained from the examined feature can help to achieve a deeper understanding of the investigated dataset. In turn, this understanding can be used to design a recommender that produces more individual and accurate predictions.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In our work we proposed a novel approach to rating prediction, where collaborative filtering and content-based filtering techniques were combined. Although there already exist a number of hybrid recommendation systems, our approach is special in that rating and content information are joined to an unified model. The main advantages of this unified model are less parameters and more reasonable prediction results.
Our hybrid recommender was implemented in the *MATLAB* programming language and tested by means of the well-known *MovieLens* rating dataset. All supplementary movie information was retrieved from the *IMDB* online archive. We described the linkage of both data sources, and illustrated the relations between their entities. By means of various experiments, we demonstrated that the extracted content features are beneficial to the prediction accuracy of our hybrid recommendation system.
Due to the multiplicity of content features our model expended to an enormous size, by what the computational effort increased substantially. In order to minimize the system runtime, we reduced the dimension of our hybrid model by means of singular value decomposition. Compared to the basic collaborative filtering algorithm, our dimension reduced hybrid approach performed better in prediction accuracy and runtime. On that account we believe that our novel approach is practical to real-life applications.
In addition, this thesis presents an innovative technique to extract latent feature relations that help to achieve a deeper understanding of user preferences and item characteristics. This knowledge can be employed to design an advanced recommendation system with the ability to generate more individual and accurate prediction results.

7.2 Future Work

Rating Information: For the evaluation of our hybrid approach we employed the *MovieLens-100K* dataset, which contains 100.000 user-item ratings. However, the *GroupLens Research Team* furthermore provides far bigger *MovieLens* data snapshots with 1M or 10M ratings respectively. It would be quite interesting to analyze these two corpora, because a higher number of ratings is expected to give more information about the exact behavior of our hybrid recommender. Moreover, a data volume of about 10M ratings is close to what most present recommendation systems are required to handle, and would be a perfect test for our own implementation.

Content Features: In our research we investigated the item features *Date*, *Country* and *Genre*, as well as the user features *Age*, *Occupation* and *Gender*. These content features were chosen because of their comparatively high information density. However, there might exist many other item and user features that are worth to be examined. We believe that some features with low informational content would be applicable as well, if all items are grouped according their values (e.g. *User-Zipcode*). Moreover, it would be interesting to study another dataset from a different domain, because unlike content attributes seem to cause divergent prediction results.

Feature Relations: Due to the multiplicity of content features extracted from the *IMDB* and *MovieLens* data sources, there exist many possible item-user feature combinations that contain valuable information. In our work we investigated the *Genre-Age*, *Genre-Gender* as well as *Genre-Occupation* item-user relations, which unveil the degree of popularity for the examined genres regarding different user groups. In order to get an even better understanding of the latent item characteristics and user properties, further item-user combinations should be scrutinized. The uncovered item-user specifics could be stored in a separate knowledge base, which assists in making more individual recommendations. We believe that a combination of our hybrid approach with an additional knowledge base would achieve superior prediction results.

Bibliography

[1] GAUCH, SUSAN, MIRCO SPERETTA, ARAVIND CHANDRAMOULI and ALESSANDRO MICARELLI: *User Profiles for Personalized Information Access*. In BRUSILOVSKY, PETER, ALFRED KOBSA and WOLFGANG NEJDL (editors): *The Adaptive Web*, volume 4321 of *Lecture Notes in Computer Science*, pages 54–89. Springer, 2007.

[2] ADOMAVICIUS and TUZHILIN: *Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions*. IEEETKDE: IEEE Transactions on Knowledge and Data Engineering, 17, 2005.

[3] PAZZANI, MICHAEL J. and DANIEL BILLSUS: *Content-Based Recommendation Systems*. In BRUSILOVSKY, PETER, ALFRED KOBSA and WOLFGANG NEJDL (editors): *The Adaptive Web*, volume 4321 of *Lecture Notes in Computer Science*, chapter 10, pages 325–341. Springer-Verlag, Berlin, Germany, May 2007.

[4] SCHAFER, J. BEN, DAN FRANKOWSKI, JON HERLOCKER and SHILAD SEN: *Collaborative Filtering Recommender Systems*. In BRUSILOVSKY, PETER, ALFRED KOBSA and WOLFGANG NEJDL (editors): *The Adaptive Web*, volume 4321 of *Lecture Notes in Computer Science*, chapter 9, pages 291–324. Springer-Verlag, Berlin, Germany, may 2007.

[5] KOREN, YEHUDA: *Tutorial on recent progress in collaborative filtering*. In PU, PEARL, DEREK G. BRIDGE, BAMSHAD MOBASHER and FRANCESCO RICCI (editors): *Proceedings of the 2008 ACM Conference on Recommender Systems, RecSys 2008, Lausanne, Switzerland, October 23-25, 2008*, pages 333–334. ACM, 2008.

[6] *MovieLens Dataset*, USA, 2003. University Minnesota.

[7] IDMB: *IMDB Dataset*, 1990-2009.

[8] KOREN, YEHUDA: *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*, pages 426–434, New York, NY, USA, 2008. ACM.

[9] A. TOESCHER, M. JAHRER, R. LEGENSTEIN: *Improved Neighborhood-Based Algorithms for Large-Scale Recommender Systems*. 2nd Netflix-KDD Workshop, 2008.

[10] BURKE, ROBIN D.: *Hybrid Web Recommender Systems*. In BRUSILOVSKY, PETER, ALFRED KOBSA and WOLFGANG NEJDL (editors): *The Adaptive Web, Methods and Strategies of Web Personalization*, volume 4321 of *Lecture Notes in Computer Science*, pages 377–408. Springer, 2007.

[11] MELVILLE, P., R. MOONEY and R. NAGARAJAN: *Content-boosted collaborative filtering*. In *ACM SIGIR Workshop on Recommender Systems*, September 2001.

[12] LINDEN, GREG, BRENT SMITH and JEREMY YORK: *Amazon.com Recommendations: Item-to-Item Collaborative Filtering*. IEEE Internet Computing, 7(1), 2003.

[13] SARWAR, BADRUL M., GEORGE KARYPIS, JOSEPH A. KONSTAN and JOHN T. RIEDL: *Application of Dimensionality Reduction in Recommender System - A Case Study*. In *In ACM WebKDD Workshop*, 2000.

[14] MICARELLI, ALESSANDRO, FILIPPO SCIARRONE and MAURO MARINILLI: *Web Document Modeling*. In BRUSILOVSKY, PETER, ALFRED KOBSA and WOLFGANG NEJDL (editors): *The Adaptive Web, Methods and Strategies of Web Personalization*, volume 4321 of *Lecture Notes in Computer Science*, pages 155–192. Springer, 2007.

[15] CHEN, LIREN and KATIA P. SYCARA: *WebMate: A Personal Agent for Browsing and Searching*. In *Agents*, pages 132–139, 1998.

[16] STERN, DAVID H., RALF HERBRICH and THORE GRAEPEL: *Matchbox: large scale online bayesian recommendations*. In *WWW '09: Proceedings of the 18th international conference on World wide web*, pages 111–120, New York, NY, USA, 2009. ACM.

[17] MELVILLE, PREM, RAYMOND J. MOONEY and RAMADASS NAGARAJAN: *Content-Boosted Collaborative Filtering for Recommendations*. In *AAAI/IAAI*, pages 187–192, 2002.

[18] SOBOROFF, IAN M. and CHARLES K. NICHOLAS: *Combining Content and Collaboration in Text Filtering*, 1999.

[19] CANNY, JOHN F.: *Collaborative filtering with privacy via factor analysis*. In *SIGIR*, pages 238–245. ACM, 2002.

[20] SHLENS, JONATHON: *A Tutorial on Principal Component Analysis*, December 2005.

[21] WANG, JUN, ARJEN P. DE VRIES and MARCEL J. T. REINDERS: *Unifying user-based and item-based collaborative filtering approaches by similarity fusion*. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 501–508, New York, NY, USA, 2006. ACM Press.

[22] BELL, ROBERT, YEHUDA KOREN and CHRIS VOLINSKY: *Modeling relationships at multiple scales to improve accuracy of large recommender systems*. In *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 95–104, New York, NY, USA, 2007. ACM.

[23] BELL, ROBERT, YEHUDA KOREN and CHRIS VOLINSKY: *Modeling relationships at multiple scales to improve accuracy of large recommender systems*. In BERKHIN, PAVEL, RICH CARUANA and XINDONG WU (editors): *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Jose, California, USA, August 12-15, 2007*, pages 95–104. ACM, 2007.

[24] BELL, ROBERT M., YEHUDA KOREN and CHRIS VOLINSKY: *The BellKor solution to the Netflix Prize*. KDD 2007, ICDM 2007, 2007.

[25] BELL, ROBERT M. and YEHUDA KOREN: *Improved Neighborhood-based Collaborative Filtering*.

[26] SARWAR, BADRUL, GEORGE KARYPIS, JOSEPH KONSTAN and JOHN REIDL: *Item-based collaborative filtering recommendation algorithms*. In *WWW '01: Pro-*

ceedings of the 10th international conference on World Wide Web, pages 285–295, New York, NY, USA, 2001. ACM.

[27] BELL, ROBERT M. and YEHUDA KOREN: *Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights*. In *ICDM '07: Proceedings of the 2007 Seventh IEEE International Conference on Data Mining*, pages 43–52, Washington, DC, USA, 2007. IEEE Computer Society.

[28] LUJAN, M., A. USMAN, P. HARDIE, T. L. FREEMAN and J. R. GURD: *Storage Formats for Sparse Matrices in Java*. Lecture Notes in Computer Science, pages 364–371, 2005.

List of Figures

List of Tables

Abbreviations

ASCII	American Standard Code for Information Interchange
CBF	Content-Based Filtering (text-based prediction method)
CF	Collaborative Filtering (adjacent user/item prediction method)
CPU	Central Processing Unit (electronic circuit)
ERD	Entity Relationship Diagram (abstract representation of data)
IF	Item-Features (additional movie information)
IMDB	Internet Movie Database (online collection of movie information)
IO	Input-Output (communication between hardware or software components)
IR	Information Retrieval (science of searching information)
ISM	Item-Similarity Matrix (internal data structure)
JSA	Java Sparse Array (abstract data structure)
KDD	Knowledge Discovery in Databases (interest group on data mining - kdd.org)
LR	Linear Regression (statistical analysis)
LSI	Latent Semantic Indexing (technique in natural language processing)
MAE	Mean Absolute Error (performance measure)
MTJ	Matrix Toolkit for Java (collection of matrix data structures)
NNH	Nearest Neighborhood Approach
RMSE	Root Mean Squared Error (performance measure)
SVD	Singular Value Decomposition (matrix factorization method)
TF-IDF	Term Frequency - Inverse Document Frequency (statistical measure)
UF	User-Features (additional user information)

Appendix

A Appendix: Source Code

Feature Retrieval - Java Program

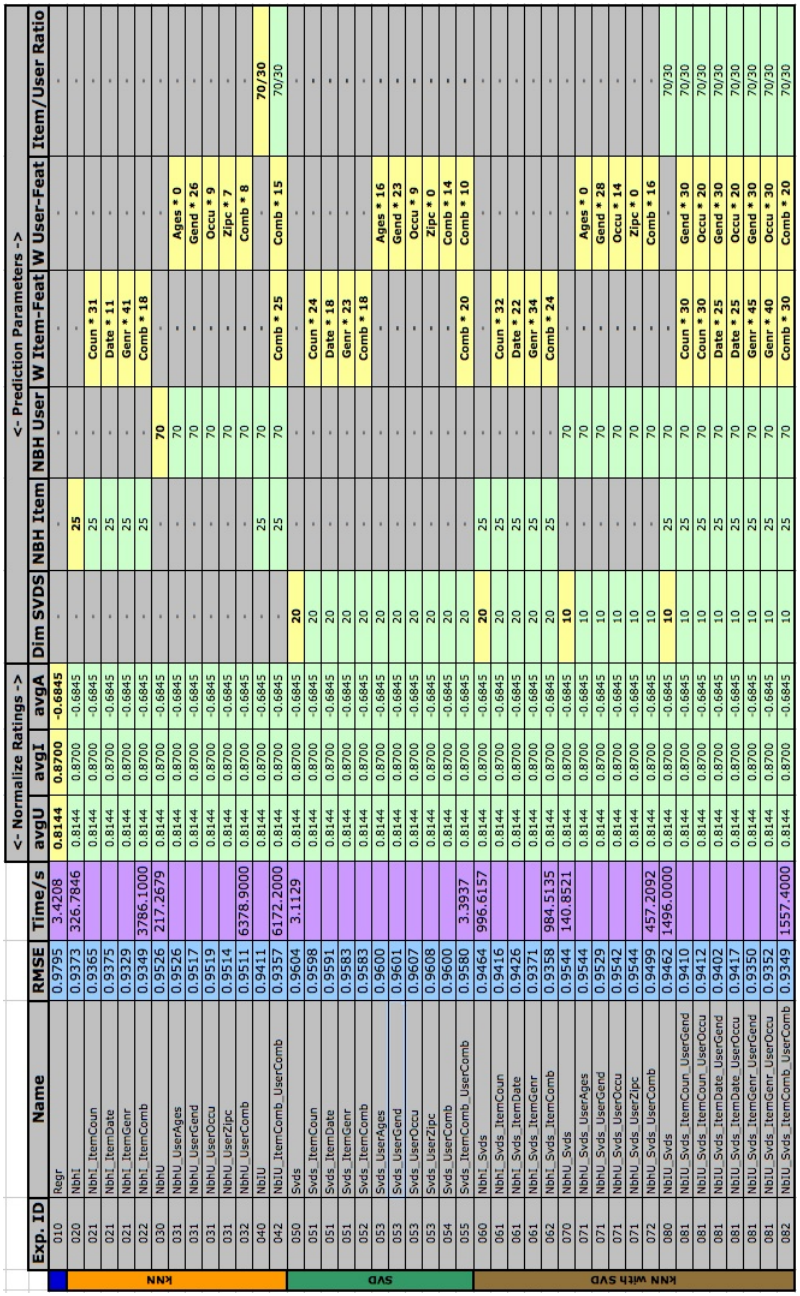
```
1  public class MainItems {
2    public static void main(String[] args) {
3      MovieFeatureMatrix MFM = new MovieCountryMatrix();
4      HashMap<String,Integer> movieTitleID = MFM.parseItemFile("input/u.item");
5      HashMap<String,Integer> featureNameID = MFM.parseFeatureFile("input/countries.
           list");
6      FlexCompRowMatrix M = new FlexCompRowMatrix(featureNameID.values().size(),
           movieTitleID.values().size());
7      M = MFM.parseMovieFeatureFile(M, "input/countries.list");
8      FlexCompRowMatrix MR = MFM.reduceMatrixDimension(M);
9      MFM.writeMatrixToFile(MR,"output/movieCountryMatrix.txt");
10   }
11 }
```

```
1  public abstract class MovieFeatureMatrix {
2    public HashMap<String,Integer> parseItemFile(String fileName) { ... }
3    public abstract HashMap<String,Integer> parseFeatureFile(String fileName);
4    public abstract FlexCompRowMatrix parseMovieFeatureFile(FlexCompRowMatrix M, String
         fileName);
5    public FlexCompRowMatrix reduceMatrixDimension(FlexCompRowMatrix M) { ... }
6    public void writeMatrixToFile(FlexCompRowMatrix MR, String fileName) { ... }
7  }
```

```
1  public class MovieCountryMatrix extends MovieFeatureMatrix {
2    public MovieCountryMatrix() {
3      super();
4    }
5    public HashMap<String,Integer> parseFeatureFile(String fileName) { ... }
6    public FlexCompRowMatrix parseMovieFeatureFile(FlexCompRowMatrix M, String fileName
       ) { ... }
7  }
```

B Appendix: Experimental Results

Comparison of Rating Prediction Approaches



	Exp. ID	Name	RMSE	Time/s	<- Normalize Ratings -> avgU	avgI	avgA	<- Prediction Parameters -> Dim SVDS	NBH Item	NBH User	W Item-Feat	W User-Feat	Item/User Ratio
					0.8144	**0.8700**	**-0.6845**	-	-	-	-	-	-
kNN	010	Regr	0.9795	3.4208	0.8144	0.8700	-0.6845	-	-	-	-	-	-
	020	NbhI	0.9373	326.7846	0.8144	0.8700	-0.6845	-	**25**	-	-	-	-
	021	NbhI_ItemCoun	0.9365		0.8144	0.8700	-0.6845	-	25	-	**Coun * 31**	-	-
	021	NbhI_ItemDate	0.9375		0.8144	0.8700	-0.6845	-	25	-	**Date * 11**	-	-
	021	NbhI_ItemGenr	0.9329		0.8144	0.8700	-0.6845	-	25	-	**Genr * 41**	-	-
	022	NbhI_ItemComb	0.9349	3786.1000	0.8144	0.8700	-0.6845	-	25	-	**Comb * 18**	-	-
	030	NbhU	0.9526	217.2679	0.8144	0.8700	-0.6845	-	-	**70**	-	-	-
	031	NbhU_UserAges	0.9526		0.8144	0.8700	-0.6845	-	-	70	-	**Ages * 0**	-
	031	NbhU_UserGend	0.9517		0.8144	0.8700	-0.6845	-	-	70	-	**Gend * 26**	-
	031	NbhU_UserOccu	0.9519		0.8144	0.8700	-0.6845	-	-	70	-	**Occu * 9**	-
	031	NbhU_UserZipc	0.9514		0.8144	0.8700	-0.6845	-	-	70	-	**Zipc * 7**	-
	032	NbhU_UserComb	0.9511	6378.9000	0.8144	0.8700	-0.6845	-	-	70	-	**Comb * 8**	-
	040	NbIU	0.9411		0.8144	0.8700	-0.6845	-	25	70	-	-	**70/30**
	042	NbIU_ItemComb_UserComb	0.9357	6172.2000	0.8144	0.8700	-0.6845	-	25	70	**Comb * 25**	**Comb * 15**	70/30
SVD	050	Svds	0.9604	3.1129	0.8144	0.8700	-0.6845	**20**	-	-	-	-	-
	051	Svds_ItemCoun	0.9598		0.8144	0.8700	-0.6845	20	-	-	**Coun * 24**	-	-
	051	Svds_ItemDate	0.9591		0.8144	0.8700	-0.6845	20	-	-	**Date * 18**	-	-
	051	Svds_ItemGenr	0.9583		0.8144	0.8700	-0.6845	20	-	-	**Genr * 23**	-	-
	052	Svds_ItemComb	0.9583		0.8144	0.8700	-0.6845	20	-	-	**Comb * 18**	-	-
	053	Svds_UserAges	0.9600		0.8144	0.8700	-0.6845	20	-	-	-	**Ages * 16**	-
	053	Svds_UserGend	0.9601		0.8144	0.8700	-0.6845	20	-	-	-	**Gend * 23**	-
	053	Svds_UserOccu	0.9607		0.8144	0.8700	-0.6845	20	-	-	-	**Occu * 9**	-
	053	Svds_UserZipc	0.9608		0.8144	0.8700	-0.6845	20	-	-	-	**Zipc * 0**	-
	054	Svds_UserComb	0.9600		0.8144	0.8700	-0.6845	20	-	-	-	**Comb * 14**	-
	055	Svds_ItemComb_UserComb	0.9580	3.3937	0.8144	0.8700	-0.6845	20	-	-	**Comb * 20**	**Comb * 10**	-
kNN with SVD	060	Nbhl_Svds	0.9464	996.6157	0.8144	0.8700	-0.6845	**20**	25	-	-	-	-
	061	Nbhl_Svds_ItemCoun	0.9416		0.8144	0.8700	-0.6845	20	25	-	**Coun * 32**	-	-
	061	Nbhl_Svds_ItemDate	0.9426		0.8144	0.8700	-0.6845	20	25	-	**Date * 22**	-	-
	061	Nbhl_Svds_ItemGenr	0.9371		0.8144	0.8700	-0.6845	20	25	-	**Genr * 34**	-	-
	062	Nbhl_Svds_ItemComb	0.9358	984.5135	0.8144	0.8700	-0.6845	20	25	-	**Comb * 24**	-	-
	070	NbhU_Svds	0.9544	140.8521	0.8144	0.8700	-0.6845	**10**	-	70	-	-	-
	071	NbhU_Svds_UserAges	0.9544		0.8144	0.8700	-0.6845	10	-	70	-	**Ages * 0**	-
	071	NbhU_Svds_UserGend	0.9529		0.8144	0.8700	-0.6845	10	-	70	-	**Gend * 28**	-
	071	NbhU_Svds_UserOccu	0.9542		0.8144	0.8700	-0.6845	10	-	70	-	**Occu * 14**	-
	071	NbhU_Svds_UserZipc	0.9544		0.8144	0.8700	-0.6845	10	-	70	-	**Zipc * 0**	-
	072	NbhU_Svds_UserComb	0.9499	457.2092	0.8144	0.8700	-0.6845	10	-	70	-	**Comb * 16**	-
	080	NbIU_Svds	0.9462	1496.0000	0.8144	0.8700	-0.6845	**10**	25	70	-	-	70/30
	081	NbIU_Svds_ItemCoun_UserGend	0.9410		0.8144	0.8700	-0.6845	10	25	70	**Coun * 30**	**Gend * 30**	70/30
	081	NbIU_Svds_ItemCoun_UserOccu	0.9412		0.8144	0.8700	-0.6845	10	25	70	**Coun * 30**	**Occu * 20**	70/30
	081	NbIU_Svds_ItemDate_UserGend	0.9402		0.8144	0.8700	-0.6845	10	25	70	**Date * 25**	**Gend * 30**	70/30
	081	NbIU_Svds_ItemDate_UserOccu	0.9417		0.8144	0.8700	-0.6845	10	25	70	**Date * 25**	**Occu * 20**	70/30
	081	NbIU_Svds_ItemGenr_UserGend	0.9350		0.8144	0.8700	-0.6845	10	25	70	**Genr * 45**	**Gend * 30**	70/30
	081	NbIU_Svds_ItemGenr_UserOccu	0.9352		0.8144	0.8700	-0.6845	10	25	70	**Genr * 40**	**Occu * 30**	70/30
	082	NbIU_Svds_ItemComb_UserComb	0.9349	1557.4000	0.8144	0.8700	-0.6845	10	25	70	**Comb * 30**	**Comb * 20**	70/30