Pedro Ricardo Gomes Dias

Licenciado em Engenharia Informática

Recommending media content based on machine learning methods

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática

Orientador : Prof. Doutor João Magalhães, Prof. Auxiliar Convidado, FCT/UNL

	Júri:
Presidente:	Prof. Doutor Nuno Preguiça
Arguente:	Prof. Doutor Francisco Revilla
Vogal:	Prof. Doutor João Magalhães

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Novembro, 2011

Recommending media content based on machine learning methods

To Rieux, for staying alive

Acknowledgements

Elaborating this thesis has been a long and fascinating journey through a path of discipline, hard work, discovery and rewarding sense of accomplishment. I did not walk this path alone, which is why I owe those who supported me throughout this process a sincere and grateful acknowledgement.

To my thesis advisor Prof. Dr. João Magalhães, for his confidence in my work, for his guidance and knowledge sharing, for his flexibility and availability, and for his remarkable ability to keep me highly motivated that made the elaboration of this thesis such a pleasant and rewarding experience.

To my faculty Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, in particular to the Informatics Department, for providing an exceptional environment for making science and for leading a high-quality academic life.

To the Immersive TV project in which this thesis is inserted, for granting me a scholarship that helped me carry on with my studies.

To my teachers and friends Prof. Dr. Christopher D. Auretta and Prof. Dr. Ruy Costa, for being such a rich and inspiring example.

To all my university colleagues and friends, in particular Diogo Bernardino, Gustavo Azevedo, João Santos, Márcia Silva, Pedro Ramos Afonso, Sofia Gomes and Tewodros A. Beyene for their friendship and company along the way.

To all my long time friends, in particular André Macedo, Carla Ferreira, Filipe Macedo, Hugo Aguiar and Sofia Canelas for always being there for me and for their unconditional friendship, example and frequent laughters.

To my wonderful mother Judite Martins and sister Rita Gomes for loving me and shaping me deeply in so many different ways.

To my father José Dias for supporting and loving me.

To everyone else in my wonderful and generous family for giving me a home.

To my beautiful and graceful girlfriend Agnieszka Kapral, for brightening up my days.

To all of you, thank you so much.

Abstract

Information is nowadays made available and consumed faster than ever before. This information technology generation has access to a tremendous deal of data and is left with the heavy burden of choosing what is relevant. With the increasing growth of media sources, the amount of content made available to users has become overwhelming and in need to be managed. Recommender systems emerged with the purpose of providing personalized and meaningful content recommendations based on users' preferences and usage history. Due to their utility and commercial potential, recommender systems integrate many audiovisual content providers and represent one of their most important and valuable services. The goal of this thesis is to develop a recommender system based on matrix factorization methods, capable of providing meaningful and personalized product recommendations to individual users and groups of users, by taking into account users' rating patterns and biased tendencies, as well as their fluctuations throughout time.

Keywords: recommender systems, collaborative filtering, matrix factorization, group-based recommendations, interactive TV

Resumo

A informação é hoje em dia disponibilizada e consumida mais depressa do que nunca. Esta geração das tecnologias da informação tem acesso a uma enorme quantidade de diferentes conteúdos, competindo-lhe a árdua tarefa de seleccionar aqueles que são relevantes. Com o crescente aumento das fontes de informação, a quantidade de conteúdos disponibilizados aos utilizadores tornou-se avassaladora, surgindo a necessidade da existência de meios que façam a sua gestão. Os sistemas de recomendação surgiram com o propósito de oferecer sugestões de conteúdos personalizadas e pertinentes, baseadas nas preferências dos utilizadores e no seu histórico. Devido à sua utilidade e potencial comercial, os sistemas de recomendação integram diversos fornecedores de conteúdos audiovisuais, representando um dos seus mais valiosos e importantes serviços. O objectivo desta dissertação é desenvolver um sistema de recomendação baseado em métodos de factorização de matrizes, capaz de fazer recomendações pertinentes e personalizadas de produtos a utilizadores individuais e a grupos de utilizadores, tendo em conta os seus padrões de *rating* e respectivos desvios individuais, bem como as suas flutuações ao longo do tempo.

Palavras-chave: sistemas de recomendação, métodos colaborativos, factorização de matrizes, recomendações de grupo, TV interactiva

Contents

List of Figures

List of Tables

1 Introduction

1.1 Recommender systems

This is an information era. Information is nowadays made available and consumed faster than ever before. This information technology generation has access to a tremendous deal of data and is left with the heavy burden of choosing what is relevant. With the increasing growth of media sources, the amount of content made available to users has become overwhelming and in need to be managed. The quality of user experience is thus defined by how this phenomenon is dealt with. Because browsing the wide range of available content is impractical on a user perspective, it becomes crucial to find ways of identifying the small portion of relevant content by searching through all possibilities. The ability to perform such filtering determines the quality and productivity of user experience when interacting with information sources.
Recommender systems emerged with the purpose of providing personalized and meaningful content recommendations based on user preferences and usage history. Relying on the closest friends, family members or anyone else with whom one shares similarities to give trustworthy and useful advices has always been a characteristic of human behaviour, and different opinions weigh differently when it comes to making the final choice. The limitation on receiving good opinions from other people starts with the fact that, usually, one does not have many trustworthy or like-minded people to rely on for getting advice, and those few people have very limited knowledge, considering everything that exists and can be recommended, on a global scale point of view. Acknowledging this leads to wondering how to get suggestions with an acceptable degree of reliability. Stated simple: *How can I find peers with preferences similar to my own who would most definitely suggest items I would like?*

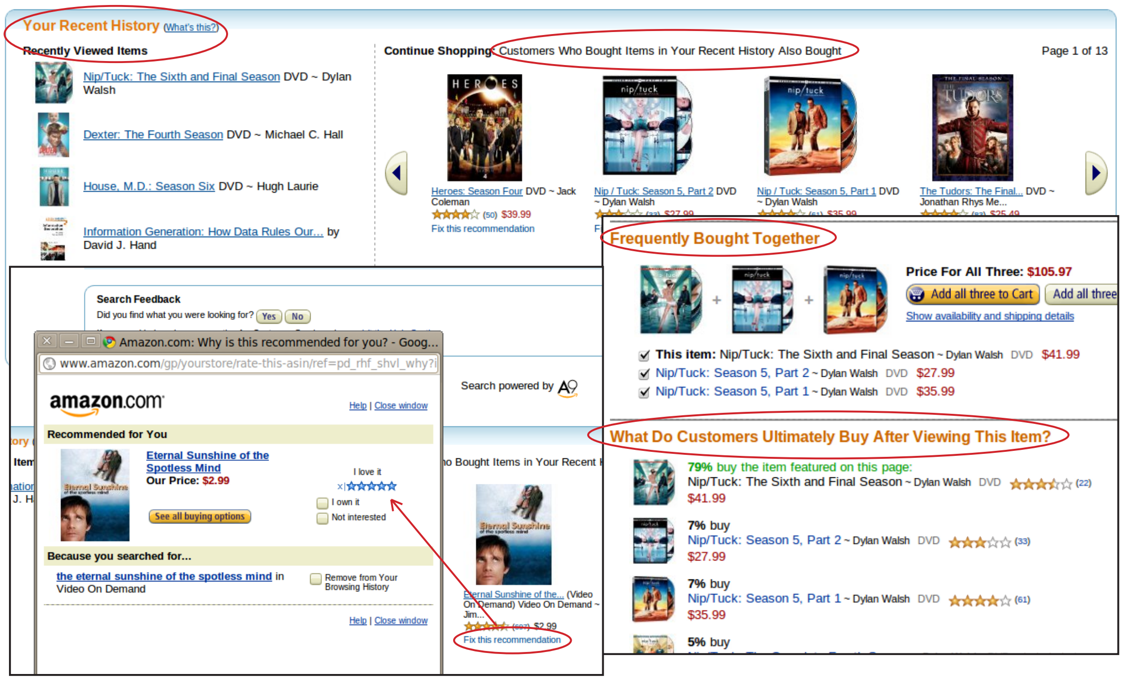
1.2 Media marketplaces and consumption

Recommender systems are offered by the widest audiovisual content providers, such as Amazon video-on-demand (VoD), Hulu or NetFlix, and represent one of their most important and valuable services [11]. The amount of available data involved is enormous due to the millions of users and the millions of products registered on their databases. Most of these systems operate similarly: users are encouraged to give ratings to products and based on those ratings and on any additional information about the users, such as demographic data and search history, products are recommended.

1.2.1 Amazon video-on-demand (VoD)

Amazon is a worldwide online marketplace with millions of registered users and products. At Amazon's VoD section one can easily notice the presence of a recommender system after performing a few searches for content or just navigating through and visualizing some of the available products. Recommendations based on this kind of search and navigation history, referred to as *implicit user feedback*, are made to the user from the first interactions with the system. Figure 1.1 shows an example of product recommendations based on the lists of products recently viewed by the user.

Additionally, the system provides hints about other users' consumption patterns related to products the target user might be interested in, as shown on Figure 1.1. To improve the accuracy of recommendations, Amazon's recommender system often offers their users the possibility of providing *explicit feedback* by rating products in a 1 to 5 scale or toggling a *like* button for a specific product. These possibilities are offered in many ways, an example is the *"Fix this recommendation"* feature, also illustrated by Figure 1.1, that allows for the user to provide a 1 to 5 rating to a product that has been recommended, thus *fixing* the system's rating prediction. The *"Fix this recommendation"* feature shows that the predictive model used by the system is constantly open to improvements. The system also allows for the user to search for specific products to rate, under the promise of providing better recommendations in the future, as shown on Figure 1.2.



Your Recent History (What's this?)

Recently Viewed Items

Nip/Tuck: The Sixth and Final Season DVD ~ Dylan Walsh

Dexter: The Fourth Season DVD ~ Michael C. Hall

House, M.D.: Season Six DVD ~ Hugh Laurie

Information Generation: How Data Rules Our... by David J. Hand

Continue Shopping: Customers Who Bought Items in Your Recent History Also Bought

Page 1 of 13

Heroes: Season Four DVD ~ Jack Coleman (50) $39.99

Fix this recommendation

Nip / Tuck: Season 5, Part 2 DVD ~ Dylan Walsh

Nip / Tuck: Season 5, Part 1 DVD ~ Dylan Walsh

The Tudors: The Final... DVD ~ Jonathan Rhys Me...

Search Feedback

Did you find what you were looking for? Yes No

Amazon.com: Why is this recommended for you? - Goog...

www.amazon.com/gp/yourstore/rate-this-asin/ref=pd_rhf_shvl_why?

amazon.com Help | Close window

Recommended for You

Eternal Sunshine of the Spotless Mind

Our Price: $2.99

See all buying options

I love it

I own it

Not interested

Because you searched for...

the eternal sunshine of the spotless mind in Video On Demand

Remove from Your Browsing History

Help | Close window

Search powered by A9

...o Bought Items in Your Recent...

Eternal Sunshine of the... (Video On Demand) Video On Demand ~ Jim... $2.99

Fix this recommendation

Frequently Bought Together

Price For All Three: $105.97

Add all three to Cart | Add all three

Show availability and shipping details

This item: Nip/Tuck: The Sixth and Final Season ~ Dylan Walsh DVD $41.99

Nip/Tuck: Season 5, Part 2 ~ Dylan Walsh DVD $27.99

Nip/Tuck: Season 5, Part 1 ~ Dylan Walsh DVD $35.99

What Do Customers Ultimately Buy After Viewing This Item?

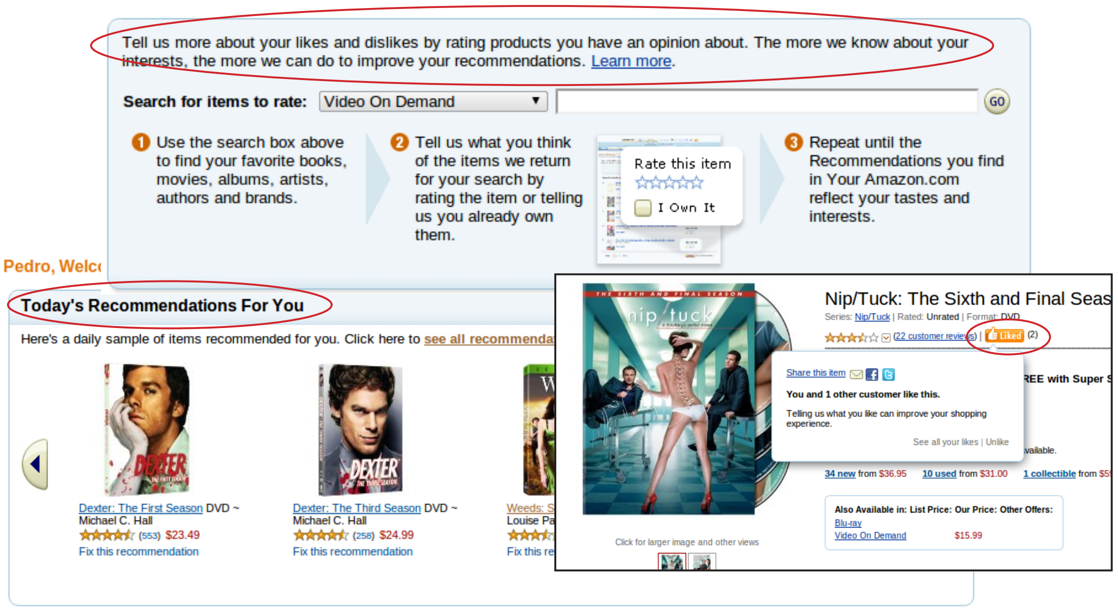
79% buy the item featured on this page: Nip/Tuck: The Sixth and Final Season ~ Dylan Walsh DVD (22) $41.99

7% buy Nip/Tuck: Season 5, Part 2 ~ Dylan Walsh DVD (33) $27.99

7% buy Nip/Tuck: Season 5, Part 1 ~ Dylan Walsh DVD (61) $35.99

5% buy

Figure 1.1: Amazon recommendations based on a user recent history and consumption patterns of similar users. *Fix this recommendation* feature.



Tell us more about your likes and dislikes by rating products you have an opinion about. The more we know about your interests, the more we can do to improve your recommendations. Learn more.

Search for items to rate: Video On Demand GO

1 Use the search box above to find your favorite books, movies, albums, artists, authors and brands.

2 Tell us what you think of the items we return for your search by rating the item or telling us you already own them.

Rate this item

I Own It

3 Repeat until the Recommendations you find in Your Amazon.com reflect your tastes and interests.

Pedro, Welc

Today's Recommendations For You

Here's a daily sample of items recommended for you. Click here to see all recommenda

Dexter: The First Season DVD ~ Michael C. Hall (553) $23.49

Fix this recommendation

Dexter: The Third Season DVD ~ Michael C. Hall (258) $24.99

Fix this recommendation

Weeds: S... Louise Pa...

Fix this re...

Nip/Tuck: The Sixth and Final Seas

Series: Nip/Tuck | Rated: Unrated | Format: DVD

(22 customer reviews) | Liked (2)

Share this item

You and 1 other customer like this.

Telling us what you like can improve your shopping experience.

See all your likes | Unlike

...REE with Super S...

...vailable.

34 new from $36.95 10 used from $31.00 1 collectible from $5...

Click for larger image and other views

Also Available in: List Price: Our Price: Other Offers:

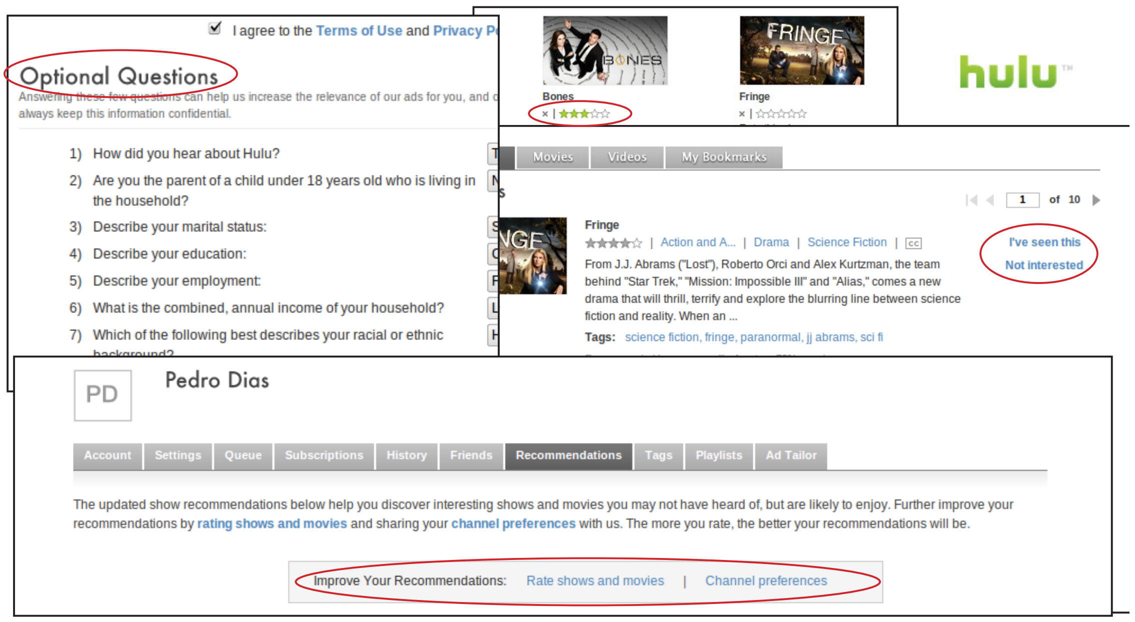
Blu-ray

Video On Demand $15.99

Figure 1.2: Amazon's *"Today's recommendations"*, search and rate feature and *like* button.

1.2.2 Hulu

Hulu is a website that provides video streaming of TV shows and movies from a wide panoply of TV networks and studios, having registered hundreds of thousands of streams viewed monthly during 2010. The recommender system integrated in Hulu website applies user feedback and profile characteristics to make recommend products. Upon registering, the user is asked to introduce some optional personal information such as marital status, annual income or ethic background, as shown on Figure 1.3, that will be used to set up a user profile, thus adding some content-based methodologies to the recommendation algorithm used to make product suggestions. Like many other collaborative filtering recommender systems, Hulu is based mostly on user feedback to compute predictions about what products the users will like. Users can give their explicit feedback by giving ratings to products in a scale that ranges from 1 to 5. Additionally, users can explicitly inform the system about which videos they have already watched or which ones they simply have no interest in, as Figure 1.3 illustrates, allowing for the system to know more about users' preferences. Moreover, on the user personal area there is a *recommendations* tab on which the user gets the solicitation *Improve Your Recommendations*, being then led to rate more products or manifest channel preferences, as shown on Figure 1.3.



I agree to the Terms of Use and Privacy P

Optional Questions

Answering these few questions can help us increase the relevance of our ads for you, and always keep this information confidential.

1) How did you hear about Hulu?
2) Are you the parent of a child under 18 years old who is living in the household?
3) Describe your marital status:
4) Describe your education:
5) Describe your employment:
6) What is the combined, annual income of your household?
7) Which of the following best describes your racial or ethnic background?

Fringe
Action and A... | Drama | Science Fiction
From J.J. Abrams ("Lost"), Roberto Orci and Alex Kurtzman, the team behind "Star Trek," "Mission: Impossible III" and "Alias," comes a new drama that will thrill, terrify and explore the blurring line between science fiction and reality. When an ...
Tags: science fiction, fringe, paranormal, jj abrams, sci fi

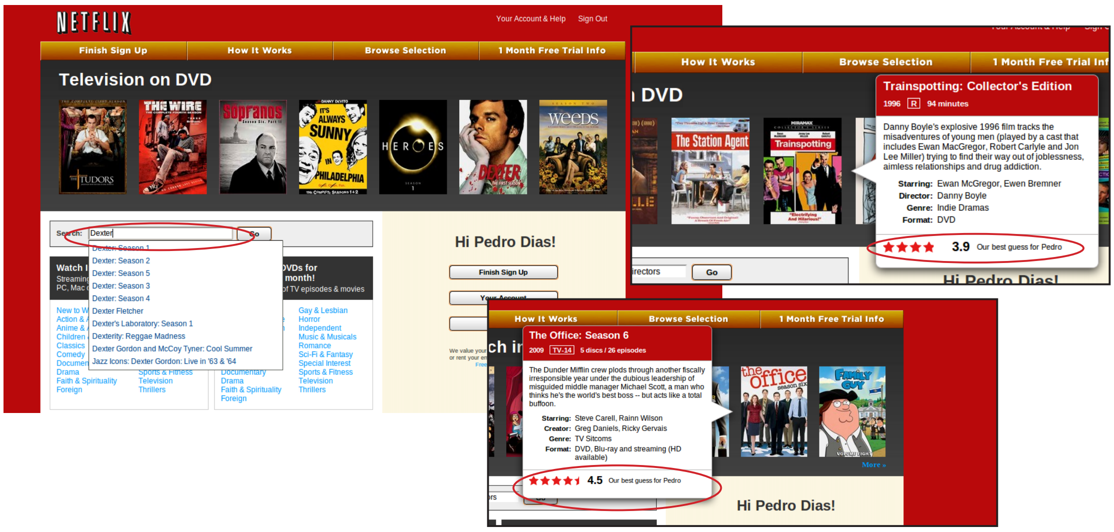
The updated show recommendations below help you discover interesting shows and movies you may not have heard of, but are likely to enjoy. Further improve your recommendations by rating shows and movies and sharing your channel preferences with us. The more you rate, the better your recommendations will be.

Improve Your Recommendations: Rate shows and movies | Channel preferences

Figure 1.3: Hulu's rating scale, demographic data solicitation and *Improve Your Recommendations* feature.

1.2.3 Netflix

Netflix is an online company that provides on-demand streaming video and dvd rental by mail, counting more than 100,000 titles and above 10 million registered users. Like the previously mentioned recommender systems, Netflix relies on user feedback and profile to make recommendations. Users can rate products from 1 to 5 and search and navigation history are taken into account in the process of generating recommendations. When prompted with product suggestions, upon consulting the recommended product's details, Netflix's recommender system shows its *"best guess"* rating prediction for that user, as illustrated on Figure 1.4.



Search: Dexter

Dexter: Season 1
Dexter: Season 2
Dexter: Season 5
Dexter: Season 3
Dexter: Season 4
Dexter Fletcher
Dexter's Laboratory: Season 1
Dexterity: Reggae Madness
Dexter Gordon and McCoy Tyner: Cool Summer
Jazz Icons: Dexter Gordon: Live in '63 & '64

Trainspotting: Collector's Edition
1996 R 94 minutes

Danny Boyle's explosive 1996 film tracks the misadventures of young men (played by a cast that includes Ewan MacGregor, Robert Carlyle and Jon Lee Miller) trying to find their way out of joblessness, aimless relationships and drug addiction.

Starring: Ewan McGregor, Ewen Bremner
Director: Danny Boyle
Genre: Indie Dramas
Format: DVD

★★★★ 3.9 Our best guess for Pedro

The Office: Season 6
2009 TV-14 5 discs / 26 episodes

The Dunder Mifflin crew plods through another fiscally irresponsible year under the dubious leadership of misguided middle manager Michael Scott, a man who thinks he's the world's best boss -- but acts like a total buffoon.

Starring: Steve Carell, Rainn Wilson
Creator: Greg Daniels, Ricky Gervais
Genre: TV Sitcoms
Format: DVD, Blu-ray and streaming (HD available)

★★★★½ 4.5 Our best guess for Pedro

Figure 1.4: Netflix's search box from which preferences are inferred and *Our best guess* feature.

1.2.3.1 Netflix prize

In 2006 Netflix held a contest offering 1 million dollars to anyone who could create a collaborative filtering recommendation algorithm that would surpass their own - by then named *Cinematch* - in terms of prediction accuracy by at least 10%. Only 3 years later the team *BellKor's Pragmatic Chaos*, which was a fusion between several teams initially competing for the prize individually, accomplished such deed. Every team had available to train and test their algorithms around 100 million ratings given by more than 480,000 users to almost 18,000 movies. The fact that Netflix was willing to pay such amount of money for a recommender algorithm stresses how valuable and important good recommendations are for the success of this kind of business.

1.3 Problem definition and thesis objective

This project is inserted in an interactive TV (iTV) context where the iTV provider wants to offer its customers suggestions on what they might enjoy watching. In this context, the scope of possible recommendable products is within multimedia content, and suggestions are meant to be as accurate as possible, thus maximizing user satisfaction. From the iTV provider point of view the main challenge subsists, but the question is slightly different: *Which recommendations would get customers to consume the recommended products?* To answer this question some other aspects must be clarified beforehand. The first aspect lies in understanding the surrounding environment. In this case, it is a TV environment, therefore widely populated. Millions of people are TV content consumers and many different things can be watched on TV, as well. So, it is about:

Millions of users $\times$ *Millions of products* $=$ *Trillions of data to work on*

Consumption patterns can be inferred from this information and users may share similar consumption habits among each other. Simplified, if somehow users can be matched according to consumption patterns and preference similarities they can be reliable advice-givers to each other.
Once both users and products are conveniently characterized recommendations can be made using different methods:

- **Content-based methods**, in which the characteristics inherent to products and users are explored to assess similarities, and recommendations are made according to those similarities;

- **Collaborative methods**, in which the users' previous purchases and consumption habits are explored, and recommendations are made based on that;

Within an interactive TV project, there is the need to assess each user's individual preferences to provide a service as personalized as possible that fits their needs and, by doing so, stimulate consumption. Providing such service involves making personalized product recommendations with a good degree of accuracy.
The accuracy of the recommendation system depends on the available data to work on and the methods used to combine this data so that recommendations can be then produced. To do so, several different methods involving content-based and/or collaborative filtering must be experimented and analysed, and the extent to which these may be fit for this specific problem and environment must be determined. This can only be done after achieving an understanding about the surrounding environment and its actors: users and products.
Users are the main actors in the system, since they are the consumers and the data providers. Users can be defined by two main classes of attributes:

- *Demographic data*, which includes every personal information such as age, gender, nationality, area of residence, marital status, ethnic background, average income, etc. This kind of data should be mapped onto numerical values so it can be processed. As an example, MovieLens stores the ages of users in age groups identified by a number(ex.: 1: 18-24; 2: 25-34). Not all this data is always provided by users but, whenever available, it can be used to complement collaborative algorithms with content-based features.

- *Preferences*, which can be explicitly expressed to the system through product ratings, or inferred from history of visualized and/or searched products and from user's consumption habits.

Products are represented by their inherent characteristics. In this case, the products are movies that can be characterized by title, genre, length, year of production, actors, director, etc. Depending on the method and process of generating recommendations, the

dimensions of the referential on which products are characterized may change. Products will then be represented by the extent to which they fit in each of the referential dimensions (ex.: how funny, how reflexive, how scary), mapped onto numerical values, like coordinates on a map.
The relation between a user and a product is defined by the extent to which that user likes that product. This relation can be easily determined when the user rated the product, and its representation is simple: a value, usually between 1 and 5. However, most of the times there are no ratings relating users to products. In this cases, the system shall try to predict the rating a user would give to a product. By doing so, the system is then able to recommend products that the user will probably like. The best techniques and methods to achieve that may differ, but the latest studies show that the most successful approaches to this kind of problem involve matrix factorization methods. Moreover, preferences evolution over time and the ability to infer preferences through implicit feedback also contribute to improve the quality of predictions.
Thus, the objective of this thesis is:

to develop a recommendation algorithm based on (i) the collaborative ratings of products, (ii) the rating biases associated to users and products, (iii) the temporal fluctuations of ratings and (iv) group preferences.

To accomplish this objective, a baseline matrix factorization model shall be implemented based on the ratings given to products by users. This model must then be improved by adding information accounting for user-related and product-related rating biases. The baseline model shall be further improved by contemplating temporal fluctuations of ratings, through inference of rating pattern changes over time. Finally, matrix factorization methods will be combined with clustering methods to provide group-based recommendations.

1.4 Contributions

The contributions of this thesis are:

- A software package implementing matrix factorization based collaborative filtering techniques for recommender systems, using a Stochastic Gradient Descent (SGD) learning algorithm with optional parallel computation. This implementation also contemplates the use of rating biases and temporal fluctuations, inpired by Y. Koren's work [20].

- A software package implementing a group-based recommendation framework. The design and architecture of this framework was also proposed by me.

- A scientific paper submitted to the European Conference on Information Retrieval (ECIR) 2012.

The matrix factorization implementation described in this thesis produced results that ranked me among the top 7% teams of the Yahoo! SIG-KDD CUP 2011.

1.5 Organization

The rest of this thesis is organized as follows:

- **Chapter 2:** Related work overview addressing the topics of data representation, similarity metrics, group discovery and recommendation algorithms and techniques.
- **Chapter 3:** Matrix factorization methods and associated computational constraints, implementation challenges and evaluation.
- **Chapter 4:** Embedding rating biases and temporal dynamics to the matrix factorization methods and associated computational constraints, implementation challenges and evaluation.
- **Chapter 5:** Group-based recommendations combining matrix factorization with clustering techniques and respective evaluation.
- **Chapter 6:** Conclusions and future work, to be accomplished along the course of the imTV project.

2

Background and related work

2.1 Introduction

Recommender systems emerged with the intent of tackling the problem of multimedia content overload and provide meaningful recommendations. Recommender systems have gained noticeable popularity for the past few years, partially due to the Netflix Prize contest held by Netflix - an online DVD rental company - in 2007, awarding with $1M the first team to outperform *Cinematch* (Netflix's recommender system) with a 10% accuracy improvement. It was not before 2 years later that the team Bellkor's Pragmatic Chaos finally achieved this goal. More recently, Yahoo held the recommender system contest KDD Cup 2011 on which hundreds of contestants competed to reach the top scores, i.e. to produce user rating predictions with the lowest possible error, stressing the value of this research field.

2.2 Recommendation techniques

Recommender systems rely on two main categories of methods: *Content-based filtering* and *Collaborative filtering*. The two mentioned categories of methods will be further addressed in 2.2.1 and 2.2.2, respectively. Additionally, there are *Hybrid Approaches* that combine both the aforementioned methods in different ways:

- Producing content-based and collaborative recommendations separately and then combining the individual results;

- Introducing some content-based filtering characteristics into collaborative filtering methods, and vice-versa;

- Setting up a model which inherently incorporates both content-based filtering and Collaborative Filtering methods.

All these categories of methods have been extensively explored and compared throughout the years. G. Adomavicius et al. [1] and Melville et al. [27] presented useful surveys on the matter, where they classified methods, compared approaches and pointed out limitations.

2.2.1 Content-based filtering

Content-based filtering approaches aim at exploring users' and products' inherent characteristics to produce recommendations. These approaches have their roots in text processing applications [32] and information retrieval [2], where content is mostly textual. When taking this approach towards recommender systems, users and products cannot be seen as atomic elements. Instead, these need to have a more descriptive representation, such as

$$u_i = (u_{i1}, u_{i2}, \cdots, u_{im}) \quad \text{and} \quad p_j = (p_{j1}, p_{j2}, \cdots, p_{jn}),$$

where u_i represents user i and the elements from u_{i1} to u_{im} represent the m characteristics of user i. Similarly, p_j represents product j and the elements from p_{j1} to p_{jn} represent the n characteristics of product j. These characteristics can be gender, nationality, age, genres of music, names of directors, etc., depending on the context and purpose of the recommender system, and can be interpreted as keywords to describe a user or a product. Based on these characteristics, similarities between users or products can be assessed and used to produce meaningful recommendations. Content-based approaches rely on the assumption that similar users like the same products and that users that consumed a given product will also like products similar to the one consumed. However, some limitations exist when recommending products through content-based filtering, such as the need for accurate - but general enough - descriptions of users and products, without which realistic comparisons are difficult. Another limitation of this approach, referred to as *overspecialization* in [1], lies in the fact that users are bound to only get recommendations of products with similar characteristics to those they have consumed before.

2.2.2 Collaborative filtering

Unlike content-based approaches, collaborative filtering methods do not take into account the inherent characteristics of users or products. Instead, this approach attempts to infer user preferences by analysing the patterns and historic of consumption of all users in the system, mining the relations between users and products based on their interactions. The roots of collaborative filtering can be traced back to 1992, when Goldberg et al. proposed an electronic mail filtering system [9] where users could contribute with their feedback about the content they read, allowing for a collaborative effort of sorting relevant from irrelevant e-mail messages. Another early application of collaborative filtering was the open architecture GroupLens, implemented by Resnick et al. [30] with a similar purpose, only this time aiming at filtering netnews based on ratings given by users. This approach introduced the concept of user feedback, which is the input provided by users regarding products. User feedback can be provided explicitly by users in the form of ratings (explicit feedback) or it can be extracted from user activity analysis (implicit feedback). Although explicit feedback is more reliable because it reflects a preference intentionally provided by the user, implicit feedback is an important source of data to complement explicit feedback, especially when there are not many available ratings. This often happens when new users or products are introduced in the system, which in literature is commonly referred to as the "cold start" problem [27, 1]. Moreover, implementations for exclusively implicit feedback datasets had already been proposed back in 1998, when D.Oard et al. [28] explored the use of implicit data to produce recommendations. More recently, Y.Koren [12] presented a framework to rely exclusively on this kind of input, refining it by associating implicit observations with varying confidence levels. Additionally, other implementations were proposed [18, 15] incorporating both kinds of input. Collaborative filtering can be divided in two broad types of models: *Neighborhood models* and *Latent factor models*. In 2004, J.Herlocker et al. [10] conducted an extensive empirical analysis of the most relevant collaborative filtering methods, evaluating and comparing them against each other. Since then, both these approaches have had many developments [21], having attracted particular attention from the scientific community during the Netflix Prize [38, 19, 22, 40, 37] competition, standing nowadays as state-of-the-art techniques for recommender systems.

2.2.2.1 Neighbourhood models

Neighbourhood models try to infer user preferences based on the preferences of like-minded users. The most commonly applied neighbourhood method is the k-nearest-neighbours (k-NN), which can be either user-oriented or item-oriented. On a user-oriented approach rating predictions for a user are calculated as a weighted average of the ratings given by the *k* users most similar to the target user. Similarities between users are calculated according to their product ratings and can be represented on a symmetric user-user similarity matrix:

$$\begin{bmatrix} 1 & \cdots & s_{1,m} \\ \vdots & 1 & \vdots \\ s_{n,1} & \cdots & 1 \end{bmatrix}$$

Figure 2.1: User-user similarity matrix

Each $s_{u1,u2}$ value on the user-user similarity matrix represents a similarity score between users $u1$ and $u2$ for all m users in the system. Higher values correspond to higher similarities and lower values correspond to lower similarities. The item-oriented approach [33, 41] works analogously to the user-oriented approach: similarities between products are measured and represented on an item-item similarity matrix. Then, rating predictions are calculated as a weighted average of the ratings given by the target user to the k products, within the set of products rated by the user, that are more similar to the target product. This approach has been preferred by web-based companies such as Amazon[23] for being more scalable and allowing for a better explanation of the obtained results, since recommendations are made based on comparisons between products that the user knows and consumed, instead of comparisons with users unknown to the target user. There are several possible similarity metrics suggested in the literature [33, 37], which will be further addressed in section 2.3. Once defined the similarity metric to be used within a k-NN method, product rating predictions are given by

$$\hat{r}_{ui} = \frac{\sum_{j \in S^k_{(iu)}} s_{ij} \cdot r_{uj}}{\sum_{j \in S^k_{(iu)}} s_{ij}}, \tag{2.1}$$

where $S^k_{(iu)}$ is a set with the k products rated by user u that are more similar (i.e., have a higher similarity score according to the adopted similarity metric) to product i, s_{ij} is the similarity score between products i and j and r_{uj} is the rating given by user u to product j. Once computed all product rating predictions for a given user, the products with higher rating prediction can be recommended to the target user. Additionally, Y.Koren et al. [4] proposed a version of k-NN models enhanced by calculating interpolation weights between products instead of computing each product weight separately.

2.2.2.2 Latent factor models

Latent factor models emerge as an attempt to represent users and products under the same feature space. Applications of such approach include neural networks [31], latent dirichlet allocation [5] and Singular Value Decomposition (SVD) [34]. The principle underlying latent factor approaches is that both users and products can be represented under a common reduced dimensionality space of latent factors that are inferred from the data and explain the rating patterns. Chapter 3 will provide a thorough clarification of the latent factor approach. Individually, latent factor approaches have proven to yield better results than any other methods in terms of predictive accuracy, as the literature produced during the Netflix Prize competition shows. Nevertheless, the winning solution of the Netflix Prize [19] resulted from a combination of the output generated by different collaborative filtering methods, obtained through well-established blending techniques [13].

2.3 Similarity metrics

To determine similarity scores between users, a metric that returns a numerical value to work with must be defined. The similarity metrics most commonly used to compare users are addressed in this section.

2.3.1 Pearson correlation coefficient

The most widely used similarity metric is the Pearson correlation coefficient [36], which measures the extent to which two variables are linearly dependent. In a context where the goal is to find similarities between users based on their item ratings, it can be defined by:

$$sim_{ab} = \frac{\sum\limits_{j \in I}(r_{aj} - \bar{r}_a)(r_{bj} - \bar{r}_b)}{\sqrt{\sum\limits_{j \in I}(r_{aj} - \bar{r}_a)^2 \sum\limits_{j \in I}(r_{bj} - \bar{r}_b)^2}} \tag{2.2}$$

where I is the set of items rated by both user a and user b, r_{aj} is the rating given by user a to item j, r_{bj} is the rating given by user b to item j and $\bar{r}_u$ is the average rating given by user u.

Pearson correlation coefficient has the advantage of dealing with the problem that comes from different users having different rating scales, which could lead to misinterpretations about user preferences similarities. This problem is addressed by subtracting the average rating $\bar{r}_u$ from each item rating r_{uj}.

2.3.2 Cosine similarity

As an alternative, the ratings of each user can be treated as vectors and compared to other users' ratings by calculating the cosine of the angle between them, thus obtaining a cosine similarity defined by:

$$sim_{ab} = \cos \vec{r_a}, \vec{r_b} = \frac{\vec{r_a} \cdot \vec{r_b}}{\|\vec{r_a}\|^2 \times \|\vec{r_b}\|^2} \tag{2.3}$$

To calculate cosine similarity, there can be no negative ratings and items left unrated are treated as having a rating of zero. This is not a problem when the set I contains only items rated by both users being compared and the rating scale does not allow ratings below zero. However, empirical studies have proven Pearson correlation coefficient to usually have better results than cosine similarity [6].

2.3.3 Tanimoto coefficient

Tanimoto coefficient is a measure of similarity between two sets. It returns a ratio between the intersection and the union of two sets, thus showing how much two sets have in common comparing to what they don't. It is defined by:

$$sim_{a,b} = \frac{|a \cap b|}{|a \cup b|} = \frac{|a \cap b|}{|a| + |b| - |a \cap b|} \tag{2.4}$$

where $|a|$ and $|b|$ represent the number of elements in set a and in set b, respectively, $|a \cap b|$ is the number of elements common to set a and set b and $|a \cup b|$ is the number of elements within the union between set a and set b. The similarity score between two sets obtained with Tanimoto coefficient ranges from 0 (no elements in common) to 1 (all elements in common). This similarity metric is particularly useful when comparing two users based on attributes with binary value. For example, if the feedback given by two users about items within a set is limited to *like/don't like* ratings, or if the comparison between two users is based on whether they bought or didn't buy items within a set, then Tanimoto coefficient would be a good candidate for measuring similarities. The real-world applications for this similarity metric are mostly group discovery methods.

2.4 Group recommendation

Although recommender systems have recently attracted a lot of attention from the scientific community, group recommendation has not been widely addressed, since most recommendation techniques are oriented to individual users and focus on maximizing the accuracy of their preference predictions. A. Jameson et al. [14] conducted an enlightening survey in 2007 presenting the most relevant works on the field of group recommendation, as well as the most common issues addressed by the authors of the surveyed group recommender systems.

The main challenges faced when providing group recommendations are (1) capturing user preferences, (2) combining user preferences into a representation of group preferences, (3) defining criteria to assess the adequacy of recommendations, and (4) delivering recommendations. Group recommender systems can be compared according to how they deal with these challenges.

In 2002, the Flytrap system was proposed by A. Crossen et al. [7], presenting a simple system designed to build a soundtrack that would please all users within a group in a target environment. In Flytrap system, user preferences were obtained by registering what they listen to on their private computers, in an implicit fashion. Recommendations were then computed by comparing songs within the system database to those listened to by the group members based on artist and genre. Songs whose artist or genre are known to please more users within the target group were then more eligible to be recommended automatically, without the user having any control over what's being recommended. The content-based nature of recommendations provided by Flytrap is a constant in most group recommender systems described in literature.

A similar approach was taken in the system CATS (Collaborative Advisory Travel System) by McCarthy et al. [26]. CATS is a system designed to recommend travel packages to groups of users. It relies on a form of user feedback named *critiquing*, which consists in having the group users give their real-time opinion about some features associated with the recommended products in a *more of this / less of that* fashion. For example, when presented with a travel package recommendation a user can let the system know about his preference for a cheaper or shorter plan, without specifying price or duration values. This user feedback is recorded and linearly combined between all users within the group to be afterwards compared against the set of features that represent each travel package. The CATS systems can then recommend the travel packages that suit better the groups' critiques.

Another example of group recommender systems is the system Bluemusic proposed by Mahato et al. [24]. In this system users are detected via bluetooth and the awareness of their presence has direct influence on a playlist which is being played on a public place. To be taken into account, a user must register his preferences beforehand. The concept introduced by the Bluemusic system is very simple but introduces an interesting alternative for incorporating transient awareness of user presences into a real-time playlist recommendation scenario.

2.4.1 Discovering groups

Discovering groups of users that are somehow related is a useful step to produce personalized recommendations. Finding means of identifying groups of users with similar preferences narrows the scope of items to recommend, thus increasing the probability of recommending items that users will like. Performing data clustering allows for discovering groups and the two main methods used to achieve that are *hierarchical clustering* and *k-means method*, which will both be addressed in this section.

2.4.1.1 Hierarchical clustering

Hierarchical clustering [35] consists on continuously merging the two most similar groups until all groups are merged. A group can contain a single item and the algorithm starts with each item belonging to a different group. Similarity between groups is measured by the distance that separates them, which is calculated according to a given similarity metric (see Section 2.3). The algorithm, illustrated with a 5-item example by Figure 2.2, stops its iterations when there is only one group left. Depending on the application, hierarchical clustering can be applied to find groups of similar items or users.

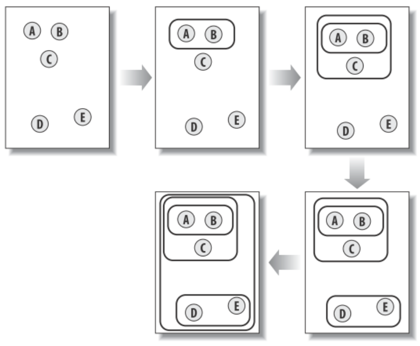


Figure 2.2: Hierarchical clustering algorithm iterations. Illustration taken from [35].

As the example illustrated by fig. 2.2 shows, items A, B, C, D and E are initially placed in a 2D space according to their characteristics and stand as individual groups. After the first iteration of the hierarchical clustering algorithm, items A and B are merged into a single group, since their proximity to each other was higher than the proximity between any other pair of groups, which were single items, at that point. After the second iteration, the group composed by items A and B is merged with the group containing item C, following the same reasoning on the first iteration. This process continues until there is only one group containing all items, as illustrated by the result of the final iteration on fig. 2.2.

The results from hierarchical clustering can then be viewed in a graph called *dendrogram*, where all merged groups are represented hierarchically in horizontal tree-form, in which the distance that previously separated two items is represented by the distance between the group node and the nodes of the two merged items. Figure 2.3 illustrates an example of a *dendrogram*. However, the tree view obtained from a *dendrogram* doesn't really break the data into distinct groups without additional work and the hierarchical clustering algorithm is computationally intensive, which can be a big disadvantage when dealing with large sets of data.

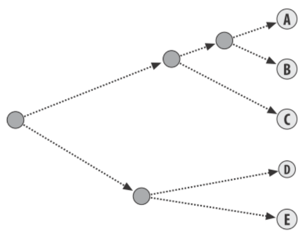


Figure 2.3: Dendrogram visualization of hierarchical clustering. Illustration taken from [35].

2.4.1.2 K-Means clustering

K-Means method [35] algorithm starts with a predefined number of clusters, which is based on the structure of the data. K-Means algorithm begins with randomly placing *k centroids*. A *centroid* is a point in space that represents the center of a cluster. After placing all *centroids*, the items, that are also placed in space according to their attributes, are assigned to the nearest *centroid*. All *centroids* then move to the average location of their assigned items and the assignments are redone. This process repeats until all *centroids* stop moving over iterations.

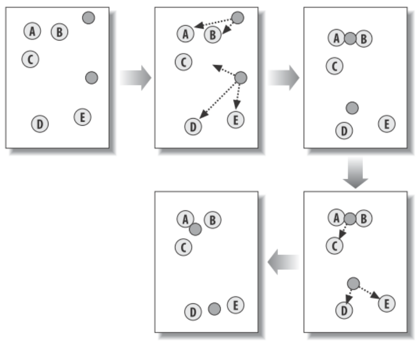


Figure 2.4: Iterations of K-means algorithm. Illustration taken from [35].

Fig. 2.4 illustrates an example of K-Means clustering with 5 items and 2 randomly placed *centroids*. In this example, the items A, B, C, D and E are placed on a 2D space and 2 *centroids*, represented as small dark circles, are randomly placed in that same 2D space. After the first iteration the items are assigned to their closest *centroid*, resulting in items A and B being assigned to one *centroid* and items C, D and E being assigned to the other one, as shown in the second of the 5 scenarios illustrated by fig. 2.4. Once all items are assigned to the closest *centroid*, each *centroid* is moved to the average location of the items assigned to it. The items are then reassigned to *centroids* according to the *centroids'* new location, resulting in items A, B and C being assigned to one *centroid* and items D and E being assigned to the other one, as shown in the third scenario. As one can observe, item C is now assigned to a *centroid* different from the one it was initially assigned to. This process continues until the *centroids* no longer change location.

2.5 Summary

In this chapter was discussed:

- how data must be represented, specifically users, products and ratings, so that it can be dealt with while performing recommendation algorithms;
- what similarity metrics can be used to measure similarities between products or users;
- what are the main algorithms for discovering groups of products or users;
- what are the main algorithms and techniques to produce product recommendations to users.

3

Recommendations by matrix factorization

3.1 Introduction

Matrix factorization is a class of linear algebra operations for decomposing matrices that can be used in collaborative filtering. Within the context of recommender systems, matrix factorization techniques are intended to be applied over a user-product ratings matrix where ratings given by users to products are stored, as illustrated by matrix R in 3.1.

$$R = \begin{bmatrix} r_{11} & \cdots & r_{1i} \\ \vdots & \ddots & \vdots \\ r_{u1} & \cdots & r_{ui} \end{bmatrix} \tag{3.1}$$

Here, each r_{ui} value in the matrix represents a rating given by user u to product i, expressed by a real value. Matrix factorization techniques have become attractive in the last few years for its accuracy and scalability, and hold nowadays an indisputable place at the top of the most successful techniques for recommender systems [19, 38, 3, 29, 22, 40]. A thorough explanation regarding the goal of applying matrix factorization to the user-product ratings matrix, as well as the decomposition it pursues and methods for obtaining such decomposition, are within the scope of this chapter and will be further addressed from section 3.2 onwards. Before looking into matrix factorization with more detail, let us revisit some important underlying concepts, introduced earlier in 2.2.2.2: latent factor approaches.
Matrix factorization techniques applied to recommender systems fall in the category

of latent factor approaches. Within collaborative filtering techniques, latent factor approaches are popular for their accuracy. The purpose of latent factor approaches is to infer implicit latent factors from a dataset that help explaining user-product interactions within the system. The goal of inferring these latent factors is to enable the mapping of both users and products onto the same latent factor space, representing these as vectors with k dimensions:

$$p_u = (u_1, u_2, \cdots, u_k) \tag{3.2}$$

$$q_i = (i_1, i_2, \cdots, i_k) \tag{3.3}$$

Here, p_u is the user u factors vector, q_i is the product i factors vector and k is the number of latent factors (dimensions) upon which each user u and each product i are represented. By representing users and products in such way, one can evaluate the extent to which users and products share common characteristics by comparing their k factors against each other. Koren et al. [22] presented a visual explanation for the motivation underlying the latent factors approach applied to a context where products are movies, which is an intuitive clarification of its intent, illustrated by fig. 3.1. On the example illustrated by

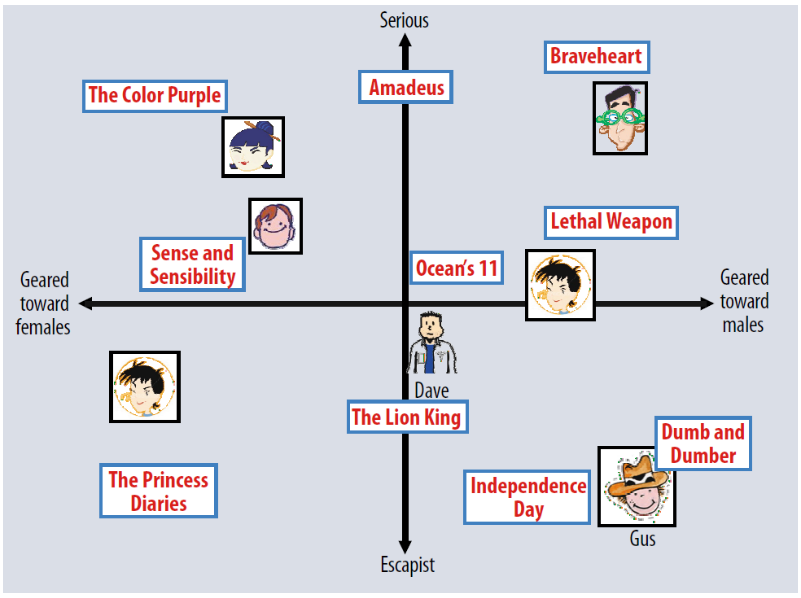


Figure 3.1: Latent factor approach with 2 factors where both users and products are represented under the same feature space. Illustration take from the work of Koren et al. [22].

Fig. 3.1 both users and products are mapped onto a two-dimension factor space allowing for an intuitive conclusion about their preferences towards the represented products. Although the factors on this example suggest interpretable product characteristics, such as the clear paradox between serious and escapist movies, latent factors do not always have an intuitive meaning, often merely representing abstract features with no real-life interpretation. The goal of such approach is to enable the assessment of user preferences for products by calculating the dot product of their factor representations, as defined by eq. 3.4:

$$r_{ui} = p_u \cdot q_i, \tag{3.4}$$

Here, r_{ui} is the preference of user u for product i, both represented as vectors as described in 3.2 and 3.3. For example, by observing the placement of users and products on the factors map in Fig. 3.1 let us assume that the user "Gus" is represented by

$$p_{Gus} = (Gus_{toMales}, Gus_{serious}) = (4, -4) \tag{3.5}$$

and the movie "Braveheart" is represented by

$$q_{Braveheart} = (Braveheart_{toMales}, Braveheart_{serious}) = (4, 5) \tag{3.6}$$

Although both user *Gus* and movie *Braveheart* score high on the "Geared toward males" factor, their "Serious" factors set them apart in terms of compatibility. As can be observed, *Gus* is very fond of "Escapist" (in opposition to "Serious") movies, as expressed by the -4 value on the "Serious/Escapist" axis, while *Braveheart* is definitely a "Serious" movie, since it scores 5 on the "Serious/Escapist" axis. This means that while the "Geared toward males" factor of *Gus* and *Braveheart* suggests *Gus* will like *Braveheart*, the "Serious" factor suggests precisely the opposite. Thus, according to Eq. 3.4, *Gus* preference for *Braveheart* is:

$$r_{Gus,Braveheart} = p_{Gus} \cdot q_{Braveheart} = (4, -4) \cdot (4, 5) = 4 \times 4 + (-4) \times 5 = -4 \tag{3.7}$$

Such prediction indicates that *Braveheart* would not be a good suggestion to present to *Gus*, which is consistent with our intuition on *Gus* and *Braveheart*. This example shows the utility of such vector representation of users and products. This chapter shall thus pursue such representation through matrix factorization techniques.

3.2 Matrix factorization

Matrix factorization comes as an appealing choice for building predictive models in the context of recommender systems, since it can be applied over the user-product ratings matrix (see eq. 3.1) to infer the implicit latent factors needed to obtain the desired user and product representations, as defined in 3.2 and 3.3, respectively. The application of matrix factorization techniques to recommender systems is motivated by the desire of decomposing the ratings matrix into a 2-matrices representation, as in eq. 3.8:

$$R = P \cdot Q^T \Leftrightarrow \begin{bmatrix} r_{1,1} & \cdots & r_{1,n} \\ \vdots & \ddots & \vdots \\ r_{m,1} & \cdots & r_{m,n} \end{bmatrix} = \begin{bmatrix} u_{1,1} & \cdots & u_{1,k} \\ \vdots & \ddots & \vdots \\ u_{m,1} & \cdots & u_{m,k} \end{bmatrix} \cdot \begin{bmatrix} p_{1,1} & \cdots & p_{1,k} \\ \vdots & \ddots & \vdots \\ p_{n,1} & \cdots & p_{n,k} \end{bmatrix}^T \tag{3.8}$$

Here, matrix R is the ratings matrix as defined in 3.1. Each vector (row) p_u of P represents a user u as in 3.2, and each vector (row) q_i of Q represents a product i as in 3.3. Thus, the decomposition suggested by 3.8 directly relates to the desired preference representation defined in 3.4.

3.2.1 Matrix decomposition fundamentals

This subsection followed the discussion presented by Manning et al. in *Introduction to Information Retrieval*, chapter 18 [25].

Before introducing the technique which will be the baseline for our final implementation, let us review the fundamentals of matrix decomposition and linear algebra supporting it. The matrix factorization technique on which our implementation is based is *Singular Value Decomposition* (SVD) - an extension of *symmetric diagonal decomposition*. Before looking into SVD, let us address two other matrix decomposition techniques named *eigen decomposition* and *symmetric diagonal decomposition*, on which SVD is based. Both these techniques apply to square real-valued matrices only and the latter applies to symmetric matrices only. All three mentioned matrix decomposition techniques rely on the concepts of *eigenvalues* and *eigenvectors*. *Eigenvalues* are the values of λ that satisfy

$$R\vec{u} = \lambda\vec{u}, \tag{3.9}$$

where R is a $M \times N$ matrix and $\vec{u}$ is a non-zero M-vector. Accordingly, the M-vectors $\vec{u}$ satisfying 3.9 are called *right eigenvectors* of R and the M vectors $\vec{v}$ satisfying

$$\vec{v}^T R = \lambda\vec{v}^T \tag{3.10}$$

are called the *left eigenvectors* of R. If R is a symmetric matrix, the *eigenvectors* corresponding to distinct *eigenvalues* are orthogonal. The *symmetric diagonal decomposition* is defined by theorem 1.

Theorem 1. *Symmetric diagonalization: Let R be a square, symmetric real-valued $M \times M$ matrix with M linearly independent eigenvectors. Then, there exists a* symmetric diagonal decomposition $R = Q\Lambda Q^T$ *where the columns of Q are the orthogonal and normalized (unit length, real) eigenvectors of R, and Λ is the diagonal matrix whose entries are the eigenvalues of R. Further, all entries of Q are real and $Q^{-1} = Q^T$.*

A schematic representation of such decomposition for a 3×3 matrix is exemplified by eq.

3.11.

$$\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \cdot \begin{bmatrix} \lambda & & \\ & \lambda & \\ & & \lambda \end{bmatrix} \cdot \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \tag{3.11}$$

The Λ diagonal matrix containing the *eigenvalues* λ of R is usually represented by omitting its non-diagonal values - since these are all zeros - as adopted on eq. 3.11.

Definition: It is important to introduce the concept of *rank*. The *rank* of a matrix is the number of linearly independent rows or columns in it. Thus, $rank(R) \leq \min\{M, N\}$. Moreover, the number of non-zero *eigenvalues* of R is at most $rank(R)$.

3.2.2 Singular Value Decomposition

The most popular and widely adopted latent factor approach for recommender systems is *Singular Value Decomposition* (SVD), which is a method for performing matrix factorization. SVD is an extension of the aforementioned matrix decomposition technique *symmetric diagonal decomposition* [25]. Unlike *eigen decomposition* and *symmetric diagonal decomposition*, SVD can be applied to non-square matrices. As one can intuitively assume, if R is a user-product ratings matrix, only in rare cases $M = N$, i.e., the number of users is rarely equal to the number of products. The original SVD formulation is:

$$R = U\Sigma V^T = \begin{bmatrix} u_{11} & \cdots & u_{1m} \\ \vdots & \ddots & \vdots \\ u_{m1} & \cdots & u_{mm} \end{bmatrix} \cdot \begin{bmatrix} \lambda & & \\ & \ddots & \\ & & \lambda \end{bmatrix} \cdot \begin{bmatrix} v_{11} & \cdots & v_{1n} \\ \vdots & \ddots & \vdots \\ v_{n1} & \cdots & v_{nn} \end{bmatrix}^T \tag{3.12}$$

Here, the $\lambda_1, \cdots, \lambda_r$ *eigenvalues* of RR^T are the same as the *eigenvalues* of R^TR. The original $M \times N$ matrix R is decomposed to be represented as the dot product between the matrix U, the diagonal matrix Σ, and the transpose of matrix V. These matrices are:

- U: a $M \times M$ matrix whose columns are the orthogonal *eigenvectors* of RR^T;

- V: a $N \times N$ matrix whose columns are the orthogonal *eigenvectors* of R^TR;

- Σ: a $M \times N$ matrix where the Σ_{ii} positions contain all *singular values* $\sigma_i = \sqrt{\lambda_i}$ for $i \in [1, r]$ with $\lambda_i \geq \lambda_{i+1}$ and $r = rank(R)$, and the remaining Σ_{ij} positions with $i \neq j$ are zeros.

On a square matrix every row/column of Σ would contain a σ_i value but for a non-square matrix this obviously does not happen. For example, if $M > N$, the Σ matrix would be represented as expressed by eq. 3.13.

$$\Sigma = \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.13}$$

As mentioned above, the σ_i values are called *singular values*, the M columns contained in U and the N columns contained in V are called *left singular vectors* and *right singular vectors*, respectively. Eq. 3.14 exemplifies such decomposition for a 3×4 matrix.

$$\begin{bmatrix} 5 & 2 & 4 & 6 \\ 0 & 3 & 1 & 7 \\ 2 & 6 & 3 & 8 \end{bmatrix} = \begin{bmatrix} -0.544 & -0.806 & -0.234 \\ -0.475 & 0.526 & -0.705 \\ -0.691 & 0.273 & 0.669 \end{bmatrix} \cdot \begin{bmatrix} 15.183 & 0.000 & 0.000 & 0.000 \\ 0.000 & 4.392 & 0.000 & 0.000 \\ 0.000 & 0.000 & 1.785 & 0.000 \end{bmatrix} \cdot \begin{bmatrix} -0.270 & -0.439 & -0.311 & -0.798 \\ -0.793 & 0.365 & -0.428 & 0.235 \\ 0.095 & 0.802 & 0.206 & -0.553 \\ -0.254 & -0.467 & -0.713 & 1.000 \end{bmatrix} \tag{3.14}$$

As observed, having $r = rank(R)$, the $N - r$ rightmost columns of matrix Σ containing the *singular values* of R are filled with zeros. Again, this happens when the decomposed matrix is not a square matrix. Similarly, if M would be greater than N, the bottom $M - r$ rows of Σ would be filled with zeros instead. For this reason, the Σ matrix is often presented in a *reduced* or *truncated* $r \times r$ form, containing only the rows and columns that are necessary to represent the singular values. Accordingly, any rows or columns in U and V corresponding to the zero-valued rows or columns in Σ are also left out in the *reduced/truncated* representation of SVD. Additionally, it is important to notice that since $\sigma_i \geq \sigma_{i+1}$, *singular values* are progressively less relevant to (i.e., have smaller impact on) the result of the final matrix product. This is a key fact for the dimensionality reduction discussion introduced in the next subsection.

3.2.3 Low-rank dimensionality reduction

In large-scale recommender systems, the ratings matrix can easily contain many millions of entries which makes it a computationally expensive task to compute and even to store its SVD representation. Thus, reducing the dimensionality of the ratings matrix is useful to tackle this problem. Low-rank dimensionality reduction consists in pursuing an approximation of a matrix while reducing its *rank*. Given a $M \times N$ matrix R the goal is to find an approximation R_k to matrix R whose *rank* is at most k.
Let us revisit the example introduced by eq. 3.14, now represented in its *reduced/truncated* form:

$$R = \begin{bmatrix} 5 & 2 & 4 & 6 \\ 0 & 3 & 1 & 7 \\ 2 & 6 & 3 & 8 \end{bmatrix} = \begin{bmatrix} -0.544 & -0.806 & -0.234 \\ -0.475 & 0.526 & -0.705 \\ -0.691 & 0.273 & 0.669 \end{bmatrix} \cdot \begin{bmatrix} 15.183 & 0.000 & 0.000 \\ 0.000 & 4.392 & 0.000 \\ \mathbf{0.000} & \mathbf{0.000} & \mathbf{1.785} \end{bmatrix} \cdot \begin{bmatrix} -0.270 & -0.439 & -0.311 & -0.798 \\ -0.793 & 0.365 & -0.428 & 0.235 \\ 0.095 & 0.802 & 0.206 & -0.553 \end{bmatrix} \tag{3.15}$$

If one would like to obtain a low-rank approximation of R, an option would be zeroing out the less relevant *singular value* of Σ, in this case the one with value 1.785, thus reducing dimensionality from $rank = 3$ to $rank = 2$, leading to:

$$R_{k=2} = \begin{bmatrix} -0.544 & -0.806 & -0.234 \\ -0.475 & 0.526 & -0.705 \\ -0.691 & 0.273 & 0.669 \end{bmatrix} \cdot \begin{bmatrix} 15.183 & 0.000 & 0.000 \\ 0.000 & 4.392 & 0.000 \\ \mathbf{0.000} & \mathbf{0.000} & \mathbf{0.000} \end{bmatrix} \cdot \begin{bmatrix} -0.270 & -0.439 & -0.311 & -0.798 \\ -0.793 & 0.365 & -0.428 & 0.235 \\ 0.095 & 0.802 & 0.206 & -0.553 \end{bmatrix} \tag{3.16}$$

As we can observe, the resulting matrix $R_{k=2}$ would be a reasonable approximation to R:

$$R_{k=2} = \begin{bmatrix} 5.037 & 2.334 & 4.084 & 5.760 \\ 0.115 & 4.009 & 1.254 & 6.298 \\ 1.882 & 5.043 & 2.750 & 8.654 \end{bmatrix} \approx \begin{bmatrix} 5 & 2 & 4 & 6 \\ 0 & 3 & 1 & 7 \\ 2 & 6 & 3 & 8 \end{bmatrix} = R \tag{3.17}$$

A measure for assessing the quality of the obtained approximation is the *Frobenius norm*, addressed in [25], defined as:

$$\|X\|_F = \sqrt{\sum_{i=1}^{M}\sum_{j=1}^{N} X_{ij}^2} \tag{3.18}$$

Here, $X = R - R_k$, where R is the original full matrix decomposition and R_k is a low-rank approximation to R. The *Frobenius norm* of the low-rank approximation $R_{k=2}$ using low-rank dimensionality reduction method would then be:

$$\|R - R_{k=2}\|_F = \left\| \begin{bmatrix} -0.037 & -0.334 & -0.084 & 0.240 \\ -0.115 & -1.009 & -0.254 & 0.702 \\ 0.118 & 0.957 & 0.250 & -0.654 \end{bmatrix} \right\|_F \approx 1.784 \tag{3.19}$$

The Eckart-Young theorem, revisited in [25](Theorem 18.4), shows that this method of dimensionality reduction results in the matrix of *rank k* with the lowest possible *Frobenius* error.

3.3 A matrix factorization model

As explained in previous sections, SVD allows to obtain low-rank approximations to R by zeroing out the less relevant *singular values* in matrix Σ and recalculating the product $U \cdot \Sigma \cdot V^T$ with the new lower-rank matrix Σ. With the appropriate modifications, SVD seems to be an appealing choice for pursuing the 2-matrices decomposition of the ratings matrix defined by 3.8, leading to the pursued preference representation introduced by 3.4. However, it is vital to acknowledge that original SVD is designed to be performed over a complete matrix. This raises an obstacle when applying the SVD technique to the recommendation systems problem, since the ratings matrix is highly incomplete. Since each user only rates a small portion of the whole set of available products, the ratings matrix tends to be sparsely filled. In fact, the goal is precisely to predict the missing values of the ratings matrix based on the known ones. This fact demands a new perspective over the whole question, in particular over the application of the SVD technique. First of all, the Frobenius norm is no longer suitable to assess the quality of an approximation to the ratings matrix, since there are many unknown values which cannot be treated as zeros. Treating the unknown values as zeros would lead to severe inaccuracies, since assuming ratings to be zero is merely blind guessing. Thus, the Frobenius norm is now modified by taking into account only the known ratings, leading to eq. 3.20:

$$\|X\|_F = \sqrt{\sum_{r_{ui} \in R} (r_{ui} - p_u \cdot q_i^T)^2} \tag{3.20}$$

The second question that must be addressed is related to the fact that, accordingly with eq. 3.4 it is intended to obtain a 2-matrices decomposition of R instead of a 3-matrices decomposition as provided by the original SVD technique. As mentioned in subsection 3.2.2, SVD is performed over a matrix R to obtain the decomposition $R = U\Sigma V^T$. Since our goal is to obtain a decomposition that results in the product of a users matrix and a products matrix, the need to obtain a $R = P \cdot Q^T$ decomposition leads to the factorization of the Σ matrix into a product of its square roots, so that its square root values get symmetrically blended into the U and V matrices to produce the pursued users P and products Q matrices. Eq. 3.21 formalizes this transformation.

$$R = U\Sigma V^T = U\sqrt{\Sigma} \cdot \sqrt{\Sigma} V^T = P \cdot Q^T \tag{3.21}$$

Here, $P = U \cdot \sqrt{\Sigma}$ and $Q = \sqrt{\Sigma} \cdot V$.

Thus, the SVD analogy applied to recommender systems results in the decomposition of the $M \times N$ ratings matrix in two matrices representing the M users and the N products, with dimensions $M \times k$ and $N \times k$, respectively, where k is the desired number of dimensions (*rank*) for low-rank reduction:

$$R = \begin{bmatrix} r_{1,1} & \cdots & r_{1,n} \\ \vdots & \ddots & \vdots \\ r_{m,1} & \cdots & r_{m,n} \end{bmatrix} = \begin{bmatrix} u_{1,1} & \cdots & u_{1,k} \\ \vdots & \ddots & \vdots \\ u_{m,1} & \cdots & u_{m,k} \end{bmatrix} \cdot \begin{bmatrix} p_{1,1} & \cdots & p_{1,k} \\ \vdots & \ddots & \vdots \\ p_{n,1} & \cdots & p_{n,k} \end{bmatrix}^T \tag{3.22}$$

These k dimensions turn out to be implicit latent factors common to both users and products, allowing for preference estimation as described in eq. 3.4. This decomposition is intended to be equivalent to original SVD, but having the *singular values* automatically blended into the user and product matrices, allowing for a 2-matrices representation, and pursuing a low-rank approximation by setting a maximum value of k, instead of aiming for an exact decomposition.

3.4 Learning the factorization model

Simon Funk [8] suggested an efficient solution to learn the factorization model which has been widely adopted by other researchers [22, 40, 29] and consists in decomposing the ratings matrix by taking into account the set of known ratings only. Hence, the matrix decomposition process now pursues the minimization of the difference (henceforth referred to as error) between the known ratings present on the original ratings matrix and their decomposed representation. This is done with the hope that by the end of the process it will be possible to make predictions of unknown ratings by generalizing what has been learned from processing the known ones. Thus, the goal of the learning process is to solve the least-squares problem defined by eq. 3.23.

$$[P, Q] = \underset{p_u, q_i}{\arg\min} \sum_{r_{ui} \in R} (r_{ui} - p_u \cdot q_i^T)^2 \tag{3.23}$$

In most recommender system implementations, as well as on the Netflix Prize and the Yahoo KDD Cup 2011 competitions, the measure used to assess prediction error is the Root Mean Squared Error (RMSE), which is an error measure that puts emphasis on higher errors, rendering it as an ideal measure for applications where maximum accuracy is pursued. The RMSE is closely related to the error measure previously suggested in eq. 3.20, and gathers all calculated errors as described by eq. 3.24:

$$RMSE = \sqrt{\frac{\sum_{r_{ui} \in R} (r_{ui} - \hat{r}_{ui})^2}{|R|}} \tag{3.24}$$

Here, r_{ui} is the real rating given by user u to product i within the set of ratings R and $\hat{r}_{ui}$ is its prediction, obtained through $\hat{r}_{ui} = p_u \cdot q_i^T$. Lower RMSE values correspond to better predictive performance. When the learning process is finished, the user and product factor matrices can be used to predict user ratings.

3.4.1 Iterative learning

The SVD-based model that is being discussed comprises the minimization of a least-squares problem, previously introduced by eq. 3.23. Learning this model can be through several different algorithms. However, due to the large amount of data involved there is the need to choose a path that can produce results within a reasonable time. Alternate Least Squares (ALS) and Stochastic Gradient Descent (SGD) are the most widely adopted learning methods for recommender systems. S.Funk [8] suggested a SGD approach, where this minimization is obtained through an iterative process in which each step gradually shapes the model. In that sense, the learning algorithm of user and product factor vectors comprises looping through all known ratings several times, modifying the factor vector values towards a perfect fit of the observed data (the set of known ratings). Each loop through the entire ratings set will be henceforth referred to as *step*. The more steps the algorithm takes, the closer the model will be to perfectly fitting the observed data. Algorithm 3.4.1 describes in pseudo-code the idea of the iterative process carried out to build a model that perfectly fits the training dataset.

However, a perfect fit is not desired, for it would lead to a situation that is referred to in

Algorithm 3.4.1 Iterative learning process to a perfect fit

```
$predictedTrainRatings \leftarrow predict(trainingSet)$
$trainRmse \leftarrow computeRMSE(predictedTrainRatings, realTrainRatings)$
while $trainRmse > 0$ do
    $model \leftarrow updateAccordingToError(model, trainRmse)$
    $predictedTrainRatings \leftarrow predict(trainingSet)$
    $trainRmse \leftarrow computeRMSE(predictedTrainRatings, realTrainRatings)$
end while
```

literature as *over-fitting*. It is intended that the model captures users' past rating patterns with a good degree of precision, but there is the need to assure that the model preserves enough generality to predict future ratings. Thus, arrangements are in need to determine how many steps the algorithm should take until the model reaches its best predictive potential.

Cross-validation: at the end of each step taken over the training set, a validation set containing additional known ratings that have not been used to train the model is used to control and evaluate the progress of the learning process. The systems tries to predict all ratings contained in the validation set based on the predictive model built until that moment and calculates the RMSE of its predictions. This intermediate evaluation is repeated at the end of each step to assess if the predictive accuracy of the model (evaluated over the validation set) is improving or stabilizing. When the model's predictive accuracy no longer seems to improve, the learning process shall be interrupted and the model is expected to be at its best to predict new unknown ratings. The final tests are then performed over a test set containing known ratings that have not been used neither for training nor for validation. Ideally, the test set is used only once and the results obtained from testing the model over this test set are final and reflect the quality of the model. The three mentioned sets - training set, validation set and test set - are obtained by splitting the initial dataset into three subsets beforehand. The splitting portions may vary over algorithms but, as an example, a reasonable split would be 75%-10%-15%. This method of splitting the dataset in such way to train, validate and test the model is called *cross-validation*.
By using cross-validation, the iterative learning process would be modified into alg. 3.4.2 (modifications are highlighted in blue color).

3.4.2 Regularization

As the algorithm runs through the set of known ratings R only, the resulting factor vectors will converge to a perfect fitting of the observed data, which results in an undesirable outcome previously referred to as *over-fitting*. Again, over-fitting is undesirable because the purpose of matrix factorization is to build a model that is able to predict the missing rating values, while observing the known rating values. Therefore, while gradually fitting the observed data, there is the need to preserve generality by avoiding over-fitting.

Algorithm 3.4.2 Iterative learning process with cross-validation

```
$predictedTrainRatings \leftarrow predict(trainingSet)$
$trainRmse \leftarrow computeRMSE(predictedTrainRatings, realTrainRatings)$
$previousValidRmse \leftarrow \infty$
$predictedValRatings \leftarrow predict(validationSet)$
$currValidRmse \leftarrow computeRMSE(predictedValidRatings, realValidRatings)$
while $currValidRmse < previousValidRmse$ do
    $model \leftarrow updateAccordingToError(model, trainRmse)$
    $predictedTrainRatings \leftarrow predict(trainingSet)$
    $trainRmse \leftarrow computeRMSE(predictedTrainRatings, realTrainRatings)$
    $previousValidRmse \leftarrow currValidRmse$
    $predictedValidRatings \leftarrow predict(validationSet)$
    $currValidRmse \leftarrow computeRMSE(predictedValidRatings, realValidRatings)$
end while
$predictedTestRatings \leftarrow predict(testSet)$
$testRmse \leftarrow computeRMSE(predictedTestRatings, realTestRatings)$
```

Some solutions have been suggested to tackle this problem, mostly by adding a regularization term to the function intended to minimize. These modifications lead to a modification to the least-squares problem previously introduced by eq. 3.23, now defined by eq. 3.25.

$$[P, Q] = \underset{p_u, q_i}{\arg\min} \sum_{r_{ui} \in R} (r_{ui} - p_u \cdot q_i^T)^2 + \lambda \cdot (\|p_u\|^2 + \|q_i\|^2), \tag{3.25}$$

where λ is a constant defining the extent of regularization, usually chosen by cross-validation.

The goal of this regularization term is to penalize the magnitudes of large factor vector values. This way, the algorithm does not strive recklessly to fit the observed data as if this data would be the unequivocal and absolute source of all knowledge about the users and products involved, thus introducing the necessary flexibility into the prediction of what is unknown. Another regularization measure consists in having a control data set against which the predictive ability of the model can be tested throughout all the steps. Such control is provided by the cross-validation process introduced earlier, since what the algorithm learns by observing the training set only is recurrently tested against the validation set. What is expected to happen is that, while the prediction error for the ratings on the training set will indefinitely decrease over the steps, the prediction error for the ratings on the validation set will at some point stabilize, right before it starts increasing into an over-fitting situation. Thus, the best moment to interrupt the learning process it that when the validation error stabilizes, before it starts increasing.

3.4.3 Stochastic gradient descent

As mentioned previously, Simon Funk [8] popularized a method for learning the factor vectors using *stochastic gradient descent*. This method comprises modifying the values of the factor vectors according to the associated prediction error, defined as

$$e_{ui} = r_{ui} - \hat{r}_{ui} \tag{3.26}$$

where $\hat{r}_{ui}$ is the rating prediction for user u to product i. Predictions $\hat{r}_{ui}$ are calculated according to the following rule:

$$\hat{r}_{ui} = p_u \cdot q_i^T \tag{3.27}$$

Again, p_u and q_i^T are the factor vectors associated with user u and product i, respectively. The algorithm iterates through every known rating r_{ui} and updates the factor vectors according to the following rules:

$$\begin{aligned} p_u &\leftarrow p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda \cdot p_u) \\ q_i &\leftarrow q_i + \gamma \cdot (e_{ui} \cdot p_u - \lambda \cdot q_i) \end{aligned} \tag{3.28}$$

The λ parameter is a regularization constant to control over-fitting and parameter γ is the learning rate at which these modifications should occur. Both these parameters are usually chosen by cross-validation. This algorithm involves processing each latent factor separately, allowing for each factor to improve prediction accuracy after the previous factors have given their best contribution. The implementation presented on this thesis is inspired by Simon Funk's suggestion, and its main learning loop can be described in pseudo-code as expressed by alg. 3.4.3. The full sequence of steps to compute RMSE values described in previous algorithms was now omitted for simplicity.

Algorithm 3.4.3 Stochastic gradient descent learning algorithm

```
while validation RMSE decreases do
  for all latent factor $k$ in $K$ do
    for all $r_{ui}$ in trainingSet do
      $\hat{r}_{ui} \leftarrow 0$
      for all latent factor $f$ in $K$ do
        $\hat{r}_{ui} \leftarrow \hat{r}_{ui} + P_{uf} \cdot Q_{if}$
      end for
      $err \leftarrow r_{ui} - \hat{r}_{ui}$
      $P_{uk} \leftarrow P_{uk} + \gamma \cdot (err \cdot Q_{ik} - \lambda \cdot P_{uk})$
      $Q_{ik} \leftarrow Q_{ik} + \gamma \cdot (err \cdot P_{uk} - \lambda \cdot Q_{ik})$
    end for
  end for
  compute validation RMSE
end while
```

Here, K is the number of latent factors used by the algorithm, $\hat{r}_{ui}$ is the rating prediction for user u and product i, and P and Q are the user-factor and product-factor matrices, respectively.

3.5 Implementation details

To implement the above described model, the first step is to define data structures to store the involved variables. On this project, the implementation was carried on in C++ language, but generality will be preserved in the following structures and algorithm descriptions:

- **trainRatings**: a structure to store the training set of ratings, positive and usually integer values;
- **valRatings**: a structure to store the validation set of ratings, positive and usually integer values;
- **testRatings**: a structure to store the test set of ratings, positive and usually integer values;
- **uSvd[k,u]**: a 2-dimensional array to store all user factors, which are real values;
- **pSvd[k,p]**: a 2-dimensional array to store all product factors, which are real values;
- **lrate**$\{index\}$: real variables to store the different learning rates;
- **lambda**$\{index\}$: real variables to store the different normalization constants;

For the learning process, the factor vector values must not be initialized as zeros, as one can easily infer from the learning rules definition (3.28), so a reasonable option is to initialize these with small values, such as 0.01.

3.5.1 Accuracy optimization

Maximizing the accuracy of predictions is an important issue within the scope of recommender systems. To carry out such task one must take into account the context of the system, as well as the available input data, and exhaustively tune the model to meet its purpose. There are countless ways of gaining predictive accuracy, as the slightest change in the algorithm may have a strong impact on the final outcome. A few simple accuracy enhancing measures were taken and will be described in this section.

3.5.1.1 Clipping

Prediction accuracy can be further improved with some modifications to the algorithm. A straightforward modification that can be done and that certainly contributes to better results is clipping the final prediction to assure it always falls into the range of possible rating values. So, if the rating scale ranges from 0 to 100, the new prediction function is modified by returning a *clipped* result through an auxiliary clipping function that truncates rating predictions that fall outside these limits. This improvement assures there will only be produced rating predictions that make sense within the possible rating range.

3.5.1.2 Parameter tuning

Additionally, accuracy can be substantially improved by parameter tuning. Parameter tuning consists in finding the combination of parameters that yields the best results in terms of prediction accuracy. In the current model, parameters are the learning rates for the factor vectors, as well as the normalization constants and the minimum learning rate threshold. This is often a cumbersome task, especially when there are many free parameters involved. However, to obtain optimal performance, parameters should be exhaustively tuned according to the context and dataset to which the recommender system is applied. For the Netflix Prize winner solution an *Automatic Parameter Tuning* (APT) mechanism was implemented, as described in [39]. In our implementation a wide range of learning rates was tested for all models to assess which were the values that yielded the best results. All modifications suggested in this section will be further discussed in the evaluation section.

3.5.2 Speeding up the algorithm

For large datasets, i.e. datasets containing many millions of ratings given by hundreds of thousands of users to hundreds of thousands of products, the described algorithm can become computationally expensive. Thus, when computational resources are limited, arrangements need to be made to minimize the complexity of the algorithm.

3.5.2.1 Adaptive learning rate

Additional reduction of computation time and even some accuracy improvement can be achieved by having an adaptive learning rate, represented by *lrate* in the pseudo-code and γ in eq. (4.3), to the progress of the learning process, instead of having it static. The goal of such modification is to have the algorithm updating the factor vectors with smaller increments as the model gets closer to fitting the data. This would allow for faster - and more *greedy* - learning at the beginning without losing precision, since the learning rate gets smaller over iterations, allowing for a careful convergence towards the final model. Ideally, this modification would lead to reducing the steps needed to obtain a solution. A simple way of obtaining such effect is by scaling the learning rate by 0.9 at each step, as suggested by Y. Koren [18]. This way, the *lrate* decay does not directly result from the observed error decrease, but it sets on the assumption that the prediction error decreases at each step, which is a fair assumption considering the nature of the model. It is easy to observe that after some steps the learning rate will be very low, since it is converging to zero. This is not ideal because after a while the factor vector values will not be modified and convergence stops prematurely. To address this issue a threshold was defined, setting the minimum learning rate to 0.0007. This means that once the learning rate decays to 0.0007, its value will no longer be decreased throughout steps. The value for the threshold was not exhaustively tested, but it is intended to be lower than the smallest learning rate tested, which is 0.001, so that the algorithm is allowed to take steps towards the solution that at some point are smaller than those taken if the learning rate was not adaptive.

3.5.2.2 Storing factor residuals

By analysing the main learning loop described in alg. 3.4.3, one can easily identify the bottleneck that results from having to iterate through all K factors to get each rating prediction, which is done once per training rating, per factor, per step, associated to the computation of rating predictions. This leads to a complexity of $O(K^2 \times R)$ per step, where R is the number of ratings on the training set. A way of minimizing the impact of this bottleneck is by saving the sum of all calculated $P_{uk} \times Q_{ik}$ (with $k \in K$) factor products into a data structure *trainResiduals* so that when computing each factor, the sum of all previously computed factors is available for immediate access, avoiding unnecessary loops through all K factors. The computation of rating predictions described in alg. 3.4.3 is then improved, modifying each step of the learning loop into alg. 3.5.1.

Algorithm 3.5.1 Learning step of SGD with training residuals structure

```
for all latent factor $k$ in $K$ do
  for all $r_{ui}$ in trainingSet do
    $\hat{r}_{ui} \leftarrow trainResiduals_{ui} + P_{uk} \cdot Q_{ik}$
    $err \leftarrow r_{ui} - \hat{r}_{ui}$
    $P_{uk} \leftarrow P_{uk} + \gamma \cdot (err \cdot Q_{ik} - \lambda \cdot P_{uk})$
    $Q_{ik} \leftarrow Q_{ik} + \gamma \cdot (err \cdot P_{uk} - \lambda \cdot Q_{ik})$
  end for
end for
```

This improved way of computing rating predictions uses the new structure *trainResiduals*, which stores the sum of all factor products except the one being computed at the moment. This solution involves updating the residuals before and after processing each factor, which adds an extra complexity overhead. Before entering a new factor loop, the *trainResiduals* structure must subtract from its saved values the contribution given by previous computations of the factor which will be processed on the new loop. This needs to be done before the new loop starts because once inside the loop, the current factor values will be constantly modified at each rating computation. Accordingly, after each factor loop these residuals must be updated as well, by adding to its values the contributions of the fresh computed factor. After this is done, the algorithm is ready to move on to the next factor. The main learning loop of the algorithm is then modified into alg. 3.5.2.
Updating the residuals involves looping through all values in the residuals structure. Since there is the need to store one residual value for each rating on the training set, this represents looping through R values, according to the notation introduced in the complexity discussion. Despite the overhead introduced by the residuals update operations, this solution offers great improvement in terms of complexity reduction for most values

Algorithm 3.5.2 SGD main learning loop with training residuals structure

```
while validation RMSE decreases do
  for all latent factor $k$ in $K$ do
    for all $r_{ui}$ in trainingSet do
      $trainResiduals_{ui} \leftarrow trainResiduals_{ui} - P_{uk} \cdot Q_{ik}$
    end for
    for all $r_{ui}$ in trainingSet do
      $\hat{r}_{ui} \leftarrow trainResiduals_{ui} + P_{uk} \cdot Q_{ik}$
      $err \leftarrow r_{ui} - \hat{r}_{ui}$
      $P_{uk} \leftarrow P_{uk} + \gamma \cdot (err \cdot Q_{ik} - \lambda \cdot P_{uk})$
      $Q_{ik} \leftarrow Q_{ik} + \gamma \cdot (err \cdot P_{uk} - \lambda \cdot Q_{ik})$
    end for
    for all $r_{ui}$ in trainingSet do
      $trainResiduals_{ui} \leftarrow trainResiduals_{ui} + P_{uk} \cdot Q_{ik}$
    end for
  end for
  compute validation RMSE
end while
```

of K. The complexity of this solution is $O(K \times (R + 2 \times R)) \Leftrightarrow O(3K \times R)$ which is substantially lower than $O(K^2 \times R)$ for $K > 1$. Moreover, the benefit brought by this solution increases with the number of K factors.

3.5.3 Stochastic parallel optimization

Whenever multiple processors are available, parallelizing the algorithm can bring great improvement in terms of processing speed. Usually, it is a better practice to use multi-processor parallelization when the learning algorithm allows for independent computation of variables, such as the *alternating least squares* (ALS) algorithm. The main idea of ALS is turning the prediction error minimization problem into a quadratic problem by alternately fixing the user factors and the product factors, thus allowing for an independent manipulation of each factor value. ALS has been applied in the context of recommender systems in several implementations [12, 4], as it is more suitable for full matrices and parallel processing than the *Stochastic Gradient Descent* algorithm. However, we did not use ALS, and opted to use SGD for one main reason: SGD is simpler and usually faster than ALS for highly sparsely filled matrices, which is the case when the algorithm learns based on explicit feedback only. The problem associated with parallelizing a gradient descent algorithm is that more than one modification to the same factor value may occur simultaneously, leading to conflicting learning operations. As an example, let us consider a training set where ratings are sequenced randomly, i.e, with no specific user nor product order, 4 processors are available and the learning of users and products factor values is parallelized by simply splitting the training set into 4 balanced portions and assigning each to a different processor. What could happen in this case is exemplified by

Processor 1	Processor 2	Processor 3	Processor 4
$r_{\mathbf{1},3}$	$r_{\mathbf{1},7}$	$r_{4,2}$	$r_{7,6}$
$r_{4,\mathbf{3}}$	$r_{3,\mathbf{3}}$	$r_{7,\mathbf{3}}$	$r_{8,1}$

the following scheme:

The above scheme intends to illustrate a situation where more than one modification to the same factor values occur simultaneously. Let us assume that the factor being processed is $k = 5$. Ratings r_{ui} on the same line of the scheme are processed in parallel fashion by 4 processors. As described in alg. 3.4.3, whenever a training rating r_{ui} is processed, the corresponding P_{uk} and Q_{ik} values are modified for each factor k. In the situation illustrated by the scheme, $P_{1,5}$ is being modified by 2 processors at the same time on the first line, and on the second line $Q_{3,5}$ is being modified by 3 processors at the same time. Whenever such situations happen, inconsistencies in the learning process come as a consequence, which is undesirable. However, this problem can be partially fixed by sorting the training set by users and making sure that all ratings given by the same user are processed by the same processor. If instead of randomly splitting the training set into 4 portions, we split it by assigning $1/4$ of the total users to each processor, then simultaneous modifications to the same user factor values P_{uk} would be avoided. A more intelligent way of splitting the training set by users was implemented though, pursuing a balanced division of ratings by processor, regardless of how many users are assigned to each processor, thus maximizing efficiency. With this improvement, the initial problem is narrowed down to dealing with simultaneous modifications to product factors Q_{ik}. However, one can intuitively come to the conclusion that if the probability of a product being rated in common by 2 users is small enough, the risk of inconsistencies is negligible and is often worth to trade-off for efficiency. The test session documented on the next section can provide conclusive answers on this matter. Following the same line of thought, parallelization can also be applied to update user and product biases. Additionally, the update of factor residuals, as well as the computation RMSEs can be parallelized with no risk of inconsistencies, since these operations do not modify any model variables.

3.6 Evaluation

In this section, the previously introduced SVD-based model will be tested and analysed, according to the described implementation. The proposed accuracy and speed-up enhancement measures will also be put to the test to assess their relevance. This model comprising SVD-based matrix factorization will be henceforth referred to as *plain-SVD*.

3.6.1 Datasets

All tests were performed on the Yahoo! KDD Cup 2011 dataset. The dataset provided by Yahoo! contains $262,810,175$ ratings given by $1,000,990$ users to $624,961$ products and comes split into 3 subsets: training, validation and test. These subsets were provided by Yahoo! so that contestants could train their prediction algorithms with the training and validation sets, and then submit their predictions for the ratings contained in the test set. During the Yahoo! KDD Cup 2011 competition the rating values for the test subset were not revealed, since the goal was to predict these. When Yahoo! KDD Cup 2011 finally finished, these rating values were released and enabling us to use them for the evaluation session and discussion presented on this thesis. The Yahoo! KDD Cup 2011 dataset statistics are:

- **Total of ratings:** $262,810,175$ ratings
- **Training set:** $252,800,275$ (96.19%) ratings
- **Validation set:** $4,003,960$ (1.52%) ratings
- **Test set:** $6,005,940$ (2.29%) ratings
- **Nr. of users:** $1,000,990$
- **Nr. of products:** $624,961$

Products are musical items which can be of 4 different categories: genres, artists, albums or tracks. There are additional files provided by Yahoo! establishing relations between the products (linking tracks to the corresponding albums, artists and genres, for example) which can be directly used to help building the model. However, we allowed the matrix factorization algorithm to capture these relations on its own. Both training, validation and test sets are ordered by users, which are unequivocally identified by a number and sequenced starting from zero. Products are also identified by a number. The validation set contains exactly 4 ratings per user, the test set contains exactly 6 ratings per user and the training set contains at least 10 ratings per user. Set entries are grouped by user and have the following format:

```
<UsedId>|<#UserRatings>\n
```

Each of the next `<#UserRatings>` lines describes a single rating by `<UserId>`, sorted in chronological order. Rating line format is:

```
<ItemId>\t<Score>\t<Date>\t<Time>\n
```

Rating values range from 0 to 100 and are integer. However, for intermediate calculations, ratings were scaled down by a factor of $\frac{1}{20}$ so that their range is reduced to $[0, 5]$. This scaling operation solved overflow problems that were detected during the learning process. The final outcome is scaled back to it original $[0, 100]$ range. The temporal information provided for each rating was not used at this stage but will be addressed in the next chapter.

3.6.2 Experiment design

Due to the magnitude of the Yahoo! dataset and the computational resources it demands to process, only a portion of it was used at a time in each test, so that more results could be presented within the available time for the test session. All experiments on the test session were performed over a $35,000$-user portion of the dataset. For some of the experiments, a 3-fold cross-validation evaluation was performed, meaning that 3 randomly picked disjoint $35,000$-user portions of the dataset were used for 3 independent experiments, and the presented results are an average of these 3 experiments. The choice of splitting the dataset by users was made to assure that all available data about each user is used to make predictions for that user, and because the Yahoo! dataset already comes grouped by user, which simplifies the splitting process. In all tests, different numbers of latent factors were used: 20, 50, 100 and 200, as well as 8-core parallelization by default. The tests performed can be then divided as follows:

- **Learning progress**: For the plain-SVD model, the learning progress over the steps was monitored and analysed to confirm that it meets the expected behaviour.
- **Parameter tuning**: For the plain-SVD model, tests were performed to find which is the learning rate (the γ parameter in 3.28) that provides best results. In the literature several values were suggested for this parameter, ranging from 0.001 to 0.008. Hence, tests were performed for the range of $[0.001, 0.020]$ with 0.001 increments between tested values. The regularization parameter λ was fixed at 0.015.
- **Adaptive learning rate**: The plain-SVD model was tested with and without using adaptive learning rate and results were compared.
- **Model performance**: The plain-SVD model was tested with the best factor learning rate previously assessed and its results were analysed.
- **Parallelization**: The learning process of the plain-SVD model was tested with different levels of parallelization (single-core, 2-core, 4-core and 8-core) and respective performances were analysed and compared in terms of speed and accuracy.

3.6.3 Results and discussion

3.6.3.1 Learning progress

The chart presented by fig. 3.2 shows the behaviour of the learning process for the plain-SVD model for different numbers of latent factors, using a $35,000$-user portion of the dataset, a learning rate of 0.015 and a regularization parameter value of 0.015.

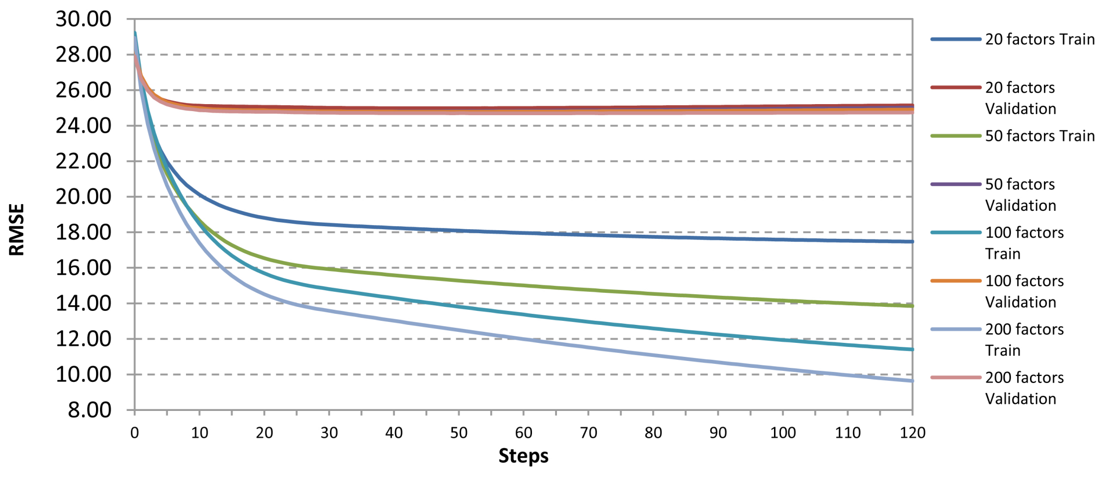


Figure 3.2: Plain-SVD model learning behaviour over 120 steps with different numbers of latent factors

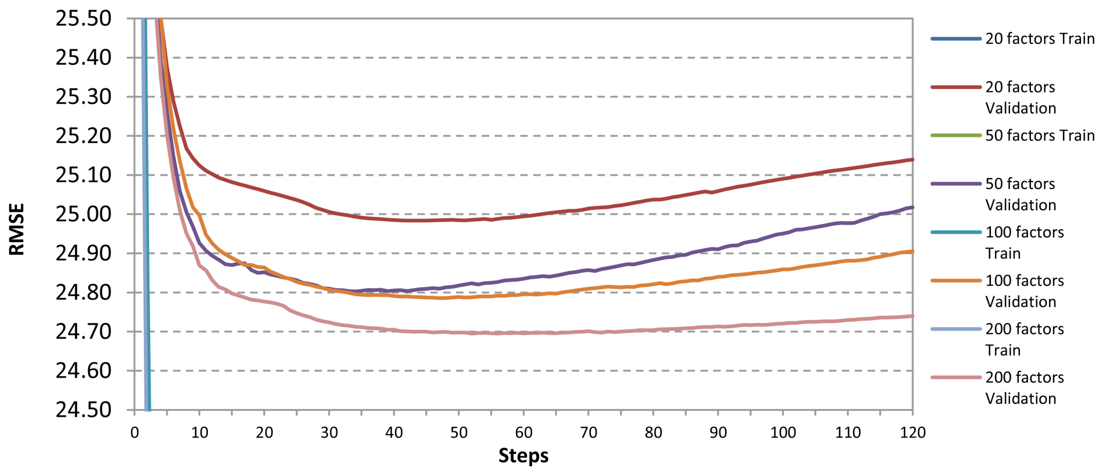


Figure 3.3: Plain-SVD model learning behaviour over 120 steps with different numbers of latent factors (zoomed)

A high learning rate of 0.015 was chosen for all models in this experiment so that convergence - and subsequent over-fitting - would occur sooner and learning behaviours would be more easily compared across models with different numbers of latent factors. The learning behaviour illustrated by fig. 3.2 shows that while the training error continuously decreases over the steps, the validation error stabilizes after some point, which occurs for all different numbers of factors tested at around step 55. Such behaviour meets the expectations and the moment when the validation error stabilizes corresponds to the moment when the model starts to over-fit the training data. A zoomed version of the chart is also presented, illustrated by fig. 3.3, in which one can also observe that building the model with a higher number of latent factors tends to yield better results, converging after approximately the same number of steps.

3.6.3.2 Parameter tuning

After analysing the learning behaviour, tests were performed to assess which were the best learning rates for the plain-SVD model. The results in terms of RMSE on the test set are presented on fig. 3.4. It can be observed that the plain-SVD model provides different

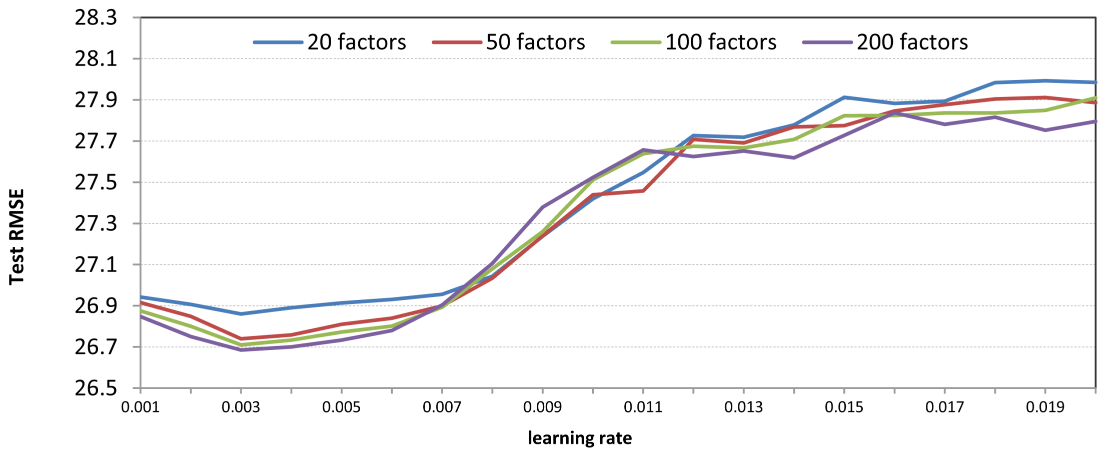


Figure 3.4: Plain-SVD model performance with different learning rates

results for different learning rates. The optimality of learning rates depends not only on the dataset but also on the number of latent factors and the model, as well as all other free variables involved, such as the regularization parameters. In our case, a learning rate of 0.003 seems to be the one yielding the best results for all different numbers of latent factors.

3.6.3.3 Adaptive learning rate

The following chart, illustrated by fig. 3.5, shows the results of experiments made with the plain-SVD model using adaptive and non-adaptive learning rates for several learning rate values. The improvements we aimed to achieve by using an adaptive learning rate were related mostly with computation time, but also with accuracy. We expected to find the right combination between a learning rate and a progressive decay rate that would allow the algorithm to have a fast learning at the beginning but a slower and more careful learning when getting closer to the final solution.
For the plain-SVD model, this modification did not bring significant improvements, since the best RMSE results obtained on the test set were 26.6851 using adaptive learning rate with a learning rate of 0.003, and 26.6870 using non-adaptive learning rate with a learning rate of 0.001. Moreover, the number of steps taken was 150 (the maximum allowed) in

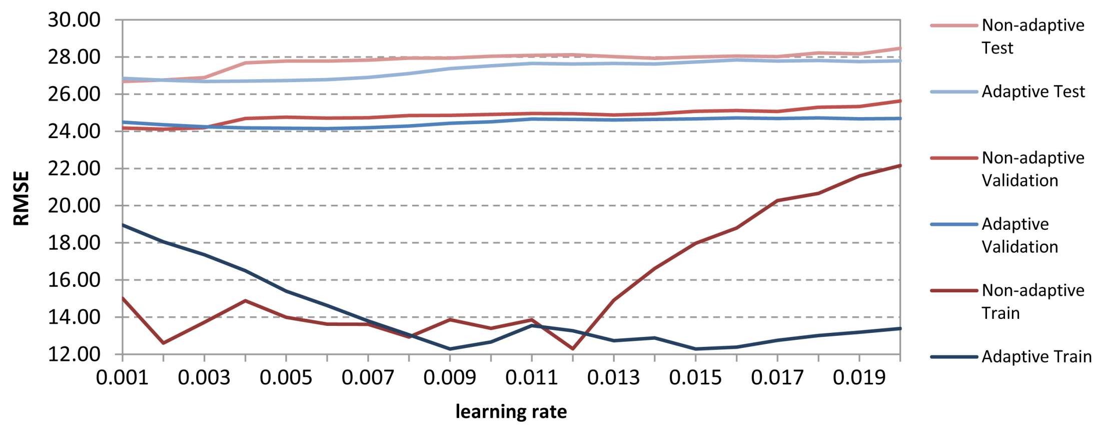


Figure 3.5: Performance comparison between adaptive and non-adaptive learning rates for the plain-SVD model using 200 factors

both cases, rendering both as similarly efficient options.

Although many different learning rates were tried, both the minimum learning rate set for the adaptive learning rate approach (set to 0.0007) and the progressive decay rate (set to 0.9) were not exhaustively tested. Additional experiments testing different combinations of learning rate, min. learning rate and progressive decay rate would be useful to further test this option. In future works, a more exhaustive experimentation session with such intent shall be taken.

3.6.3.4 Model performance

Fig. 3.6 shows the performance of plain-SVD model trained with the best learning rates discovered earlier, using different numbers of latent factors. The training algorithm was allowed to run for 150 steps and the results presented are an average of a 3-fold experiment with 3 disjoint $35,000$-user portions of the dataset. As a reference value, the lowest RMSE I obtained in the Yahoo! KDD Cup 2011 competition was 24.5848. This result ranked me $85th$ among 1287 active teams. It is important to add that this result was obtained with my best model (one that extends the plain-SVD and will be presented in the next chapter) using 200 latent factors while computing over the whole dataset ($1,000,990$ users) and by using both training and validation sets to train the model, while in the evaluation and discussion presented in this thesis only a $35,000$-user portion was used and the training was performed through cross-validation, i.e., using the training set to train the model and the validation set only to validate and monitor its learning process. Nonetheless, let us keep this 24.5848 RMSE value in mind as an accuracy reference. Additionally, the lowest RMSE obtained by the winning team was 21.0147.

The results obtained from this experiment show that using a higher number of latent factors increases the model's predictive accuracy. Moreover, the best result presented,

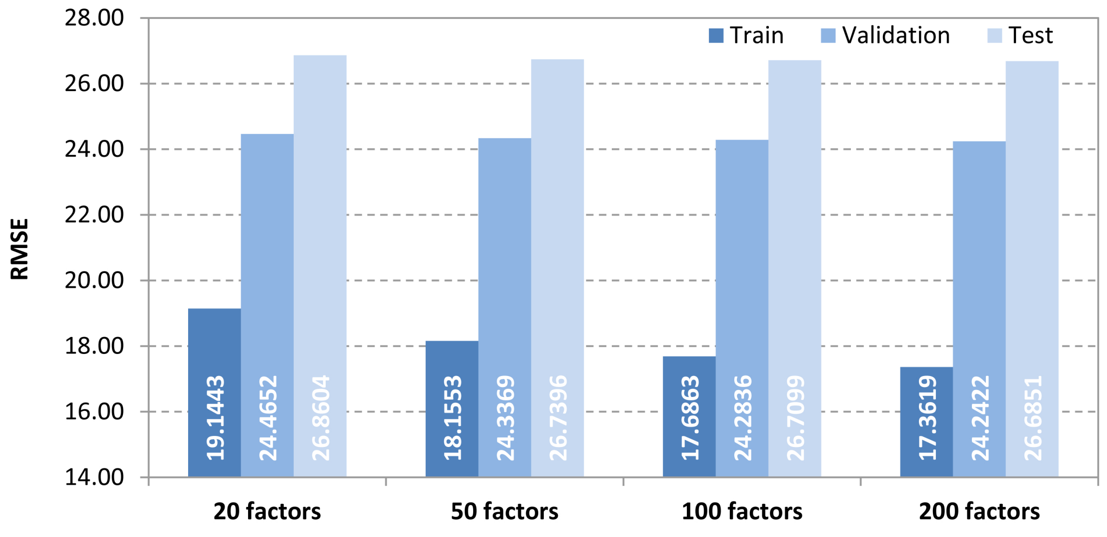


Figure 3.6: Performance of plain-SVD model with different numbers of latent factors

obtained using 200 latent factors, corresponds to a 26.6851 RMSE which is an encouraging value considering our 24.5848 RMSE reference.

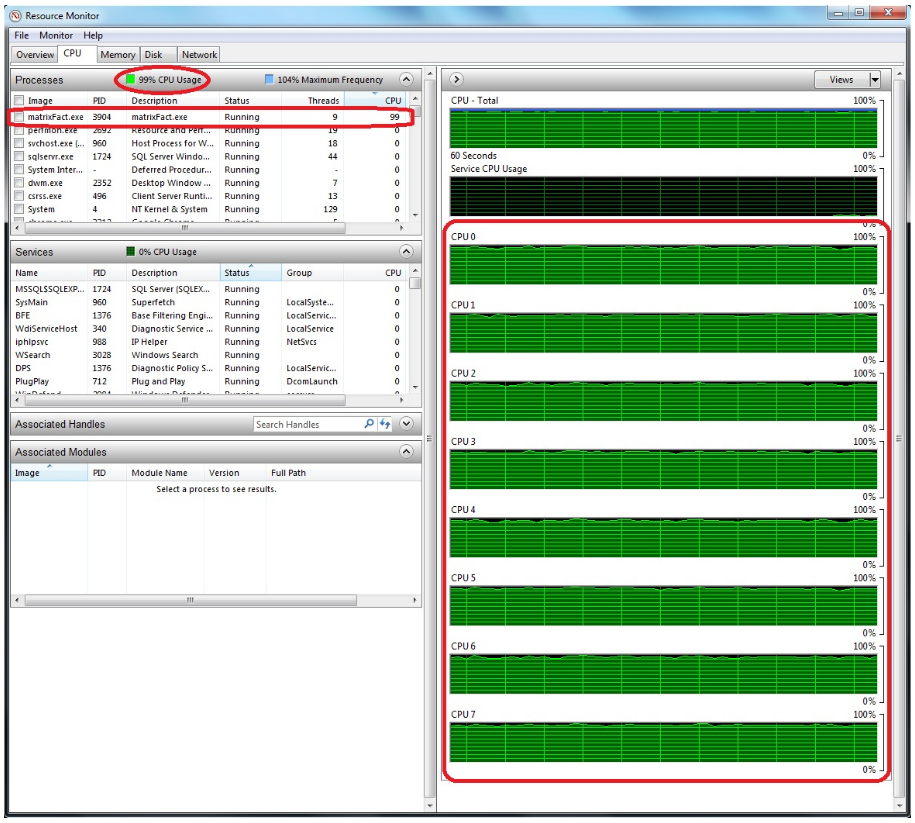
3.6.3.5 Parallelization

Finally, tests were performed to compare the learning performances with four different levels of paralellization: 1-core, 2-core, 4-core and 8-core. Before analysing the results of such tests, is it important to verify whether splitting the training subset by groups of users in the "intelligent way" suggested in 3.5.3 was a good option or not. To confirm the viability of such option, one must verify if the available computational resources are being efficiently used when the algorithm is running. The *resources monitor* of the operative system was of great utility for this task, as fig. 3.7 shows.

Figure 3.7 shows that all available processors are being efficiently used, working at nearly 100% of their full capacity. From this observation, one can conclude that the split by users as proposed is a viable option.

The parallelization tests to assess the plain-SVD model performance were performed with a learning rate of 0.015 and the learning process was allowed to run for 30 steps only. Such high learning rate value was used to amplify the eventual damage in terms of predictive accuracy brought by parallelization-related inconsistencies, when these occur, and 30 learning steps are enough to compare model performances. Moreover, the plain-SVD model was tested with 100 factors.

The results of parallelization tests are illustrated by fig. 3.8. As expected, the processing time per step is lower when more processors are used. As for the RMSE values obtained, these are somewhat surprising. The fact that there is no significant accuracy loss when



Processes — 🟩 99% CPU Usage — 104% Maximum Frequency

Image	PID	Description	Status	Threads	CPU
matrixFact.exe	3904	matrixFact.exe	Running	9	99
perfmon.exe	2692	Resource and Perf...	Running	19	0
svchost.exe (...	960	Host Process for W...	Running	18	0
sqlservr.exe	1724	SQL Server Windo...	Running	44	0
System Inter...	-	Deferred Procedur...	Running	-	0
dwm.exe	2352	Desktop Window ...	Running	7	0
csrss.exe	496	Client Server Runti...	Running	13	0
System	4	NT Kernel & System	Running	129	0

Services — 0% CPU Usage

Name	PID	Description	Status	Group	CPU
MSSQL$SQLEXP...	1724	SQL Server (SQLEX...	Running		0
SysMain	960	Superfetch	Running	LocalSyste...	0
BFE	1376	Base Filtering Engi...	Running	LocalServic...	0
WdiServiceHost	340	Diagnostic Service ...	Running	LocalService	0
iphlpsvc	988	IP Helper	Running	NetSvcs	0
WSearch	3028	Windows Search	Running		0
DPS	1376	Diagnostic Policy S...	Running	LocalServic...	0
PlugPlay	712	Plug and Play	Running	DcomLaunch	0

Associated Handles — Search Handles

Associated Modules

Image	PID	Module Name	Version	Full Path

Select a process to see results.

Figure 3.7: CPU usage at 8-core parallel processing of the plain-SVD model

parallelizing can be motivated by the theory presented earlier in 3.5.3, but observing that the accuracy can even improve with a multi-core environment is intriguing. Fig. 3.8 tells us that the test RMSE values obtained with single-core (27.8977) and 2-core (27.8139) are not as good as those obtained with 4-core (27.6465 - the best) and 8-core (27.6747). However, it may have a simple explanation: at the moment, the algorithm is designed to run through the ratings in the dataset by order of appearance. Since the dataset is ordered by users, the algorithm inevitably follows that order when processing ratings. This may introduce some bias to the learning process, as the model is iteratively learned by recurrently fitting the data of some users before others, thus possibly leading to a loss of impartiality and consequent loss of predictive accuracy. When parallelizing the algorithm, although introducing some risk of inconsistency it also introduces randomness in processing ratings, making the learning process more stochastic. For example, when running through the dataset in a single-core fashion, by the time the ratings of user 2 start being processed the model already learned towards fitting all ratings of user 1. On a multi-core alternative, ratings from different users are being processed simultaneously, which introduces the bit of randomness in the processing of ratings that may eventually improve accuracy. The 4-core alternative yielded the best results in this experiment but it carries a higher processing time than the 8-core alternative (13.10 secs. vs 9.81 secs. per step). Again, depending on the system requirements, a trade-off between accuracy and

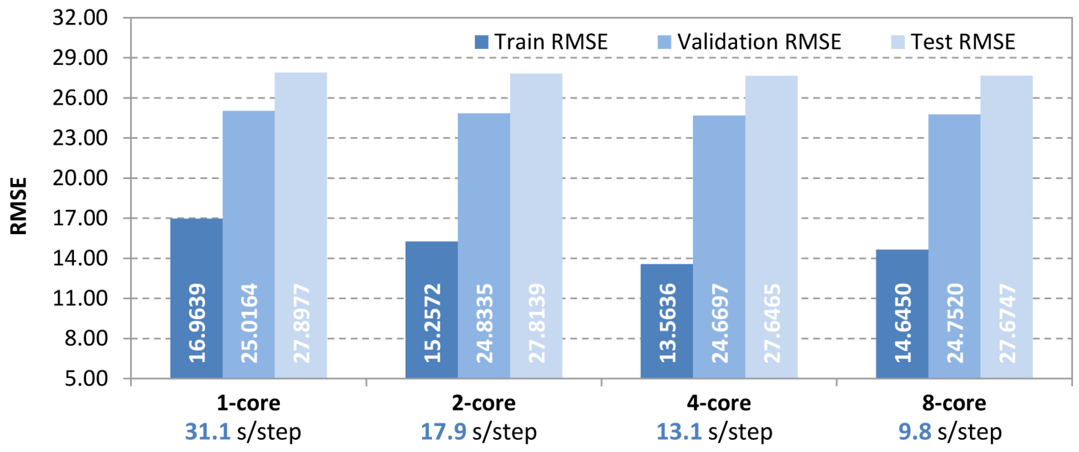


Figure 3.8: Performance of plain SVD model with different levels of parallelization

time can be a reasonable option.

3.7 Summary

In this chapter we discussed:

- The reasoning behind latent factor models and fundamentals of Singular Value Decomposition;
- Matrix factorization techniques and implementation challenges
- Parallelization of matrix factorization
- Analysis and evaluation of the implemented matrix factorization models

4

Modelling biases and temporal fluctuations

4.1 Introduction

On the previous chapter a matrix factorization model was proposed to assess user preferences in a collaborative fashion, through a latent factor approach. This model has proven to yield good results but it can be further improved by taking into account additional elements that haven't been explored before:

- **User and product biases:** users rate products differently and some products tend to get higher ratings than other. Such tendencies can be captured to enhance the matrix factorization model;

- **Temporal dynamics:** user preferences and product consumption trends drift over time. Such temporal concept drifts can also be captured to assess what portion of the rating patterns observed is associated with transitory trends and what portion tends to be preserved over time.

By analysing the dataset, one can observe global concept drifts over time that seem to affect all users, which suggests that time must be taken into account throughout the process of analysing user ratings and inferring their preferences. Fig. 4.1 illustrates time-related trends observed in the dataset. The rating patterns observed on fig. 4.1 show global variations that are common across users, supporting the theory that users rate products differently over time.

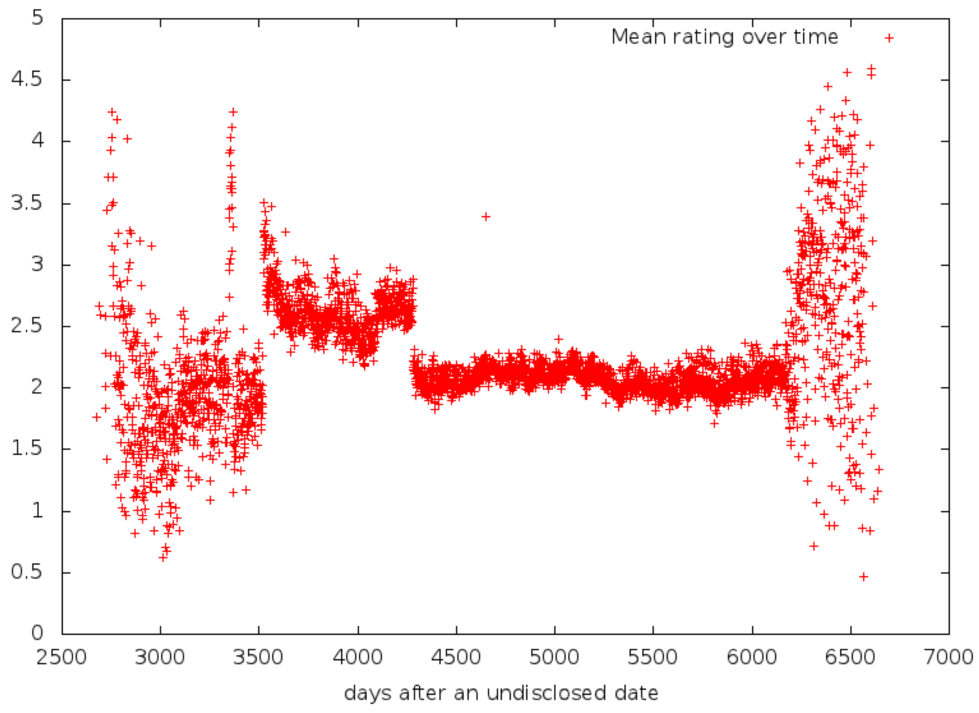


Figure 4.1: Temporal concept drift in user rating patterns

Let us now revisit the objective function established in the previous chapter, expressed by equation 4.1.

$$[P, Q] = \arg\min_{p_u, q_i} \sum_{r_{ui} \in R} (r_{ui} - p_u \cdot q_i^T)^2 + \lambda \cdot (\|p_u\|^2 + \|q_i\|^2) \tag{4.1}$$

Associated to our objective function is the rating prediction rule also established in the previous chapter, expressed by eq. 4.2.

$$\hat{r}_{ui} = p_u \cdot q_i^T \tag{4.2}$$

Now, both the objective function we intend to minimize and the rating prediction rule will undergo modifications to incorporate biases and temporal dynamics. These modifications will be introduced and addressed in the following sections.

4.2 The bias-SVD model

Although the latent factor vectors inference widely captures rating tendencies, some improvements can be made to the model by defining baseline predictors. This would allow for the factor vectors to simply swing the baseline predictions towards the real rating values instead of having to fully capture the rating patterns on their own. A straightforward choice for a baseline predictor is the global average of the observed ratings. Additionally, it is useful to account for the fact that some users tend to give higher ratings than others and some products tend to get higher ratings than others, as well. Based on this premise, arrangements can be made to capture these rating trends, regarded as user-related or product-related deviations from the average rating, henceforth referred to as user and product biases. For example, let us picture a scenario where there is a user u whose average rating to products is 4.3, a product i that tends to get an average rating of 3.7 and the global rating average is 3.5. In such scenario, user u tends to give 0.8 more rating units than the average user and product p tends to get 0.2 more rating units than the average product, which makes $\hat{r}_{ui} = 3.5 + 0.8 + 0.2$ a reasonable baseline prediction for a rating given by user u to product i. This reasoning leads to a new model where, by considering the global rating average and biases, the prediction rule can be modified into eq. 4.3.

$$\hat{r}_{ui} = \mu + b_u + b_i + p_u \cdot q_i^T \tag{4.3}$$

On the new prediction rule, the parameters μ, b_u and b_i represent the global rating average, the user bias and the product bias, respectively. Accordingly, the new least-squares problem intended to solve, which is an extension of the regularized eq. 4.1, is defined by eq. 4.4.

$$[P, Q] = \underset{p_u, q_i, b_u, b_i}{\arg\min} \sum_{r_{ui} \in R} (r_{ui} - \mu - b_u - b_i - p_u \cdot q_i^T)^2 + \lambda \cdot (\|p_u\|^2 + \|q_i\|^2 + b_u^2 + b_i^2) \tag{4.4}$$

4.2.1 Computation of the bias-SVD model

Several ways of computing biases beforehand have been suggested [8, 19]. Both cited suggestions for pre-computing biases take the same approach, which is statically assessing the mean rating deviation from the global average for each user and for each product, based on the training set. Additionally, regularization parameters are introduced in the calculation so that when there are few ratings to rely on, the bias value is shrunk towards zero. For example, if a given product was rated only once and got a rating of 5 in a 1 to 5 scale, it is not a fair assumption that its average rating is 5, since there is not enough data to support it properly. Based on the cited suggestions, a simple way of estimating user and product biases is:

$$b_i = \frac{\sum_{r_{ui} \in R_i} (r_{ui} - \mu)}{\lambda_1 + |R_i|} \tag{4.5}$$

Here, R_i is the set of ratings given to product i. Then, user biases can be estimated by:

$$b_u = \frac{\sum_{r_{ui} \in R_u} (r_{ui} - \mu - b_i)}{\lambda_2 + |R_u|} \tag{4.6}$$

Like on the product bias estimate, R_u is the set of ratings given by user u. The regularization parameters λ_1 and λ_2 should be determined by cross-validation. Estimating biases beforehand reduces the computational cost of the following steps of the algorithm, since no learning process is necessary to compute them. However, learning these biases along with the factor vectors has shown to deliver better prediction accuracy [19]. Learning biases iteratively avoids the instantaneous over-fitting brought by calculating them beforehand, since this previous calculation perfectly fits the ratings observed on the training set, leaving no margin for generalization over future ratings. The learning rules applied to learn the biases, defined by eq. 4.7, are similar to those applied to learn the factor vectors.

$$\begin{aligned} b_u &\leftarrow b_u + \gamma_2 \cdot (e_{ui} - \lambda_2 \cdot b_u) \\ b_i &\leftarrow b_i + \gamma_3 \cdot (e_{ui} - \lambda_3 \cdot b_i) \end{aligned} \tag{4.7}$$

Here, the γ_2, γ_3, λ_2 and λ_3 parameters are constants that play the same role as those involved in the factor vectors learning process but their values need not to be the same as on previous rules. These should, as well, be determined by cross-validation. User and product biases are updated upon processing each known rating and should be initialized as zeros, as if every user and product had the same rating tendencies.

4.3 The temp-SVD model

Besides changing across users and products, rating tendencies also change with time. By intuition, it would not be surprising that an overnight humour change would affect a user's rating behaviour, leading to inconsistencies if statically assessing this user's bias. Also, users tend to become more demanding with time or simply let go of the hype that once drove them to give high ratings to a specific product or category of products. Moreover, product popularity is also expected to drift over time and along with it changes in ratings given to these products occur. Such temporal drifts can be accounted for by modifying the baseline predictors, making them time-dependent. Notice that the goal of capturing temporal dynamics is not to predict future hypes for particular products or future mood changes in users, since such phenomena are nearly unpredictable. Instead, the goal of capturing past temporal dynamics is to isolate the real core of rating tendencies inherent to users and products from the noise introduced by transient hypes and mood swings that would eventually have no influence under a different temporal context. However, if periodic tendencies, i.e. tendencies that repeat over clearly defined periods of time can be unequivocally identified, such as seasonal trends (ex.: Reggae during the summer, "Home Alone" during Christmas), these could be introduced as input for future behaviour prediction. Otherwise, past temporal drifts influence is simply neutralized when predicting future ratings. Following these assumptions, Y. Koren [20] proposed an effective way of introducing time awareness into baseline predictors, which was fully adopted for this model. This solution comprises a modified definition of biases, now considering the time variable, as defined by eq. 4.8.

$$b_{ui}(t) = b_u(t) + b_i(t) \tag{4.8}$$

The function $b_{ui}(t)$ represents the time-dependent version of bias for user u and product i on day t. It is fair to assume that user rating behaviour can vary more frequently than product rating tendencies. Therefore, user and product temporal biases should be addressed differently. It is reasonable to mine user biases on a daily basis but for product biases taking into account bias drifts throughout longer periods of time ought to be enough. Thus, time bins spanning across all dataset days will be defined. The choice of the number of bins depends on how many days should be considered for each time period. In our implementation, $nBins = 40$ with $daysPerBin = 85$ each were used, but the general $Bin(t)$ function can be expressed as defined by eq. 4.9.

$$Bin(t) = \begin{cases} 1 & \text{if } 0 \leq t < daysPerBin \\ 2 & \text{if } daysPerBin \leq t < daysPerBin \times 2 \\ 3 & \text{if } daysPerBin \times 2 \leq t < daysPerBin \times 3 \\ \dots & \\ nBins & \text{if } daysPerBin \times (nBins - 1) \leq t < daysPerBin \times nBins \end{cases} \tag{4.9}$$

The $b_i(t)$ component of $b_{ui}(t)$ is then defined by:

$$b_i(t) = b_i + b_{i,Bin(t)} \tag{4.10}$$

A stationary portion b_i is preserved while adding a time-dependent portion $b_{i,Bin(t)}$ to capture product rating deviations over time.

To capture user temporal drifts on rating behaviour a slightly different approach is taken, as mentioned earlier. Gradual drifts in user behaviour can be modelled as a linear function based on the time deviation between the day of the rating attempted to predict and the user's mean day of rating. This deviation function can then be expressed as

$$dev_u(t) = sign(t - t_u) \cdot |t - t_u|^{\beta}, \tag{4.11}$$

where $|t - t_u|$ measures the distance in terms of days between the day of the rating which is being attempted to predict and the mean day of rating for user u. Additionally, the distance $|t - tu|$ is powered to a parameter β, which is determined by cross-validation. By introducing one more user-specific parameter α_u to scale the deviation function, we get the following user bias rule:

$$b_u(t) = b_u + \alpha_u \cdot dev_u(t) \tag{4.12}$$

As in product bias, a stationary portion of user bias b_u is preserved while adding a linear time deviation $dev_u(t)$ scaled by α_u. The parameter α_u, as well as b_u, needs to be automatically learned from data.

Another parameter can be added to absorb day-specific variability for each user, accounting for daily drifts on user rating behaviour, which is a good complement to the current user bias definition. This $b_{u,t}$ parameter will contain the fluctuations in user behaviour for user u on day t and must also be automatically learned from data, leading the user bias definition to:

$$b_u(t) = b_u + \alpha_u \cdot dev_u(t) + b_{u,t} \tag{4.13}$$

By including time-awareness in the baseline predictors, the prediction rule for the model becomes:

$$\hat{r}_{ui}(t) = \mu \quad + \quad b_u + \alpha_u \cdot dev_u(t) + b_{u,t} \quad + \quad b_i + b_{i,Bin(t)} \quad + \quad p_u \cdot q_i^T \tag{4.14}$$

Additionally, users rating scale can also vary with time, directly influencing the rating values given to products. Therefore, a time-dependent scaling factor $c_u(t)$ is added for each user and for each day, scaling product biases accordingly, leading to:

$$\hat{r}_{ui}(t) = \mu \quad + \quad b_u + \alpha_u \cdot dev_u(t) + b_{u,t} \quad + \quad (b_i + b_{i,Bin(t)}) \cdot c_u(t) \quad + \quad p_u \cdot q_i^T \tag{4.15}$$

Like user and product bias, the scaling factor $c_u(t)$ also results of the sum between a stationary portion c_u and a time-dependent portion $c_{u,t}$. Moreover, unlike all other bias-related variables, the c_u values for every user u are initialized as 1, instead of zero. Accordingly, the least-squares problem to solve is modified into eq. 4.16.

$$\begin{aligned}[P,Q] = \arg\min \sum_{r_{ui}(t) \in R} & (r_{ui}(t) - \mu - b_u - \alpha_u \cdot dev_u(t) - b_{u,t} - b_i - b_{i,Bin(t)} - p_u \cdot q_i^T)^2 \\ & + \lambda \cdot (\|p_u\|^2 + \|q_i\|^2 + b_u^2 + \alpha_u^2 + b_{u,t}^2 + b_i^2 + b_{i,Bin(t)}^2)\end{aligned} \tag{4.16}$$

4.3.1 Computation of temp-SVD model

The learning rules for all free variables introduced above are analogous to the static biases learning rules previously described in subsection 4.7 and need different learning rates and regularization constants, as well. A description of the temporal biases learning rules, applied for each observed rating $r_{ui}(t)$, follows:

$$\begin{aligned} b_{u,t} &\leftarrow b_{u,t} + \gamma_4 \cdot (e_{ui} - \lambda_4 \cdot b_{u,t}) \\ \alpha_u &\leftarrow \alpha_u + \gamma_5 \cdot (e_{ui} - \lambda_5 \cdot \alpha_u) \\ b_{i,Bin(t)} &\leftarrow b_{i,Bin(t)} + \gamma_6 \cdot (e_{ui} - \lambda_6 \cdot b_{i,Bin(t)}) \\ c_u &\leftarrow c_u + \gamma_7 \cdot (e_{ui} - \lambda_7 \cdot (c_u - 1)) \\ c_{u,t} &\leftarrow c_{u,t} + \gamma_8 \cdot (e_{ui} - \lambda_8 \cdot c_{u,t}) \end{aligned} \tag{4.17}$$

Parameter tuning is a cumbersome task, especially when so many variables are involved. In this implementation, several different learning rates for the factor vectors were tested to assess which were the ones that produce the best results. Other parameters used for the time-dependent baseline predictors were adopted from Y. Koren's previous works [19]. Whenever possible, for improved accuracy, parameters should be exhaustively tuned according to the context and dataset to which the recommender system is applied.

4.4 Implementation details

To embed biases and temporal dynamics into the plain-SVD model some changes are needed. Additional data structures were created to store the extra variables needed to capture biases and temporal drifts, as designed in the model:

- **meanAvg**: a variable to store the ratings mean average, a real positive value;
- **uBias[u]**: an array to store all user biases, which are real values;
- **pBias[p]**: an array to store all product biases, which are real values;
- **uBiasT[u,t]**: a two-dimensional array to store all day-specific user temporal drifts, which are real values;
- **uAlpha[u]**: an array to store all α parameters associated with user gradual temporal drifts, which are real values;
- **uAvgDay[u]**: an array to store all users' average rating days, which are integer values;
- **c[u]**: an array to store the static portion of users' scaling parameters, which are real values;
- **cT[u,t]**: an array to store the day-specific portion of users' scaling parameters, which are real values;;
- **pBiasT[u,t]**: an array to store all day-bin related product temporal drifts, which are real values;

The first improvement to be implemented was the embedding of non-temporal biases, which resulted in the *bias-SVD* model. The bias-SVD model modified the original plain-SVD model as described by alg. 4.4.1. For simplicity, the full description of the residuals update operations were omitted.
In alg. 4.4.1 user and product biases are taken into account to produce recommendations. The next improvement was to embed temporal fluctuations. Finally, the temporal information contained in the dataset is used, leading to the *temp-SVD* model. The algorithm to build the temp-SVD model works similarly to its previous bias-SVD version, only now

Algorithm 4.4.1 SGD main learning loop for the bias-SVD model

```
while validation RMSE decreases do
  for all latent factor $k$ in $K$ do
    [subtract residuals]
    for all $r_{ui}$ in trainingSet do
      $\hat{r}_{ui} \leftarrow \mu + b_u + b_i + trainResiduals_{ui} + P_{uk} \cdot Q_{ik}$
      $err \leftarrow r_{ui} - \hat{r}_{ui}$
      $P_{uk} \leftarrow P_{uk} + \gamma \cdot (err \cdot Q_{ik} - \lambda \cdot P_{uk})$
      $Q_{ik} \leftarrow Q_{ik} + \gamma \cdot (err \cdot P_{uk} - \lambda \cdot Q_{ik})$
      {/*update bias only once per step*/}
      if $k == 0$ then
        $b_u \leftarrow b_u + \gamma_2 \cdot (e_{ui} - \lambda_2 \cdot b_u)$
        $b_i \leftarrow b_i + \gamma_3 \cdot (e_{ui} - \lambda_3 \cdot b_i)$
      end if
    end for
    [sum residuals]
  end for
  [compute validation RMSE]
end while
```

taking into account the temporal variable. These modifications are intended to improve the models accuracy, while adding the least possible extra computational overhead. Alg. 4.4.2 describes the new temp-SVD algorithm.

Algorithm 4.4.2 SGD main learning loop for the temp-SVD model

```
while validation RMSE decreases do
  for all latent factor $k$ in $K$ do
    [subtract residuals]
    for all $r_{ui}(t)$ in trainingSet do
      $b_u(t) \leftarrow b_u + b_{u,t} + \alpha_u * dev_u(t)$
      $b_i(t) \leftarrow (b_i + b_{i,Bin_t}) \cdot (c_u + c_{ut})$
      $\hat{r}_{ui}(t) \leftarrow \mu + b_u(t) + b_i(t) + trainResiduals_{ui} + P_{uk} \cdot Q_{ik}$
      $err \leftarrow r_{ui}(t) - \hat{r}_{ui}(t)$
      $P_{uk} \leftarrow P_{uk} + \gamma \cdot (err \cdot Q_{ik} - \lambda \cdot P_{uk})$
      $Q_{ik} \leftarrow Q_{ik} + \gamma \cdot (err \cdot P_{uk} - \lambda \cdot Q_{ik})$
      {/*update bias only once per step*/}
      if $k == 0$ then
        $b_u \leftarrow b_u + \gamma_2 \cdot (e_{ui} - \lambda_2 \cdot b_u)$
        $b_i \leftarrow b_i + \gamma_3 \cdot (e_{ui} - \lambda_3 \cdot b_i)$
        $b_{u,t} \leftarrow b_{u,t} + \gamma_4 \cdot (e_{ui} - \lambda_4 \cdot b_{u,t})$
        $\alpha_u \leftarrow \alpha_u + \gamma_5 \cdot (e_{ui} - \lambda_5 \cdot \alpha_u)$
        $b_{i,Bin(t)} \leftarrow b_{i,Bin(t)} + \gamma_6 \cdot (e_{ui} - \lambda_6 \cdot b_{i,Bin(t)})$
        $c_u \leftarrow c_u + \gamma_7 \cdot (e_{ui} - \lambda_7 \cdot (c_u - 1))$
        $c_{u,t} \leftarrow c_{u,t} + \gamma_8 \cdot (e_{ui} - \lambda_8 \cdot c_{u,t})$
      end if
    end for
    [sum residuals]
  end for
  [compute validation RMSE]
end while
```

4.4.1 Accuracy optimization

Besides the measures that have been suggested in the previous chapter, additional accuracy improvements can be made by adjusting the model's parameters. As mentioned before, no exhaustive parameter tuning was performed for the temporal-related learning rates and regularization parameters, having adopted Y. Koren's [20] values. However, it is worth mentioning that other parameters can be tuned, such as the number of day bins chosen to represent time for product temporal bias capture purposes. With larger bins, i.e. bins that represent a wider day-span, the influence of each day on rating patterns is generalized over a larger set of days. On the other hand, using smaller bins allows for a finer resolution for capturing time-dependent trends, but there is the need to assure there is a representative number of products rated within the time-span represented by the bin. With few ratings inferred tendencies may be misleading, so it is important to mind this balance when determining the ideal number of days per bin.

4.4.2 Speeding up the algorithm

All solutions presented in chapter 3 were applied to the new bias-SVD and temp-SVD models, while adding a new biases storage measure, described in this section.

4.4.2.1 Storing biases residuals

Additional complexity reduction can be achieved by also storing the biases values. As defined on the algorithm's main loop, user and product biases values are updated only once per step. This means that throughout the main learning loop biases values remain unchanged. If the sums $b_u(t) + b_i(t)$ for each rating on the training set would be stored, it would only be necessary to access one variable instead of accessing two variables and computing their sum, for all factors after the first one. This brings an obvious improvement in terms of processing speed to the current algorithm. This observation led to the creation of a new structure $biasResiduals$ to store biases and adding an update operation for bias residuals to the main loop, as described in alg. 4.4.3.
These modifications will bring extra overhead to the processing of the first factor but will dramatically reduce computation time in processing the remaining factors.

Algorithm 4.4.3 Temp-SVD model learning algorithm with bias residuals storage

```
while validation RMSE decreases do
  for all latent factor $k$ in $K$ do
    [subtract residuals]
    for all $r_{ui}(t)$ in trainingSet do
      {/*use bias residuals only after updating the $biasResiduals$ structure*/}
      if $k == 0$ then
        $b_u(t) \leftarrow b_u + b_{u,t} + \alpha_u * dev_u(t)$
        $b_i(t) \leftarrow (b_i + b_{i,Bin_t}) \cdot (c_u + c_{ut})$
        $\hat{r}_{ui}(t) \leftarrow \mu + b_u(t) + b_i(t) + trainResiduals_{ui} + P_{uk} \cdot Q_{ik}$
      else
        $\hat{r}_{ui}(t) \leftarrow \mu + biasResiduals_{ui} + trainResiduals_{ui} + P_{uk} \cdot Q_{ik}$
      end if
      $err \leftarrow r_{ui}(t) - \hat{r}_{ui}(t)$
      $P_{uk} \leftarrow P_{uk} + \gamma \cdot (err \cdot Q_{ik} - \lambda \cdot P_{uk})$
      $Q_{ik} \leftarrow Q_{ik} + \gamma \cdot (err \cdot P_{uk} - \lambda \cdot Q_{ik})$
      {/*update bias only once per step*/}
      if $k == 0$ then
        $b_u \leftarrow b_u + \gamma_2 \cdot (e_{ui} - \lambda_2 \cdot b_u)$
        $b_i \leftarrow b_i + \gamma_3 \cdot (e_{ui} - \lambda_3 \cdot b_i)$
        $b_{u,t} \leftarrow b_{u,t} + \gamma_4 \cdot (e_{ui} - \lambda_4 \cdot b_{u,t})$
        $\alpha_u \leftarrow \alpha_u + \gamma_5 \cdot (e_{ui} - \lambda_5 \cdot \alpha_u)$
        $b_{i,Bin(t)} \leftarrow b_{i,Bin(t)} + \gamma_6 \cdot (e_{ui} - \lambda_6 \cdot b_{i,Bin(t)})$
        $c_u \leftarrow c_u + \gamma_7 \cdot (e_{ui} - \lambda_7 \cdot (c_u - 1))$
        $c_{u,t} \leftarrow c_{u,t} + \gamma_8 \cdot (e_{ui} - \lambda_8 \cdot c_{u,t})$
      end if
    end for
    [sum residuals]
    {/*update bias residuals only once per step*/}
    if $k == 0$ then
      for all $r_{ui}(t)$ in trainingSet do
        $b_u(t) \leftarrow b_u + b_{u,t} + \alpha_u * dev_u(t)$
        $b_i(t) \leftarrow (b_i + b_{i,Bin_t}) \cdot (c_u + c_{ut})$
        $biasResiduals_{ui} \leftarrow b_u(t) + b_i(t)$
      end for
    end if
  end for
  [compute validation RMSE]
end while
```

4.5 Evaluation

4.5.1 Datasets

To test the bias-SVD and temp-SVD models and compare them against the previously implemented plain-SVD model the same dataset (Yahoo! KDD Cup 2011 dataset) was used. This time, information about days of rating was used. For more details on this dataset refer to 3.6.1.

4.5.2 Experiment design

As in the previous chapter, only a portion of the dataset was used at a time in each test, so that more results could be presented within the available time for the test session. All experiments on the test session were performed over a $35,000$-user portion of the dataset. For some of the experiments, a 3-fold cross-validation evaluation was performed, meaning that 3 randomly picked disjoint $35,000$-user portions of the dataset were used for 3 independent experiments, and the presented results are an average of these 3 experiments. The choice of splitting the dataset by users was made to assure that all available data about each user is used to make predictions for that user, and because the Yahoo! dataset already comes grouped by user, which simplifies the splitting process. In all tests, different numbers of latent factors were used: 20, 50, 100 and 200, as well as 8-core parallelization by default. The evaluation section is organized as follows:

- **Parameter tuning**: For the bias-SVD and temp-SVD models, tests were performed to find which are the learning rates (the γ parameters in eq. 3.28) that provide the best results. In the literature several values were suggested for these parameters, ranging from 0.001 to 0.008. Hence, tests were performed for the range of $[0.001, 0.020]$ with 0.001 increments between tested values. The regularization parameter λ was fixed at 0.015. Additionally, the learning rates and regularization parameters used for learning biases were adopted from Y. Koren's work [18].

- **Adaptive learning rate**: The time-SVD model was tested with and without using adaptive learning rate and results were compared.

- **Model comparison**: The bias-SVD and the temp-SVD models were tested with the best factor learning rates previously assessed and their results were compared against the plain-SVD model.

- **Parallelization**: The learning process of bias-SVD and temp-SVD models were tested with different levels of parallelization (single-core, 2-core, 4-core and 8-core) and respective performances were analysed and compared in terms of speed and accuracy.

- **Model selection**: An analysis over the complex issue of selecting the best model to produce accurate recommendations was carried out.

4.5.3 Results and discussion

4.5.3.1 Parameter tuning

For the bias-SVD and temp-SVD models, tests were also performed to assess which was the best factors learning rate. The results are presented on figs. 4.2 and 4.3, respectively.

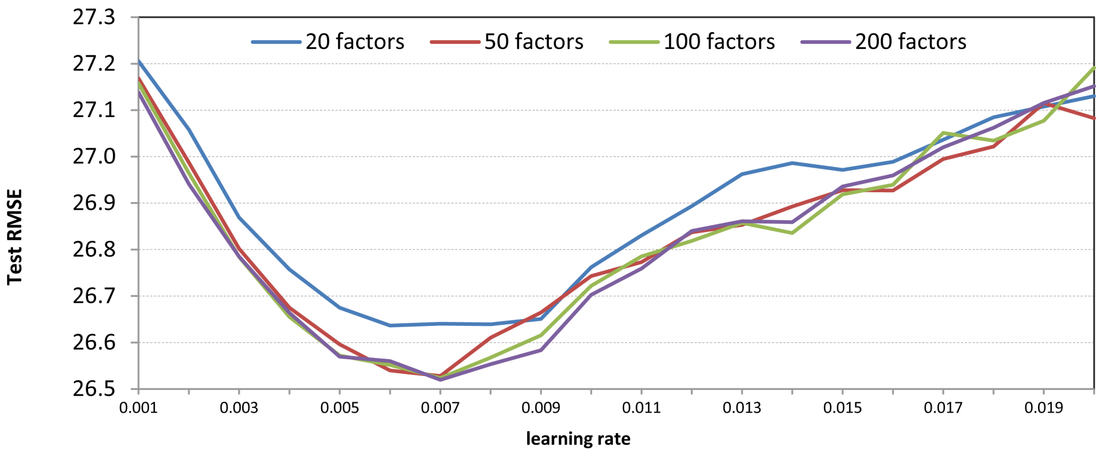


Figure 4.2: Bias-SVD model with different learning rates

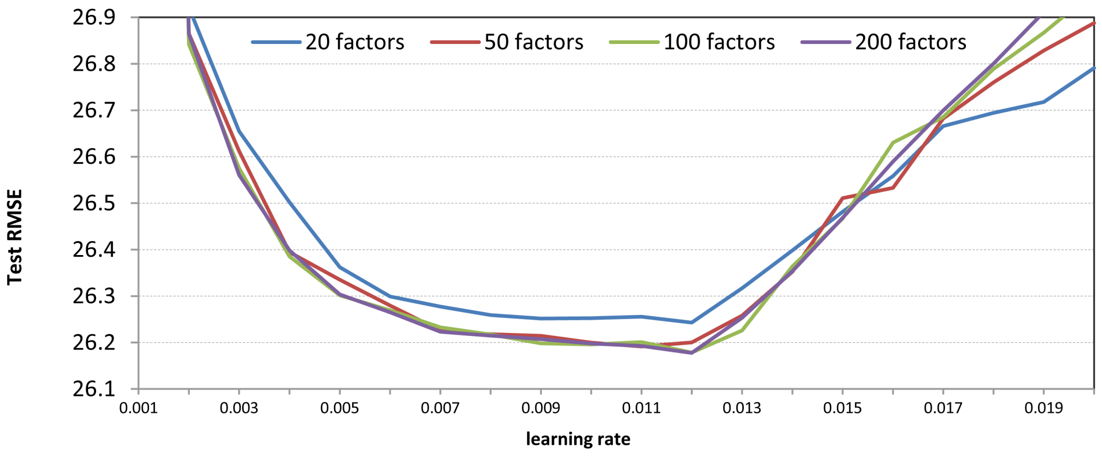


Figure 4.3: Temp-SVD model with different learning rates

It can be observed that different learning rates provide different results. For the bias-SVD model, 0.007 appears to be the best learning rate, while for the temp-SVD model 0.012 appears to be better.

4.5.3.2 Adaptive learning rate

The following chart, illustrated by fig. 4.4, shows the results of experiments made with the temp-SVD model using adaptive and non-adaptive learning rates for several learning rate values. Again, the improvements we aimed to achieve by using an adaptive learning rate were related mostly with computation time, but also with accuracy. We expected to find the right combination between a learning rate and a progressive decay rate that would allow the algorithm to have a fast learning at the beginning but a slower and more careful learning when getting closer to the final solution.

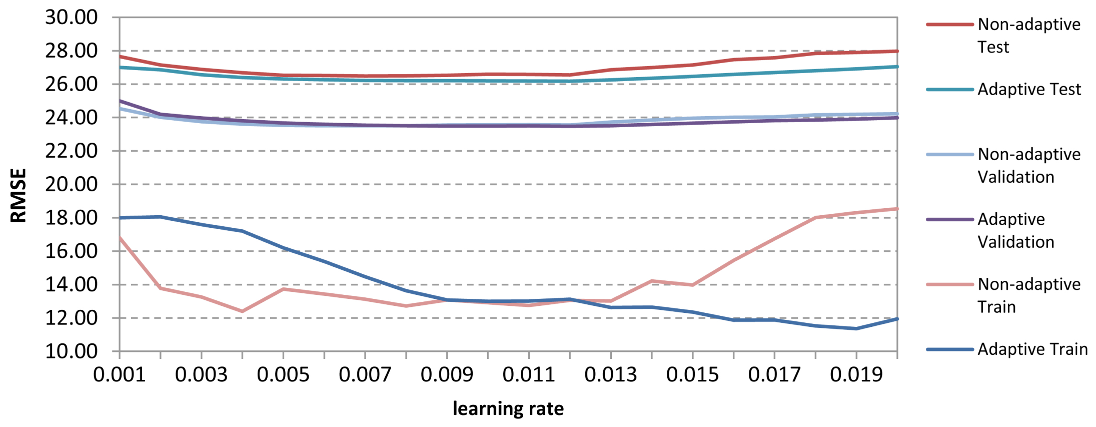


Figure 4.4: Performance comparison between adaptive and non-adaptive learning rates for the temp-SVD model using 200 factors

In chapter 3 we observed that using an adaptive learning rate for learning the plain-SVD model did not produce better results than those obtained with a non-adaptive learning rate. Unlike with the plain-SVD model, for the temp-SVD model this modification brought significant improvements in terms of accuracy and computation time, since the best RMSE results obtained on the test set were 26.1777 using adaptive learning rate with a learning rate of 0.012, and 26.4814 using non-adaptive learning rate with a learning rate of 0.007. Moreover, the number of steps taken was 103 with the adaptive learning rate and 28 with non-adaptive learning rate. Although the non-adaptive learning rate scenario allowed for the model to take less steps, it also produced a much worse final test RMSE result.

Although many different learning rates were tried, both the minimum learning rate set for the adaptive learning rate approach (set to 0.0007) and the progressive decay rate (set to 0.9) were not exhaustively tested. Additional experiments testing different combinations of learning rate, min. learning rate and progressive decay rate would be useful to further test this option. In future works, a more exhaustive experimentation session with such intent shall be taken.

4.5.3.3 Model performance comparison

Fig. 4.5 shows a comparison between all three models presented on this thesis (plain-SVD, bias-SVD and temp-SVD), each of them trained with the best learning rates discovered earlier. The embedding of biases to the plain-SVD model appears to improve the

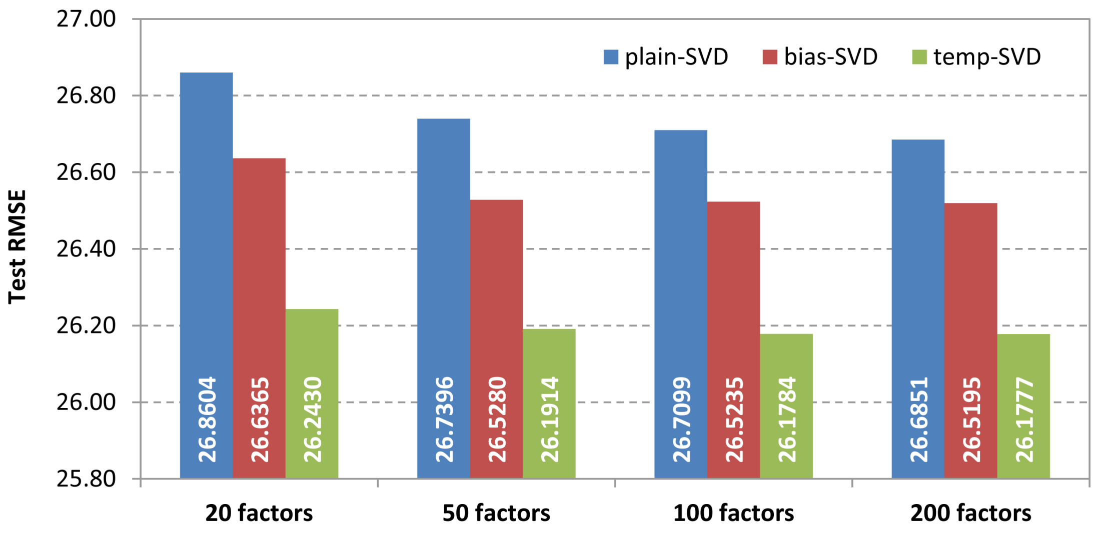


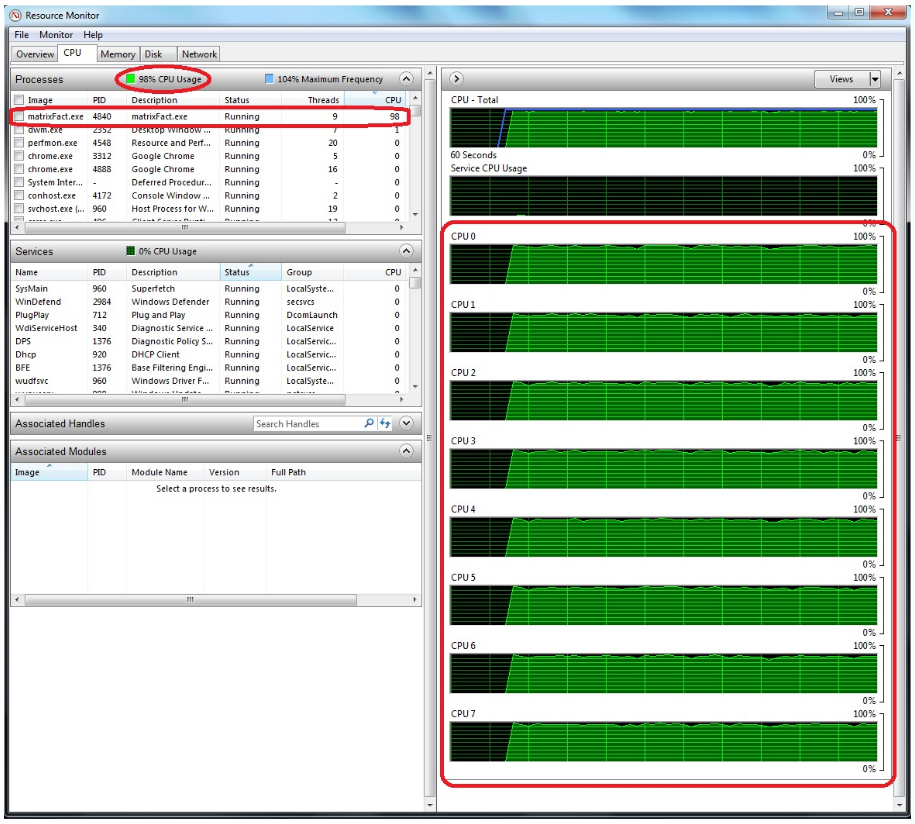
Figure 4.5: Comparison between plain-SVD, bias-SVD and temp-SVD models

model's accuracy for every tested number of latent factors. Additionally, the embedding of temporal dynamics to the bias-SVD model appears to improve the model's accuracy for every tested number of latent factors.

4.5.3.4 Parallelization

Finally, tests were performed for all three models to compare the learning performances with four different levels of paralellization: 1-core, 2-core, 4-core and 8-core. As with the plain-SVD model, the parallelization of bias-SVD and temp-SVD algorithms made an efficient use of all available processors, as fig. 4.6 shows. Parallelization tests with the bias-SVD and temp-SVD models were also performed with a learning rate of 0.015 and the learning process was allowed to run for 20 steps only. Similarly to what happened in the previous chapter, such high learning rate value was used to amplify the eventual damage in terms of predictive accuracy brought by parallelization-related inconsistencies, when these occur. The overall results of parallelization tests are illustrated by fig. 4.7.

These tests bring us to similar conclusions to those taken after the previous chapter's tests session: as expected, the processing time per step is lower when more processors are used. As for the RMSE values obtained, these are somewhat surprising. The fact that



Processes 🟩 98% CPU Usage 🟦 104% Maximum Frequency

Image	PID	Description	Status	Threads	CPU
matrixFact.exe	4840	matrixFact.exe	Running	9	98
dwm.exe	2352	Desktop Window ...	Running	7	1
perfmon.exe	4548	Resource and Perf...	Running	20	0
chrome.exe	3312	Google Chrome	Running	5	0
chrome.exe	4888	Google Chrome	Running	16	0
System Inter...	-	Deferred Procedur...	Running	-	0
conhost.exe	4172	Console Window ...	Running	2	0
svchost.exe (...	960	Host Process for W...	Running	19	0

Services 🟩 0% CPU Usage

Name	PID	Description	Status	Group	CPU
SysMain	960	Superfetch	Running	LocalSyste...	0
WinDefend	2984	Windows Defender	Running	secsvcs	0
PlugPlay	712	Plug and Play	Running	DcomLaunch	0
WdiServiceHost	340	Diagnostic Service ...	Running	LocalService	0
DPS	1376	Diagnostic Policy S...	Running	LocalServic...	0
Dhcp	920	DHCP Client	Running	LocalServic...	0
BFE	1376	Base Filtering Engi...	Running	LocalServic...	0
wudfsvc	960	Windows Driver F...	Running	LocalSyste...	0

Associated Modules

Image	PID	Module Name	Version	Full Path

Select a process to see results.

Figure 4.6: CPU usage at 8-core parallel processing of the temp-SVD model

there is no significant accuracy loss when parallelizing can be motivated by the theory presented earlier in subsection 3.5.3, but observing that the accuracy can even improve when adding processors is intriguing. However, it may have a simple explanation: at the moment, the algorithm is designed to run through the ratings in the dataset by order of appearance. Since the dataset is ordered by users, the algorithm inevitably follows that order when processing ratings. This may introduce some bias to the learning process, as the model is iteratively learned by recurrently fitting the data of some users before others, thus possibly leading to a loss of impartiality and consequent loss of predictive accuracy. When parallelizing the algorithm, although introducing some risk of inconsistency it also introduces randomness in processing ratings. For example, when running through the dataset in a 1-core fashion, by the time the ratings of user 2 start being processed the model already learned towards fitting all ratings of user 1. On an 8-core alternative, ratings from 8 different users are being processed simultaneously, which introduces the bit of randomness in the processing of ratings that may eventually improve the accuracy.

4.5.3.5 Model selection

Although we rely on cross-validation to determine when to interrupt the learning process, aiming to provoke that interruption in the moment when the model is at its best to predict future ratings, it is always impossible to be certain about whether the algorithm

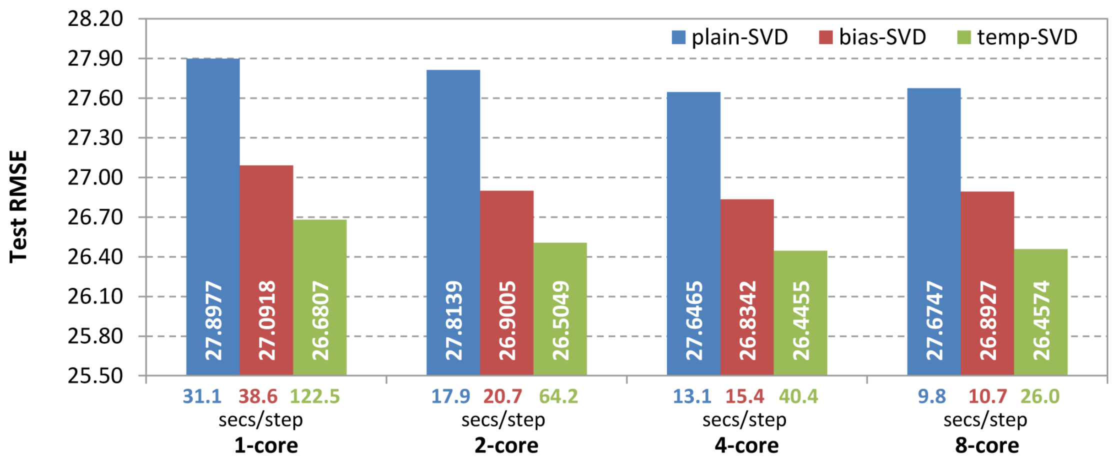


Figure 4.7: Performance of plain-SVD, bias-SVD and temp-SVD models with different levels of parallelization

finished the learning process in the right moment or not.

When we introduced the concept of cross-validation (subsection 3.4.1) it was mentioned that, ideally, the test set would only be used once, to evaluate the accuracy of the model after the algorithm converged, and the result from that evaluation would be final. The reason why the test set should be used only once is that in a real-life scenario there is no test set to help us quantify the accuracy of the model produced. We can never be sure if the model would be better if the algorithm would have been interrupted at a different stage.

Throughout these experiments, we tried to use the test set only at the end of the learning process to produce credible final results. In this subsection we will step out of this *use-it-once* practice to assess if the cross-validation method really led us to the best possible model. To do so, we attempted to predict the ratings on the test set throughout the whole learning process instead of only in the end it, regardless of whether the algorithm had converged (according to the validation RMSE progress using the cross-validation method) or not.

The experiment consisted in assessing the test RMSE throughout the learning process at each 5 steps, for the temp-SVD model using the learning rate that yielded the best results and 200 latent factors. The best test RMSE obtained with the temp-SVD model was $= 26.1777$ at step 103 using a learning rate of 0.012 and 200 latent factors.

Fig. 4.8 shows the RMSEs progression for the temp-SVD model obtained at every 5 steps of the learning process from step 30, using the aforementioned parameters.

According to the cross-validation method, the algorithm entered an over-fitting situation at around step 105 - that was the moment when the train RMSE continued decreasing but the validation RMSE stabilized. That should be the right moment to interrupt the

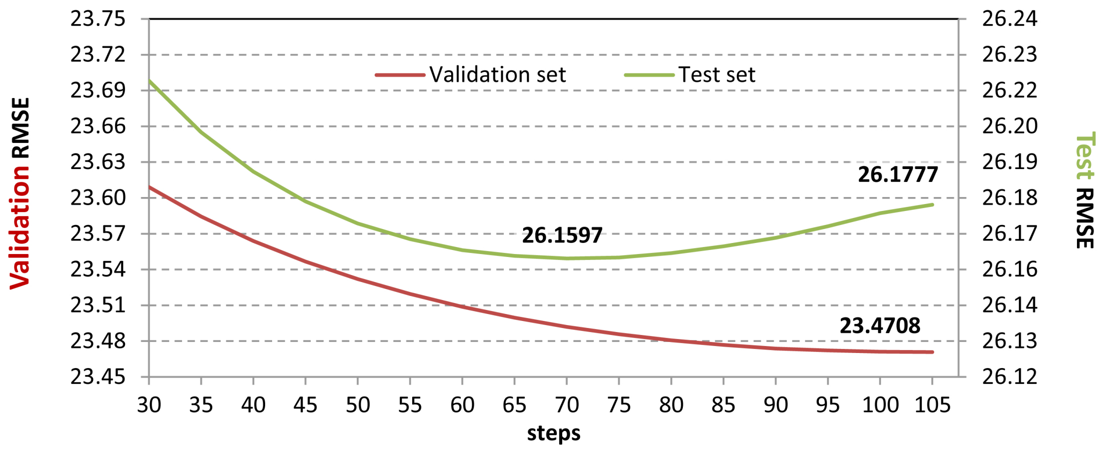


Figure 4.8: RMSEs on the test and validation sets throughout the temp-SVD model learning process

learning process and consider the model as good as it can be to predict unknown ratings. As fig. 4.8 shows, the best test RMSE obtained was at step 70, which illustrates the challenge associated with model selection. Notice that the validation and test axis have different RMSE scales, so that both progression curves could fit in the chart and be easily visualized. Although our cross-validation method indicated that the model was at its best at step 103, RMSE evaluation on the test set throughout the learning process showed different. This is an unavoidable issue when selecting the best predictive model, which is worth looking into in future works.

4.6 Summary

In this chapter we discussed:

- The introduction of user and product rating biases to enhance the matrix factorization model.
- The introduction of temporal dynamics, enhancing the model to account for temporal concept drifts.
- Computational constraints, efficiency and parallelization of the algorithm.
- Analysis and evaluation of all proposed models and solutions to the implementation challenges.

5 Group-based recommendations

5.1 Introduction

This chapter addresses an extension of the personalization problem where recommendations must be produced for different users simultaneously, i.e., recommendations for groups of users. When there is more than one user to please, recommendations must be provided in a different way so that the whole group of people is satisfied. The real-life example we will focus on to formulate the problem is a context where there are many people gathered listening to music. Hence, the challenge we take on is sorting out a playlist of songs to present to this group, based on the awareness of some context elements, namely the presence of some individuals whose preferences are known. As before, an extensive database with user preferences expressed by a user-product ratings matrix is used to build a predictive model, which is afterwards combined with clustering techniques and context awareness to produce a meaningful playlist of songs.

Group-based recommendations are recommendations geared towards not only a particular user but also to the surrounding context. Context is the term we adopt to name the scope of elements within the environment of the target users, which can be somehow captured and used to complement the recommendation system. By being aware of this context we seek for not only taking into account the users' explicit feedback on products, but also the surrounding environment on which they will receive product recommendations, and even consume them in this case, since songs on the playlist will just play automatically. Elements like the number of people around, the presence of some users already profiled by the recommender system, the time of the day, the season of the year or even the noise level can be useful to elaborate an appropriate playlist. Capturing such elements can be done by using a webcam and a microphone connected to the computer, for

example. Above all other elements, determining the group of people for whom recommendations must be computed is a key step of the playlist creation process. This thesis focuses mostly on the pre-processing stages which comprise matrix factorization techniques to build a solid predictive model that serves as spinal cord to the system. Thus, methods for capturing the aforementioned elements of the context were not explored thoroughly.
In the next sections, each of the stages comprised in this process will be addressed with more detail. Figure 5.1 presents a global overview of the proposed group recommenda-

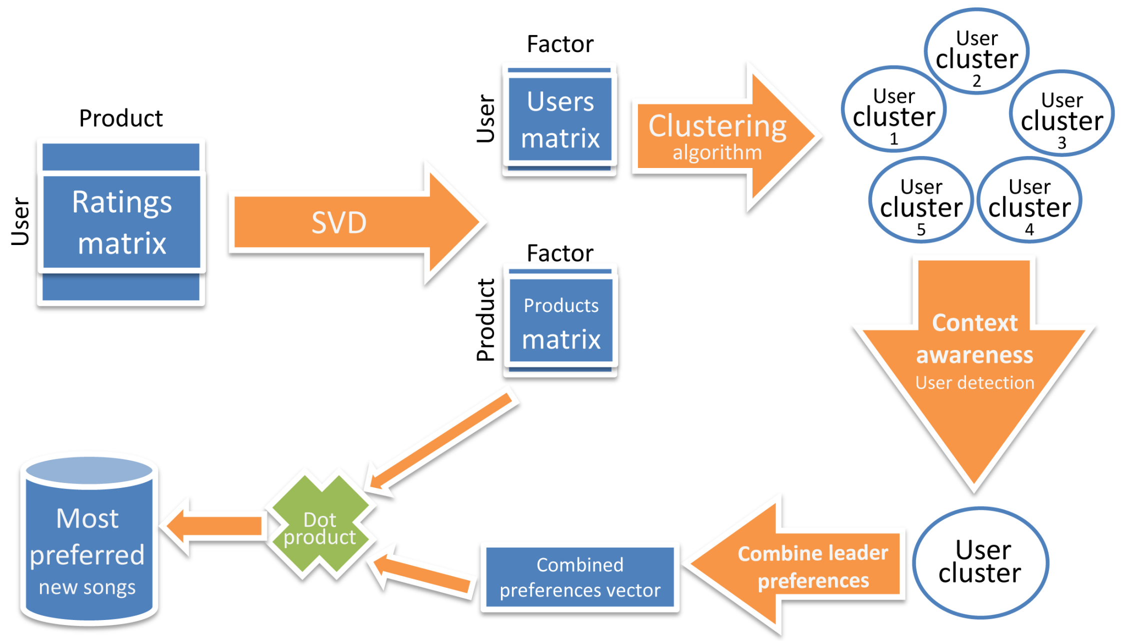


Figure 5.1: Graph of the song playlist production process

tion framework, illustrating its main stages and action flow towards the final result. On this figure, each labelled orange arrow represents one of the four main steps of the group recommendation process:

- **First step:** decomposing the user-product ratings matrix into latent factor representation of users and products through SVD, producing a user-factor matrix and a product-factor matrix;
- **Second step:** applying a clustering algorithm over the user-factor matrix, dividing users into several groups based on their preferences;
- **Third step:** detecting useful elements from the surrounding context, namely the identity of some of the users, and determine to which user groups they belong;
- **Forth step:** combining target users' preferences with group preferences to produce a list of recommendable products;

On this thesis, context awareness is obtained by direct user input, i.e., the system relies on users to manually identify some persons within the group which already had their preferences stored and analysed on the pre-processing stages. This direct input will help understanding the type of people present in the target environment. Once identified these users, the system tries to associate them with some previously discovered user groups within the database. The discovery of such groups will be addressed on section 5.2. With this information, the preferences of the detected users are combined with the preferences of the group to assess which products - songs, in this case - will most please these users as a crowd. The way these preferences are combined is further addressed on subsection 5.2.1 and on section 5.3. Once these preferences are assessed, a set of songs can be gathered accordingly. This set of songs can then be processed and adapted to build a diverse playlist.

5.2 Discovering groups of users

The discovery of groups of users is performed after the matrix factorization stage, when user-factor and product-factor matrices are already computed. One of the peculiarities of our system is that clustering is performed in the latent factor space inferred during the matrix factorization stage. The alternative would be performing clustering directly over the user-product ratings matrix, using products as dimensions and ratings as coordinates. However, such alternative would imply having a nearly 600K-dimensional space, instead of a 200-dimensional (if using 200 latent factors). The reasoning behind the choice of using latent factors as dimensions is not only the achieved dimensionality reduction but also the confidence that the latent factor representation of users and products obtained from matrix factorization truly captures user and product implicit characteristics, allowing for representing them independently from each other.

5.2.1 Groups of users and leaders

The main goal of this chapter is setting up a method for providing group recommendations. To achieve this goal we need to determine to which group each target user belongs. Knowing to which group each target user belongs helps pointing the systems' predictive power in the right direction, aiming for producing recommendations to the specific groups involved.

Fig. 5.2 illustrates the process that occurs from the moment when users are detected within the target context, represented as step 1, to the moment the detected users' and respective groups' preferences are combined into a composite preferences vector, represented as step 4.

Step 2 represents the moment when a detected user is associated to a group according to his latent factor vector. The system will consider that a user belongs to the cluster

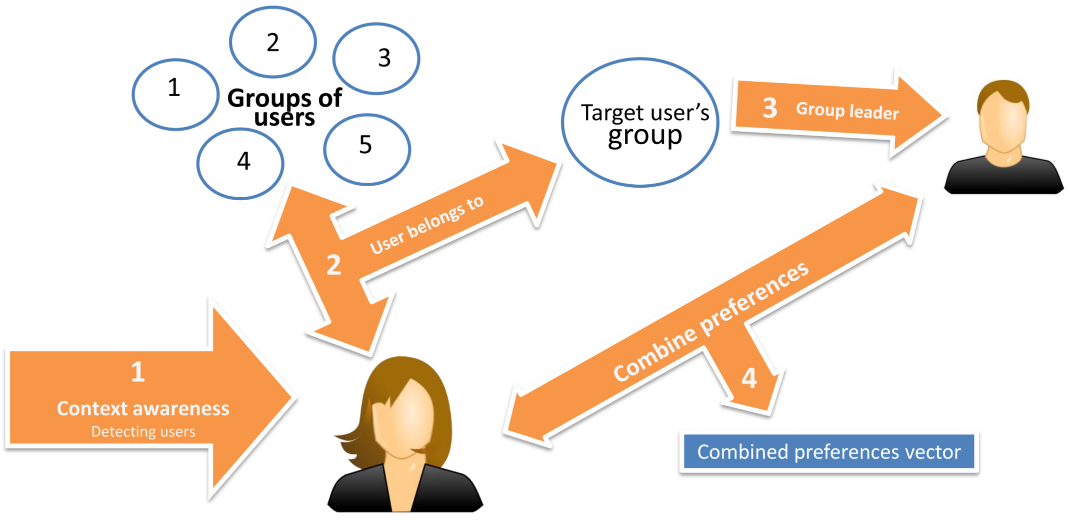


Figure 5.2: Illustration of target users detection and preference combination with group leaders

that has its centroid closer to that user. Although detected users belong to an identified group, they may have preferences that diverge from many other users in the same group, which can render these detected users as unsuitable sources for predicting the rest of their groups' preferences. This is more likely to happen for large groups where preference diversity inevitably varies more, despite the expected similarities across users within a group. For this reason, there is the need to find a representative user for each group, to serve as a reference regarding its group's preferences. We will henceforth refer to this representative user as group leader. The group leader should be the one whose preferences are closer to the average preferences of all other users within the group. The process of identifying the group leader is represented as step 3. Once identified the group leaders to which our detected target users belong, the preferences of all these actors are combined to find some common ground across all users, putting some emphasis on the detected users, though. Step 4 represents the process of combining detected users' and respective group leaders' preferences. This step is the one that raises a more interesting challenge, which is deciding how to combine these preferences. This issue will be further addressed in section 5.3.

5.2.2 Implementation description

Group discovery for this project was performed with k-means algorithms, technique which was addressed earlier in subsection 2.4.1.2. As we recall, in k-means clustering we are given a set of n data points, the users in this case, in a d-dimensional space and number of clusters k. The problem is to estimate a set of k points, called centroids, so as to minimize the mean squared distance from each data point to its nearest center, called the average distortion. The k-means algorithm applied to this problem is described by alg. 5.2.1.

Algorithm 5.2.1 K-means algorithm

```
for all $c_i$ in centroids $C$ do
  $c_i \leftarrow randomCoordinates(K)$
end for
while any $c_i$ coordinates change do
  for all $p_u$ in user-factor matrix $P$ do
    $c \leftarrow nearestCentroid(p_u, C)$
    $assignCentroid(p_u, c)$
  end for
  for all $c_i$ in centroids $C$ do
    $c_i \leftarrow meanCoordinates(assignedUsers)$
  end for
end while
```

This set of steps represents the typical k-means algorithm, also know as Lloyd's algorithm. However, the normal k-means algorithm can get stuck at local minima, far from the optimal solution. For this reason it is common to consider heuristics based on local search, in which centroids are randomly swapped in and out of an existing solution. New solutions are accepted if they decrease the average distortion, and otherwise they are ignored. It is also possible to combine these two approaches (normal k-means and local search), producing a type of hybrid solution. In this thesis we used Mount's [16, 17] implementation of k-means. Besides standard k-means, we also experimented two other variants of k-means, as listed below:

- **Swap:** A local search heuristic, which works by performing swaps between existing centroids and a set of candidate centroids.
- **Hybrid:** A more complex hybrid of normal k-means and Swap, which performs some number of swaps followed by some number of iterations of the k-means algorithm.

The distance measure used on this implementation was the euclidean distance.

5.2.3 Clustering tests

Normal k-means, Swap and Hybrid variants were tested with 200 initial random centroids and 34,000 users. Although the temp-SVD model was obtained using 200 latent factors, we decided to use a small number of factors, relying only on the most significant ones, i.e., the first ones, to perform the clustering. Clustering tests were performed using 2, 5, 8 and 10 factors. In the end of each algorithm, smaller clusters were eliminated and the users then assigned to these were reassigned to the nearest cluster. The minimum number of users per cluster was set to 150. We did not find significant differences between these 3 k-means variants in terms of clustering performance. Thus, we will henceforth focus our discussion in the normal k-means clustering algorithm. However, all tests performed with the other 2 experimented variants of k-means are documented in Appendix 7. Fig. 5.3 illustrates the results obtained for the k-means algorithm using 2, 5, 8 and 10 factors. Notice that the percentage axis is represented in $\log_2$ scale.

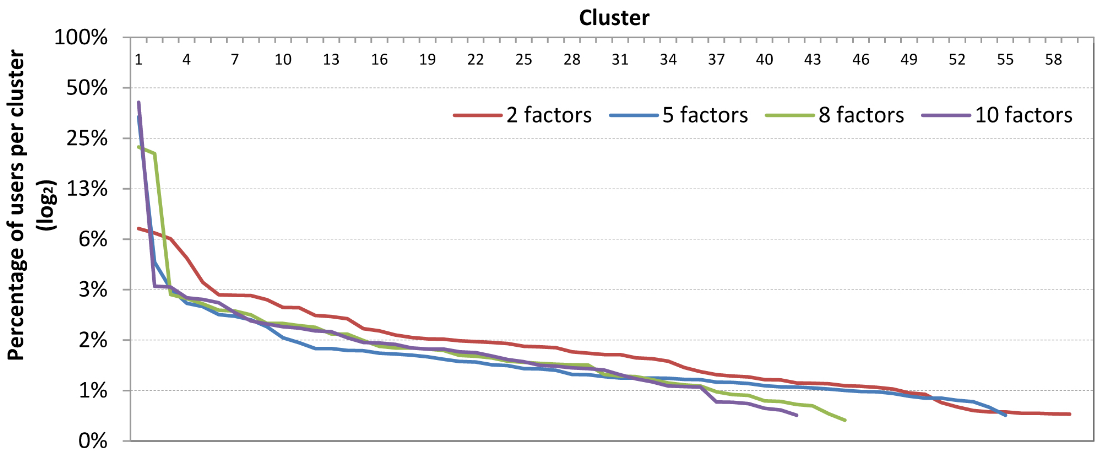


Figure 5.3: K-means clustering with different numbers of latent factors

We can observe that the most of the clustering results were good, in the sense that all experiments produced a high number of different clusters, most of them with a similar number of users. The experiment with 2 latent factors was the one that produced more evenly distributed users per cluster, where the largest cluster contains 7.24% of all users, while on the other experiments we can observe clusters containing $20\%-40\%$ of all users, which may indicate a less efficient clustering performance.

5.3 Computing group-based preferences

Computing group-based preferences comprises combining all information available on the target context in a way that allows for providing suitable recommendations. In this case, the context element to work with is the presence of some users detected by the system. If the system already possesses some knowledge regarding these detected users' preferences, such knowledge, along with pre-discovered groups of users, shall be used to produce a playlist.

Alg. 5.3.1 describes in pseudo-code the steps taken to produce group-based recommendations.

Algorithm 5.3.1 Group-based recommendation global algorithm

```
Step 1:
$[P, Q] \leftarrow tempSVD(R)$
Step 2:
$C \leftarrow kMeans(P)$
Step 3:
$U \leftarrow detectUsers()$
$tGroups \leftarrow getTargetGroups(U, C)$
Step 4:
$g \leftarrow combinePrefs(U, tGroups)$
$playlist \leftarrow getRecommendations(g, Q)$
```

Here, R is the user-product ratings matrix, C is the set of clusters resulting from performing clustering over the user-factor matrix P, U is the set of users detected, $tGroups$ is the set of target groups and g is the factor vector that results from combining the detected users' preferences and respective group preferences.

5.3.1 Combining preferences

Earlier on subsection 5.2.1 the concept of group leaders was introduced, as well as the idea of combining the preferences of detected context users with the preferences of their respective group leaders to somehow assess the global preferences of the target group. This subsection intends to discuss ideas for this preference combination. First, let us explain how group leaders are determined: each group of users discovered corresponds to a cluster obtained through the k-means algorithm. As mentioned earlier, each cluster is composed by the users that are closer to its centroid (the central point of a cluster) than to any other cluster centroid. Thus, for each cluster, one can state that the preferences of a user that happens to be placed on the very same coordinates as its centroid are an average of all other users' preferences within that same group. Such user would then be the perfect candidate for representing the whole group. For this reason, we define that the user who is closer to the respective centroid is elected as the leader of the group, representing its preferences. After detecting users and finding the respective group leaders, their preferences must be combined to produce a playlist. Two main categories of possible approaches to the users and leaders preference combination are:

- **Early fusion:** combining target users' factor vectors with group leaders' factor vectors to obtain a resulting factor vector that represents their combined preferences. Later on, use this factor vector to obtain a set of recommendable products;
- **Late fusion:** obtaining a set of recommendable products for each target user and for each group leader, based on their individual factor vectors, and afterwards combine all recommendations to obtain a resulting combined set.

Both these categories of approaches are expected to yield good but different results, as long as the methods used for combining factor vector or recommendation sets are suitable for this particular problem. In future works these two approaches shall be compared against each other. For now, we focused on early fusion techniques.

5.3.1.1 Early fusion

An *early fusion* approach was taken, where the combination of target users latent-factor vectors was performed through a linear weighted combination, assigning more weight to more participative users, i.e., users that gave more ratings to products, as expressed by eq. 5.1.

$$g = \sum_{i}^{n} (\alpha_{u_i} \cdot u_i) \tag{5.1}$$

After detecting users and assessing to which groups these belong, the system identifies the group leaders. For each group, the group leader is the user that is closer to the respective cluster centroid, thus rendering this user as the most representative one. Group leaders' preferences are used to add diversity and smooth the group recommendations by combining these with detected users' preferences, extending eq. 5.1 into eq. 5.2:

$$g = \sum_{i}^{n} (\alpha_{u_i} \cdot u_i + \alpha_{l_i} \cdot l_i) \tag{5.2}$$

In eqs. 5.1 and 5.2, n is the number of target users and α_{u_i} and α_{u_i} are the weights assigned to the factor vectors of target user u_i and group leader l_i. As mentioned earlier, the weights assigned to each target user and group leader latent-factor vectors depend on the relation between the number of ratings given to products by these users and leaders and the total number of ratings given by all referred users and leaders $\alpha_u = \frac{nRatings_u}{totalGroupRatings}$. Alg. 5.3.1 can then be described with more detail by alg. 5.3.2.

Algorithm 5.3.2 Group-based recommendation global algorithm

```
1: $[P, Q] \leftarrow tempSVD(R)$
2: $C \leftarrow kMeans(P)$
3: $U \leftarrow detectUsers(); tGroups \leftarrow getTargetGroups(U, C)$
Step 4:
$g \leftarrow$ new $vector_K(0)$
for all $u$ in $U$ do
    $c \leftarrow getGroup(u)$
    $l \leftarrow getLeader(c)$
    $g \leftarrow g + \alpha_u \cdot P_u + \alpha_l \cdot P_l$
end for
$playlist \leftarrow getRecommendations(g, Q)$
```

Now we can take a more insightful look into the group recommendation global algorithm described in alg. 5.3.1, by detailing the $combinePrefs(U, tGroups)$ function in step 4, as alg. 5.3.2 shows.

5.3.2 Finding recommendable products

As illustrated by 5.1 and described in the beginning of this chapter, the first step of the group recommendation process is obtaining a latent factor representation of users and products, which are stored in user-factor and product-factor matrices, respectively. Let us revisit the matrix decomposition represented by eq. 5.3, that results from this reasoning:

$$R = P \cdot Q^T \Leftrightarrow \begin{bmatrix} r_{1,1} & \cdots & r_{1,n} \\ \vdots & \ddots & \vdots \\ r_{m,1} & \cdots & r_{m,n} \end{bmatrix} = \begin{bmatrix} u_{1,1} & \cdots & u_{1,k} \\ \vdots & \ddots & \vdots \\ u_{m,1} & \cdots & u_{m,k} \end{bmatrix} \cdot \begin{bmatrix} p_{1,1} & \cdots & p_{1,k} \\ \vdots & \ddots & \vdots \\ p_{n,1} & \cdots & p_{n,k} \end{bmatrix}^T \tag{5.3}$$

Here, matrix R is the ratings matrix, each vector (row) p_u of P represents a user u, and each vector (row) q_i of Q represents a product i. These matrices contain all vectors representing users and products, on which the following steps rely. The intent of latent factor representation is thoroughly described and explained in chapter 3. After obtaining this user and product latent factor representation, we intend to assess users' preferences for products as eq. 5.4 shows:

$$r_{ui} = p_u \cdot q_i^T \tag{5.4}$$

Again, r_{ui} is the preference of user u for product i, both represented as latent factor vectors. This reasoning was applied before to assess individual users' preferences. Now, we intend to apply the same reasoning to groups of users, by using a latent factor vector that represents the combination between detected users and group leaders instead of the latent factor vector representing an individual user, as expressed by eq. 5.5.

$$r_{gi} = g \cdot q_i^T \tag{5.5}$$

Here, g represents the combined preferences factor vector.
Once obtained this combined factor vector introduced in the previous subsection (5.3.1), recommendations can be computed by calculating the dot product between the combined factor vector and all product vectors contained in the product-factor matrix, thus assessing which products are more likely to satisfy this group. As usual, higher values on this vectors dot product indicate higher preference. After obtaining the group's preference scores for all products and listing them, the resulting list of products and preferences must be ordered according to preference value and the N top products can be selected as recommendable products, providing a song pool from which some songs can be chosen to build a playlist.

5.4 Evaluation

In this section the quality of the produced playlists will be analysed and discussed, in order to assess whether or not the ideas introduced in this chapter are pertinent and have the potential to become interesting baselines for future applications.

5.4.1 Datasets

The dataset used for this experiment is the one provided by Yahoo! to contestants of the KDD Cup 2011, as in previous chapters. The fact that all products contained in this dataset are musical products makes it eligible for these tests. Also, the fact that this dataset has been used in previous chapters was preponderant, since all matrix factorization techniques implemented on previous chapters were developed to be used with this particular dataset and oriented to suit its peculiarities. However, in further works it would be important to try these procedures with different datasets, since the relevance of these procedures could benefit from a more solid validation.

5.4.2 Experiment design

All the experiments conducted on this section used the same user-factor and product-factor matrices, since the goal was not to evaluate the quality of the matrix factorization model itself (this was done on previous chapters 3 and 4). These matrices were obtained by processing the training dataset using the ratings of $35,000$ users regarding all $624,961$ products. The model employed to take on this task was the temp-SVD model described in chapter 4 with 200 latent factors. After building the model, its resulting user-factor and product-factor matrices were stored into files and recurrently used as baseline for the following group recommendation process. From the $35,000$ users contemplated by the model, a $34,000$-users portion was used to obtain the clusters (groups) and the remaining 1000-users portion was used to select random users to play in as users detected by the system's context awareness. Although we made experiments with all 3 previously mentioned variants of k-means, the experiments documented in this section were all performed using the normal k-means algorithm, since the clustering results produced with the other 2 variants were not significantly different from those obtained with normal k-means. Moreover, the group recommendation framework presented in this chapter may use any clustering method. Nonetheless, the results from the experiments carried out with all 3 k-means variants are documented in Appendix 7.
The evaluation criteria used to assess the quality of playlists was the percentage of positive ratings given to songs (on the test set only) by the users that belong to the target groups, within the set of songs contained in the recommended playlists. A rating equal to or higher than 3.5, on a scale from 0 to 5, is considered a positive rating.

5.4.3 Results

For each of experiment, random users were chosen and their groups were identified. To improve the quality of this evaluation session, for each experiment we picked 4 users belonging to 4 different groups, so that we could attempt to produce multi-group recommendations. Groups are identified as *g1*, *g2*, *g3* and *g4*. Table 5.4.3 presents the groups involved in each of the 10 experiment setups and the number of users contained within those groups.

		2 factors	5 factors	8 factors	10 factors
#Exp	**Groups**	nr. of users	nr. of users	nr. of users	nr. of users
1	g1	282	702	431	927
2	g2	507	753	792	639
3	g3	309	459	801	312
4	g4	745	292	261	359
5	g1+g2	789	1455	1223	1566
6	g2+g3	816	1212	1593	951
7	g3+g4	1054	751	1062	671
8	g4+g1	1027	994	692	1286
9	g1+g2+g3	1098	1914	2024	1878
10	g2+g3+g4	1561	1504	1854	1310

Table 5.1: Statistics on group recommendation experiments

The following charts illustrate the results obtained in each experiment, using k-means clustering with different numbers of latent factors. The results are grouped by single-group, 2-group and 3-group experiments, represented by figs. 5.4, 5.5 and 5.6, respectively.

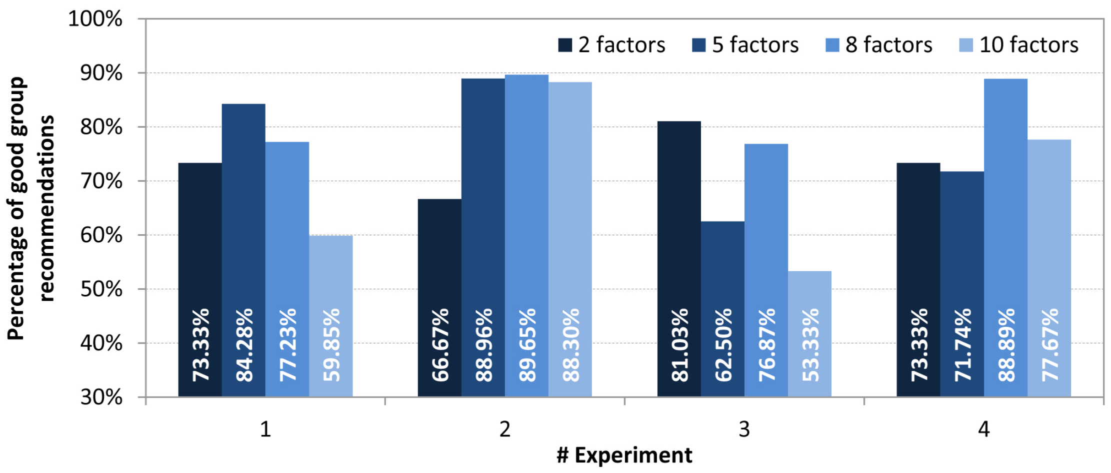


Figure 5.4: Results of single-group recommendation

In all of the single-group recommendation experiments the assessed satisfaction rates were all above 53%, and the majority of them (12 out of 16) were above 70%, which is a positive indicator.

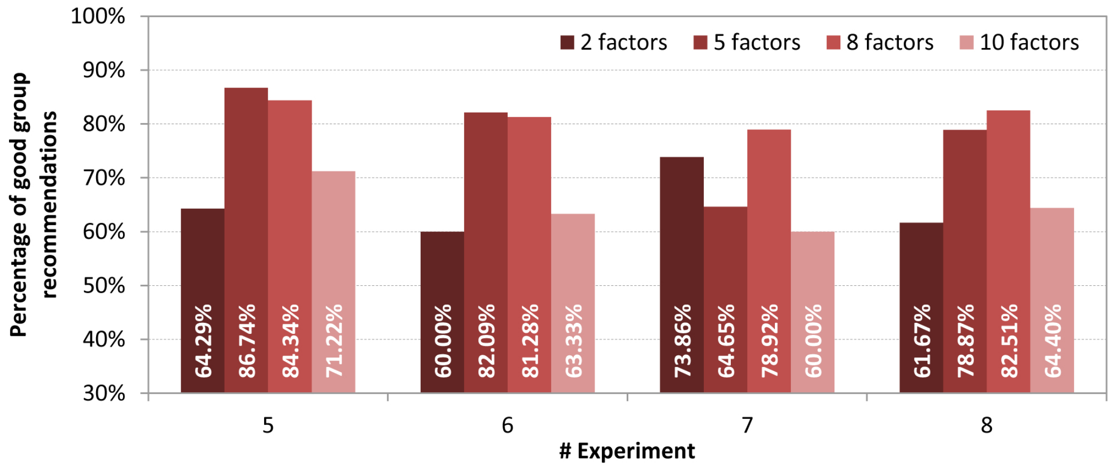


Figure 5.5: Results of 2-group recommendation

In all of the 2-group recommendation experiments the assessed satisfaction rates were all above 60% (better than the 53.33% minimum rate from the single-group experiments) and the majority of them (9 out of 16) were above 70%. Moreover, the 2-group experiment results present a smaller satisfaction rate fluctuation across experiments than the previous single-group experiments. Overall, the 2-group recommendation experiment results are encouraging.

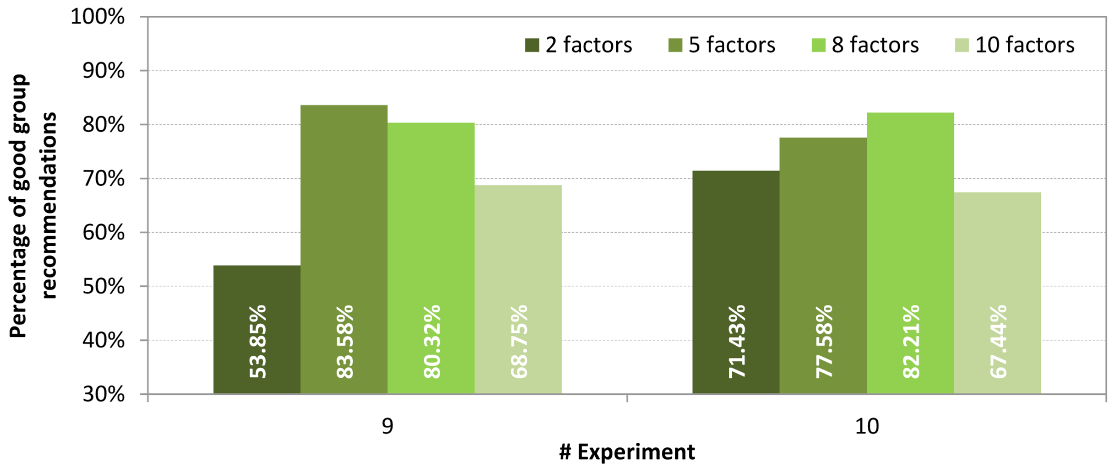


Figure 5.6: Results of 3-group recommendation

In all of the 3-group recommendation experiments the assessed satisfaction rates were all above 50% and the majority of them (5 out of 8) were above 70%, which is a positive indicator.

Additionally, we present a chart, illustrated by fig. 5.7, that shows the average results for each of the 3 groups of experiments documented above using different numbers of latent factors for clustering, providing a more global overview of these experiments.

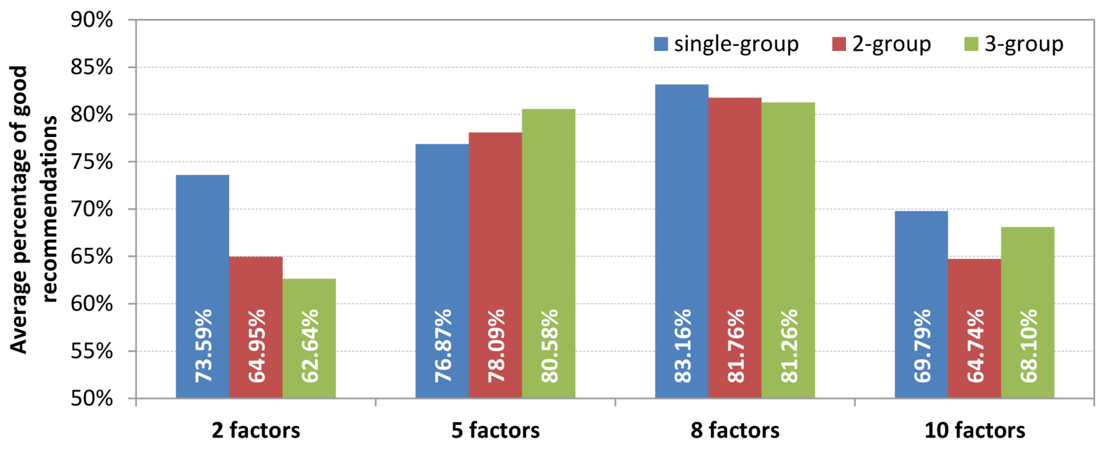


Figure 5.7: Averaged results of all 3 groups of experiments

The analysis of fig. 5.7 suggests that using the k-means algorithm with 8 latent factors for performing clustering produces the best results in terms of group satisfaction rate, with an average percentage of good recommendations above 80% for single-group, 2-group and 3-group experiments. Using 5 latent factors produced results nearly as good as those obtained with 8 factors, with an average percentage of good recommendations above 76% for single-group, 2-group and 3-group experiments.
It is my desire to perform more exhaustive experiments in future works, since the results here obtained are encouraging and render the here proposed framework as a promising baseline for future applications.

5.5 Summary

In this chapter we discussed:

- User clustering based on users' latent factor vectors, using different k-means variants.
- Introduction of the concept of group leaders.
- Combination of detected users' preferences with group leaders' preferences to produce group-oriented recommendations.
- Evaluation of group-based recommendations.

6

Conclusions and future work

6.1 Contributions summary

The goal of this thesis was to develop a system capable of analysing a dataset containing user feedback regarding products, from that feedback mine the relations between users and products and finally provide personalized product recommendations to users.
To meet this goal, a **matrix factorization model** based on singular value decomposition was implemented, exploring explicit user feedback in a collaborative fashion to infer user preferences.
This matrix factorization model was further improved to account for user and product rating biases, allowing for specific user-related and product-related rating tendencies to be captured and properly dealt with, leading to a **bias-aware matrix factorization model**.
The bias-aware matrix factorization model was further improved to account for temporal fluctuations in rating patterns, allowing for the time variable to be considered and embedded into the process of predicting user preferences for products, resulting in a **time-aware matrix factorization model**.
Along the process of developing these predictive models, computational constraints and efficiency challenges arose due to the implemented algorithm's complexity and the large dimension of the dataset. Such challenges led to the need for taking measures to optimize the efficiency of the learning algorithms, and the efforts made in that sense resulted in a **parallel stochastic gradient descent implementation** of the matrix factorization algorithm, designed to make the best possible use of all available processors.
The initial goal of this thesis was accomplished and further extended to pursue group-based recommendations, which combined the implemented matrix factorization models with clustering techniques to produce a **framework for recommending lists of products**

to groups of users, instead of focusing exclusively on a particular user.
The process of reflecting, developing, testing, discussing and documenting all these components into this thesis is my academic contribution to science, along with the promise of giving continuity to the work here presented.

6.2 Potential applications

The potential applications of the frameworks implemented in this thesis are mainly oriented towards any environments involving user consumption and evaluation of products. Every system that sells products to users and somehow collects their feedback regarding those products would greatly benefit from having an accurate recommender system mining its user-product interactions and making meaningful product recommendations that would lead users to more easily find what they like and consume more products. Any wide on-line business whose service is providing multimedia content is a good candidate to take the best advantage of an accurate - thus persuasive - recommender system.

6.3 Challenges and limitations

The main challenges and limitations associated with the implemented frameworks lie on their dependence on:

- reliable user feedback that truly captures users' preference for specific products. Such feedback is not always available or it is sometimes associated with low confidence levels;
- massive amounts of user input, without which matrix factorization approaches can't produce reliable results;
- computational power to timely process massive amounts of data and retrieve results;
- conclusive evaluation methodologies to assess the algorithms' quality. It is always a difficult task to assess whether the recommendations produced by a recommender system are good or not, before observing if the user actually consumed the recommended product. There are methods for performing such evaluation but they depend on the existence of great amounts of user feedback, otherwise the conclusions from such evaluation may be dubious.

6.4 Future work

Due to time constraints, some ideas were left out of this thesis. Nonetheless, it is worth to keep these ideas in mind for future works.

6.4.1 Implicit feedback

Implicit feedback, i.e., feedback inferred from all user-system interactions other than explicit ratings, can provide a valuable input for producing recommendations. Such input can be particularly useful to overcome the scarcity of explicit feedback, which is in many cases unavailable. Some implementations have been described in literature addressing this question and the natural course of the work produced in this thesis is to evolve into embedding implicit feedback to complement explicit feedback and provide a more solid ground for assessing user preferences.

6.4.2 Dynamic playlists with enthusiasm curves

Regarding the framework for producing playlists oriented to groups, introduced in chapter 5, an interesting extra feature when suggesting playlists would be to sort the songs within these playlists in a meaningful way. An interesting idea would be taking into account the bpm's (beats per minute) of songs to build a playlist that would suit the groups' mood. For example, if our context awareness tells us there is a large group of people in the target environment and there is a high level of noise and movement, perhaps songs with more bpm's would suit the groups' preferences for the moment. The most cumbersome step to fulfil such idea is gathering the bpm's for all songs in the dataset. For this project, we didn't have such information available. However, a different idea could have been explored to tackle the problem of sorting songs in a playlist. Assuming that within the set of candidate songs some will have a higher preference prediction than others, one can use such predictions to manage the enthusiasm brought by these songs to the group of people. Again, if somehow our context awareness tells us the crowd needs to be more stimulated, it would make sense to play in one of the most preferred songs and leave others for later. This question introduces the concept of *enthusiasm curve*. The idea is to design a curve to be followed by the playlist according to the level of predicted group preference for each song. With such curve we intend to tune the group's enthusiasm up or down along the playlist, according to a pre-defined criteria. For example, starting the playlist with enthusiastic songs (i.e., songs most highly preferred by the group), followed by some less enthusiastic ones and ending the playlist with the most enthusiastic songs could be a plan. We can easily relate such enthusiasm curve with several concerts or DJ sets we've attended before.

Additionally, song genres can be combined in a way that makes the flow of songs smooth and pleasant, avoiding abrupt changes of musical style in the environment. Song genres can also be useful to rule out songs that don't make sense within certain contexts. For example, if our context awareness tells us there is a large group around, with high levels of movement and noise and if its 11pm on a Summer evening, classical calm songs may not be the best choice to entertain the group, even if the environment is packed with Mozart fans.

6.4.3 Real-time feedback input

Within the playlist recommendation context, introduced in chapter 5, since recommendations are provided and products are consumed in real-time, it is only fair that these recommendations can adapt in real-time to the target users' feedback. Hence, users ought to be able to provide some feedback regarding the songs they are being given to listen to. An interesting idea for capturing such feedback would be setting up a webcam to track the group's movements and from those movements attempt to assess the levels of enthusiasm of that group. Such information could then be used to adapt the playlist in real-time. For example, whenever a song would be regarded as being "liked" by the users, its preference value would be increased to the top. Accordingly, whenever a song would be explicitly "disliked" by the users, it would be excluded from the playlist and some highly preferred songs would be set to play immediately after the current "disliked" song, as a way of compensating this group. These are simple and straightforward features idealized to provide a better user experience.

Bibliography

[1] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING*, 17(6):734–749, 2005.

[2] Nicholas J. Belkin and W. Bruce Croft. Information filtering and information retrieval: two sides of the same coin? *Commun. ACM*, 35:29–38, December 1992.

[3] Robert M. Bell, Jim Bennett, Yehuda Koren, and Chris Volinsky. The million dollar programming prize. *IEEE Spectr.*, 46:28–33, May 2009.

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

[5] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003.

[6] John S. Breese, David Heckerman, and Carl Myers Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *UAI*, pages 43–52, 1998.

[7] Andrew Crossen, Jay Budzik, and Kristian J. Hammond. Flytrap: intelligent group music recommendation. In *Proceedings of the 7th international conference on Intelligent user interfaces*, IUI '02, pages 184–185, New York, NY, USA, 2002. ACM.

[8] Simon Funk. Netflix update: Try this at home. *http://sifter.org/∼simon/journal/20061211.html*, 2006.

[9] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 35:61–70, December 1992.

[10] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22:5–53, January 2004.

[11] Oliver Hinz and Jochen Eckert. The impact of search and recommendation systems on sales in electronic commerce. *Business & Information Systems Engineering*, 2(2):67–77, 2010.

[12] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, pages 263–272, Washington, DC, USA, 2008. IEEE Computer Society.

[13] Michael Jahrer, Andreas Töscher, and Robert Legenstein. Combining predictions for accurate recommender systems. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '10, pages 693–702, New York, NY, USA, 2010. ACM.

[14] Anthony Jameson and Barry Smyth. The adaptive web. chapter Recommendation to groups, pages 596–627. Springer-Verlag, Berlin, Heidelberg, 2007.

[15] Gawesh Jawaheer, Martin Szomszor, and Patty Kostkova. Comparison of implicit and explicit feedback from an online music recommendation service. In *Proceedings of the 1st International Workshop on Information Heterogeneity and Fusion in Recommender Systems*, HetRec '10, pages 47–51, New York, NY, USA, 2010. ACM.

[16] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24:881–892, July 2002.

[17] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. A local search approximation algorithm for k-means clustering. In *Proceedings of the eighteenth annual symposium on Computational geometry*, SCG '02, pages 10–18, New York, NY, USA, 2002. ACM.

[18] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pages 426–434, New York, NY, USA, 2008. ACM.

[19] Yehuda Koren. The bellkor solution to the netflix grand prize, 2009.

[20] Yehuda Koren. Collaborative filtering with temporal dynamics. *Commun. ACM*, 53:89–97, April 2010.

[21] Yehuda Koren and Robert M. Bell. Advances in collaborative filtering. In Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors, *Recommender Systems Handbook*, pages 145–186. Springer, 2011.

[22] Yehuda Koren, Robert M. Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *IEEE Computer*, 42(8):30–37, 2009.

[23] Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7:76–80, 2003.

[24] Hema Mahato, Dagmar Kern, Paul Holleis, and Albrecht Schmidt. Implicit personalization of public environments using bluetooth. In *CHI '08 extended abstracts on Human factors in computing systems*, CHI EA '08, pages 3093–3098, New York, NY, USA, 2008. ACM.

[25] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. Chapter 18 - matrix decompositions and latent semantic indexing. In *Introduction to Information Retrieval*, pages 403–419. Cambridge University Press, New York, NY, USA, 2008.

[26] Kevin McCarthy, Maria Salamó, Lorcan Coyle, Lorraine McGinty, Barry Smyth, and Paddy Nixon. Cats: A synchronous approach to collaborative group recommendation. In Geoff Sutcliffe and Randy Goebel, editors, *Proceedings of the Nineteenth International Florida Artificial Intelligence Research Society Conference, Melbourne Beach, Florida, USA, May 11-13, 2006*, pages 86–91. AAAI Press, May 2006.

[27] Prem Melville and Vikas Sindhwani. Recommender systems. 2010.

[28] Douglas Oard and Jinmook Kim. Implicit feedback for recommender systems. In *in Proceedings of the AAAI Workshop on Recommender Systems*, pages 81–83, 1998.

[29] Arkadiusz Paterek. Improving regularized singular value decomposition for collaborative filtering. *Proceedings of KDD Cup and Workshop*, 2007.

[30] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: An open architecture for collaborative filtering of netnews. pages 175–186. ACM Press, 1994.

[31] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning*, ICML '07, pages 791–798, New York, NY, USA, 2007. ACM.

[32] Gerald Salton, editor. *Automatic text processing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1988.

[33] Badrul Sarwar, George Karypis, Joseph Konstan, and John Reidl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, WWW '01, pages 285–295, New York, NY, USA, 2001. ACM.

[34] Badrul M. Sarwar, 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.

[35] Toby Segaran. *Programming Collective Intelligence: Building Smart Web 2.0 Applications*. O'Reilly, Beijing, 2007.

[36] Upendra Shardanand and Pattie Maes. Social information filtering: Algorithms for automating "word of mouth". pages 210–217. ACM Press, 1995.

[37] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Major components of the gravity recommendation system. *SIGKDD Explor. Newsl.*, 9:80–83, December 2007.

[38] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Matrix factorization and neighbor based algorithms for the netflix prize problem. In *Proceedings of the 2008 ACM conference on Recommender systems*, RecSys '08, pages 267–274, New York, NY, USA, 2008. ACM.

[39] Andreas Töscher and Michael Jahrer. The bigchaos solution to the netflix prize 2008. 2008.

[40] Andreas Töscher, Michael Jahrer, and Robert M. Bell. The bigchaos solution to the netflix grand prize. 2009.

[41] Andreas Töscher, Michael Jahrer, and Robert Legenstein. Improved neighborhood-based algorithms for large-scale recommender systems. In *Proceedings of the 2nd KDD Workshop on Large-Scale Recommender Systems and the Netflix Prize Competition*, NETFLIX '08, pages 4:1–4:6, New York, NY, USA, 2008. ACM.

7

Appendix: additional experiments on group recommendation

Additional experiments were made using different variants of the k-means algorithm, as mentioned in chapter 5.

The results here presented are organized according to the number of latent factors used to perform the clustering: 2, 5, 8 or 10. We attempted to produce meaningful group recommendations for single-group, 2-group and 3-group targets, using different the k-means algorithm and different numbers of latent factors for clustering.

For each of these setups, random users were chosen and their groups were identified. To improve the quality of this evaluation session, for each setup we picked 4 users belonging to 4 different groups, so that we could attempt to produce multi-group recommendations. Groups are identified as *g1*, *g2*, *g3* and *g4*. Table 7 presents the groups involved in each of the 10 experiment setups and the number of users contained within those groups.

		2 factors			**5 factors**			**8 factors**			**10 factors**		
		k-means	swap	hybrid	k-means	swap	hybrid	k-means	swap	hybrid	k-means	swap	hybrid
#Exp	**Groups**	number of users			number of users			number of users			number of users		
1	g1	282	751	287	702	345	393	431	422	388	927	955	673
2	g2	507	714	443	753	450	1450	792	212	676	639	503	289
3	g3	309	420	767	459	475	618	801	453	738	312	523	698
4	g4	745	416	899	292	5155	1285	261	1791	417	359	276	236
5	g1+g2	789	1465	730	1455	795	1843	1223	634	1064	1566	1458	962
6	g2+g3	816	1134	1210	1212	925	2068	1593	665	1414	951	1026	987
7	g3+g4	1054	836	1666	751	5630	1903	1062	2244	1155	671	799	934
8	g4+g1	1027	1167	1186	994	5500	1678	692	2213	805	1286	1231	909
9	g1+g2+g3	1098	1885	1497	1914	1270	2461	2024	1087	1802	1878	1981	1660
10	g2+g3+g4	1561	1550	2109	1504	6080	3353	1854	2456	1831	1310	1302	1223

Table 7.1: Statistics on group recommendation experiments

The next pages will contain charts illustrating the results of the experiments made, organized by number of latent factors used for clustering. Each page will thus present results for a different number of latent factors, accordingly with the aforementioned content organization, and will contain 2 charts presenting results in different ways: (1) percentage of good group recommendations for each experiment; (2) average percentage of good group recommendations for single-group, 2-group and 3-group targets. Finally, the overall group recommendation results with different numbers of latent factors and k-means variants will be illustrated by charts as well.

Clustering with 2 latent factors

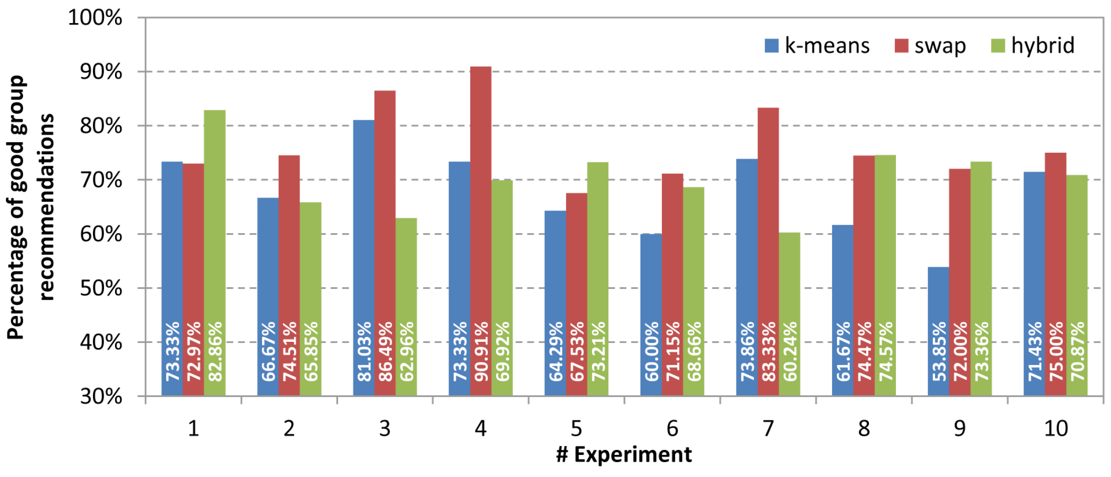


Figure 7.1: Results of group rec. with 2 factors, grouped by experiment

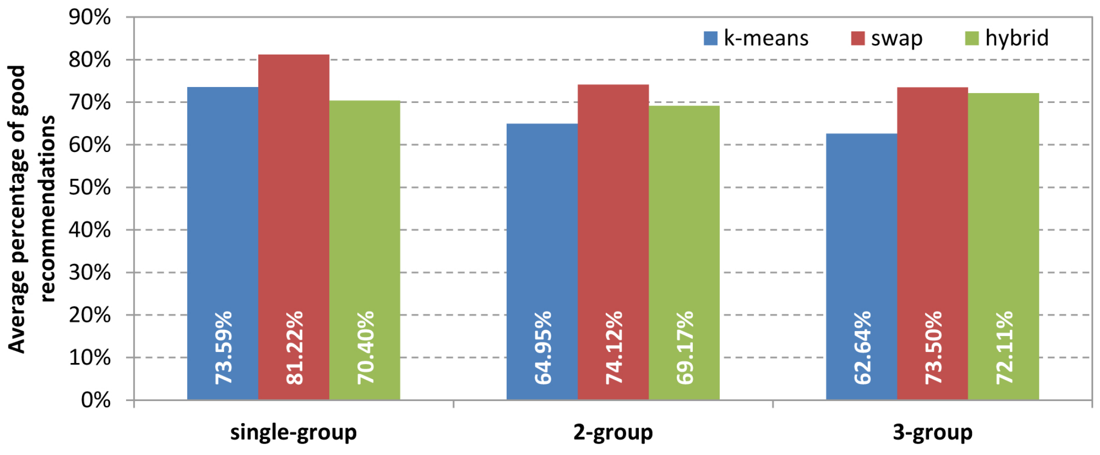


Figure 7.2: Results of group rec. with 2 factors, grouped by nr. of groups involved

Clustering with 5 latent factors

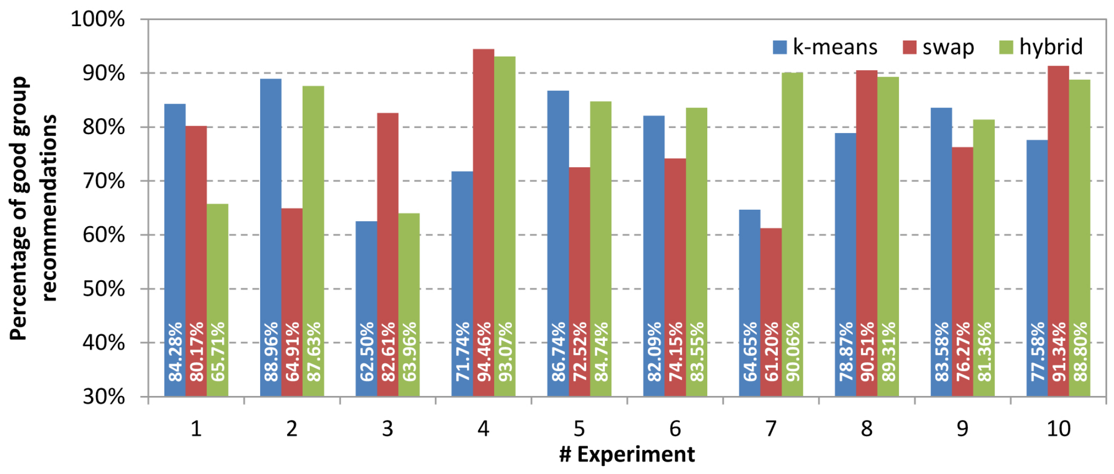


Figure 7.3: Results of group rec. with 5 factors, grouped by experiment

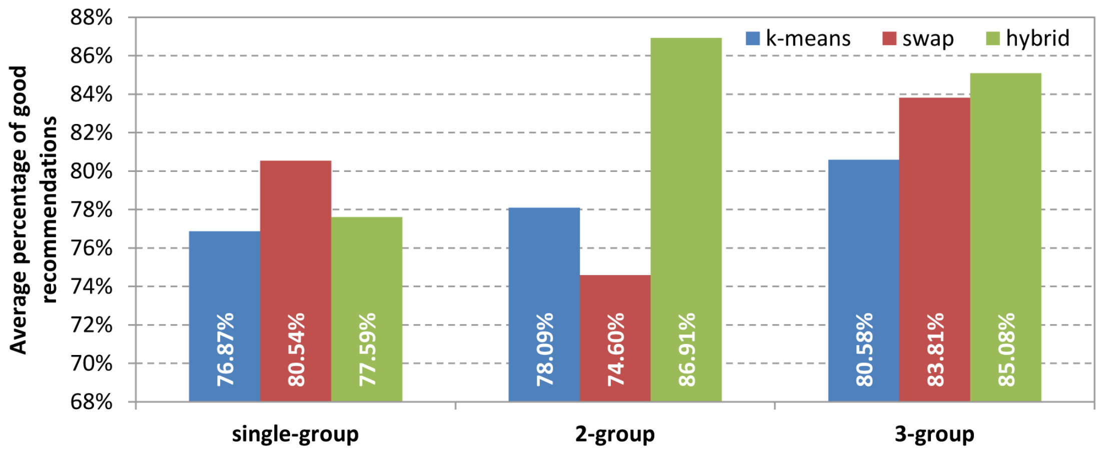


Figure 7.4: Results of group rec. with 5 factors, grouped by nr. of groups involved

Clustering with 8 latent factors

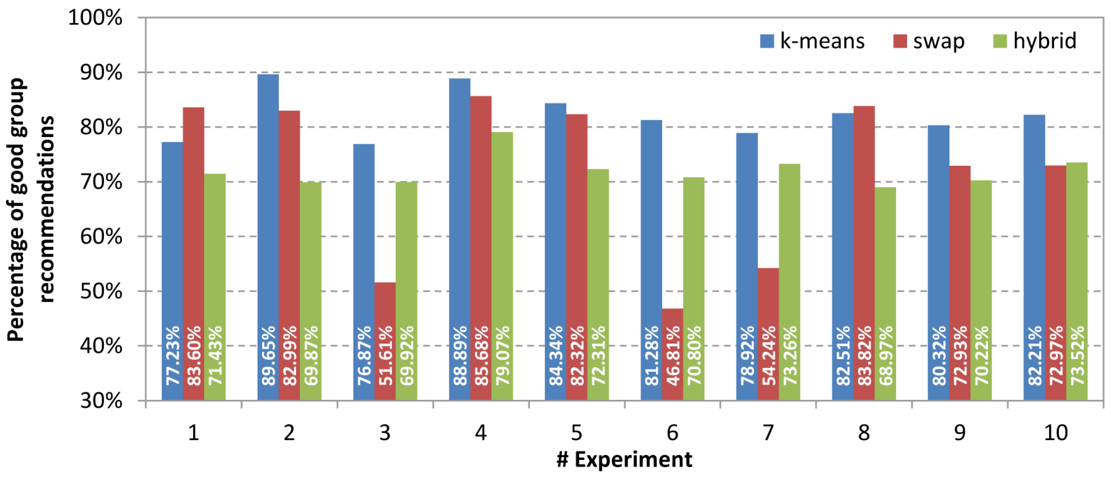


Figure 7.5: Results of group rec. with 2 factors grouped by experiment

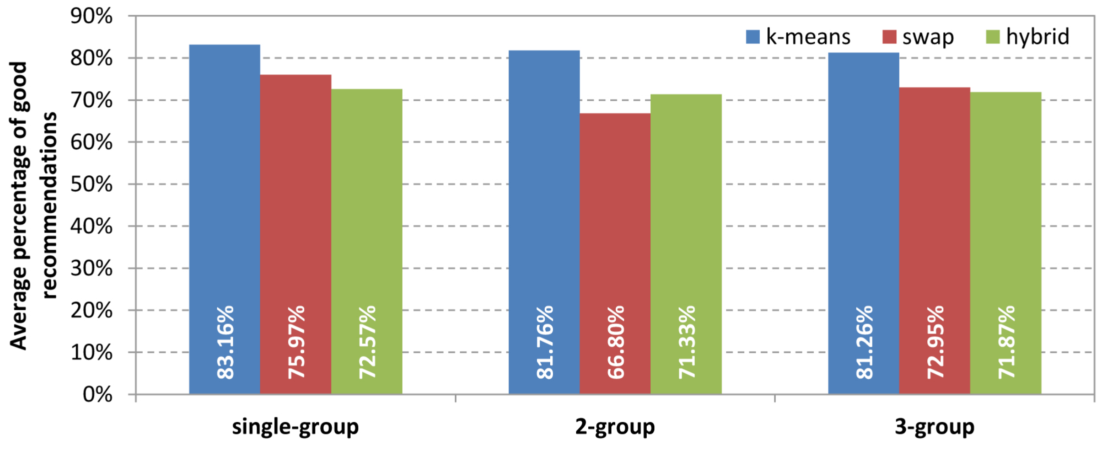


Figure 7.6: Results of group rec. with 8 factors, grouped by nr. of groups involved

Clustering with 10 latent factors

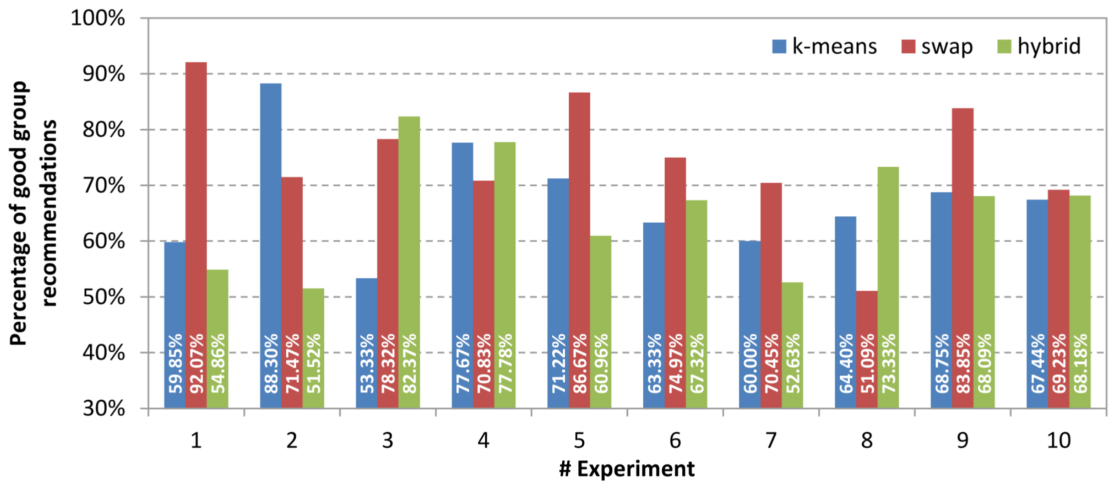


Figure 7.7: Results of group rec. with 10 factors, grouped by experiment

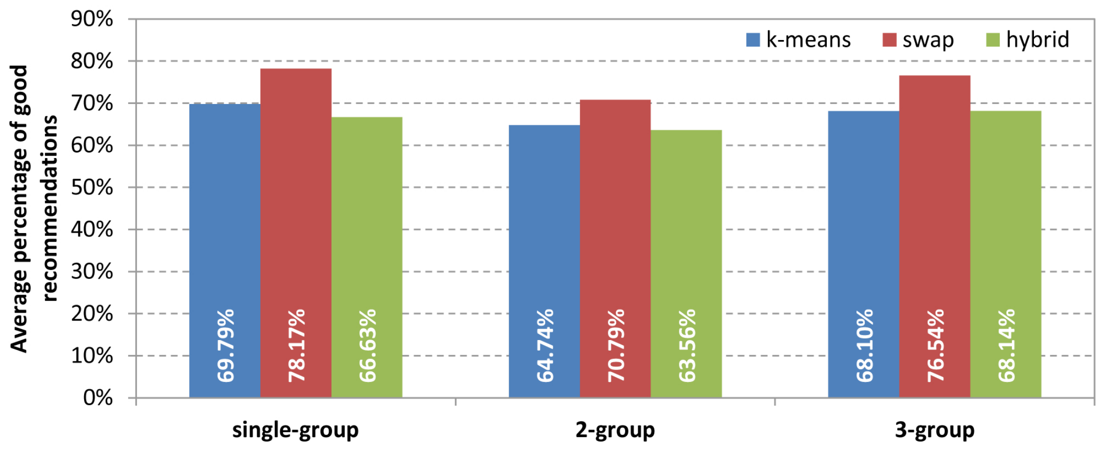


Figure 7.8: Results of group rec. with 10 factors, grouped by nr. of groups involved

Overall performance of the group recommendation framework

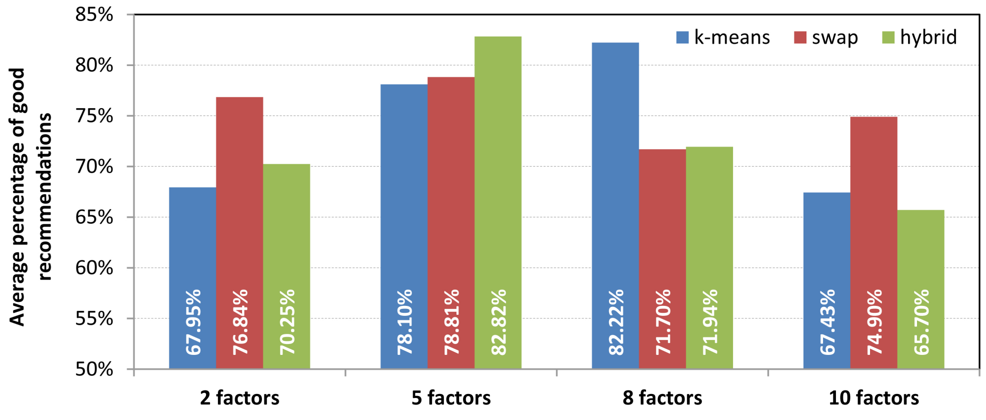


Figure 7.9: Results of group rec. with all k-means variants, grouped by nr. of factors

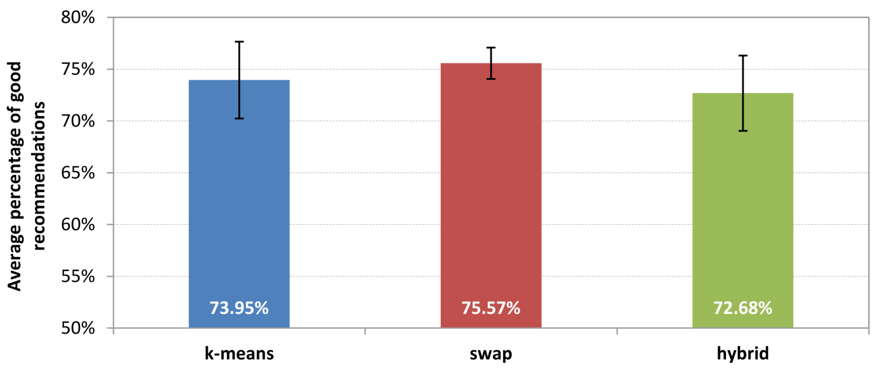


Figure 7.10: Results of group rec. with standard deviation, grouped by k-means variant

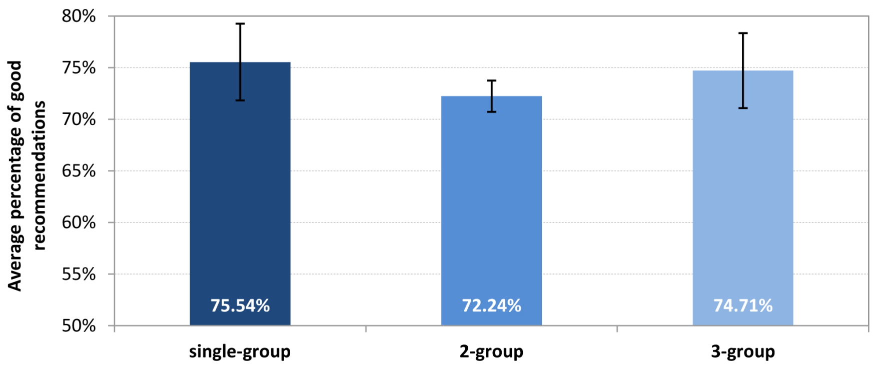


Figure 7.11: Results of group rec. with standard deviation, grouped by nr. of groups involved