Pentti Koskela

COMPARING RANKING-BASED COLLABORATIVE FILTERING ALGORITHMS TO A RATING-BASED ALTERNATIVE IN RECOMMENDER SYSTEMS CONTEXT

JYVÄSKYLÄN YLIOPISTO
TIETOJENKÄSITTELYTIETEIDEN LAITOS
2017

ABSTRACT

Koskela, Pentti
Comparing ranking-based collaborative filtering algorithms to a rating-based alternative in recommender systems context
Jyväskylä: University of Jyväskylä, 2017, 51 p.
Information Systems, Master's Thesis
Supervisor: Rönkkö, Mikko

The vast amount of content on internet services, such as e-commerce sites, can cause information overflow, which leads to a bad user experience. Recommender system is technique to support the user's decision-making by providing predicted suggestions. It is common that user is provided a list of items in user's preference, e.g. top-10 list of movies. Traditionally, these ranked lists are generated by using rating-based approaches, where ratings are predicted to unknown items which are then calculated to ranked list. Ranking-based approach calculates similarities between users and predicts a ranked list without the middle-step of predicting the ratings first.

There is a number of different collaborative filtering (CF) algorithms for different use cases. In a context of CF, ranking-based approaches are becoming more popular as the importance of ranked list accuracy has increased. However, there are several hybrid implementations where two or more different kind of recommender systems are combined, which performance cannot be compared to the algorithms in this thesis due to implementation differences.

This thesis will compare three different ranking-based CF algorithms to each other and compare the results with the rating-based CF paradigm. The results will show the prediction accuracy improvement when using ranking-based approaches compared to a rating-based one. In addition, results will also show how much the performance have been improved in ranking-based CF algorithms in the past years.

Excluding the research papers where the selected algorithms were introduced, I did not find any research publications where the selected algorithms were compared to each other. I evaluated the results using two different evaluation methods, of which Mean Average Precision is less common in this field of study.

Keywords: recommender systems, ranking-oriented collaborative filtering, rating-oriented collaborative filtering

TIIVISTELMÄ

Koskela, Pentti
Sijoitusperusteisten yhteisöllinen suodatus-algoritmien vertailu arvosanaperusteiseen vaihtoehtoon suosittelujärjestelmien kontekstissa
Jyväskylä: Jyväskylän Yliopisto, 2017, 51 s.
Tietojärjestelmätiede, Pro Gradu-tutkielma
Ohjaaja: Rönkkö, Mikko

Suuri sisältövalikoima eri internet palveluissa, kuten verkkokaupoissa, voi aiheuttaa liian suurta informaatiomäärää, mikä heikentää asiakaskokemusta. Suosittelujärjestelmät ovat teknologioita, jotka tukevat asiakkaan päätöksentekoa tarjoamalla ennustettuja suosituksia. On yleistä, että asiakkaalle näytetään lista tuotteista, joista asiakas voisi pitää, esimerkiksi top-10 lista elokuvista. Perinteisesti nämä listat ovat tuotettu käyttäen perinteistä arvosanapohjaista menetelmää, missä tuntemattomille tuotteille ennustetaan arvosana ja järjestetty lista muodostetaan arvosanojen perusteella. Sijoitusperusteinen lähestyminen laskee käyttäjien väliset samankaltaisuudet ja ennustaa järjestetyn listan ilman välivaihetta liittyen arvosanojen laskemiseen.

Erilaisia suosittelujärjestelmäalgoritmeja on julkaistu lukuisia eri käyttötarkoituksia varten. Yhteisöllisen suodatuksen kontekstissa sijoitusperusteiset menetelmät ovat yleistyneet järjestettyjen listojen tarkkuuden merkityksen kasvaessa. On olemassa useita hybridivariaatioita missä kaksi tai useampi eri suosittelujärjestelmätyyppi on yhdistetty. Näiden suorituskykyä ei voida verrata tässä tutkielmassa käytettyihin algoritmeihin johtuen niiden erilaisesta toteutustavasta.

Tämä tutkielma vertaa kolmea erilaista sijoitusperusteista yhteisöllistä suosittelujärjestelmäalgoritmia keskenään, ja vertailee tuloksia perinteisen arvosanaperusteisen algoritmin kanssa. Tulokset osoittavat parannuksen ennustustarkkuudessa sijoitusperusteista algoritmia käytettäessä, verrattuna arvosanaperusteiseen algoritmiin. Lisäksi, tulokset osoittavat sijoitusperusteisten algoritmien kehityksen parannuksen viime vuosina.

Pois lukien tieteelliset julkaisut, missä valitut algoritmit ovat esitelty, en löytänyt tutkielmaa, missä algoritmeja olisi vertailtu keskenään. Tarkastelin tuloksia käyttäen kahta eri arviointimenetelmää, joista Mean Average Precision on vähemmän käytetty tämänkaltaisissa tutkimuksissa.

Avainsanat: suosittelujärjestelmät, sijoitusperusteinen yhteisöllinen suodatus, arvosanaperusteinen yhteisöllinen suodatus

FIGURES

TABLES

TABLE OF CONTENTS

1 INTRODUCTION

The size of the Internet is about 4.7 billion pages. Besides the number of pages, amount of information on the web has increased tremendously and it is more challenging to manage. This information overload offers some serious challenges to make most of the information manageable (Wang, Sun, Gao & Ma. 2014). One of the techniques to handle this problem is recommender systems. Recommender systems have become very popular in situations where certain items ought to be addressed to a target user.

But what are recommender systems? For most people, this term does not tell anything even though majority of people are dealing with recommender systems on a daily basis. Nowadays recommender systems are used on vast amount of web services, e.g. news pages, online stores and streaming services. Suggestions provided by recommender systems supports user's decision making process (Kantor, Rokach, Ricci & Shapira, 2011, p. 6). For example: customer visits Netflix.com, the online streaming service videos, as a logged in user. From the user profile, basic demographic information like age, gender and residence can be gathered. This alone provides a possibility to recommend items according to a user's demographic profile. Finnish content for finnish user, for example. Saved browsing history, watching history and ratings user has given to products makes more accurate predictions possible. With this kind of data, we can analyze and predict what kind of items the target user likes and what should be recommended to him or her. If the user has watched only comedies, it is most likely that he or she would like to see more comedies. Perhaps the user has a favorite actor or actress, then content should be recommended accordingly. The search-function in a web site or in a service is not considered a recommender system, although it can be implemented as part of it. The behavior of recommender systems can be called 'passive' as it does not need any explicit activity from the user.

Recommender systems are extremely important business-wise. Most services have millions of items available, whether they are movies or music for streaming or physical products in online store. If there were no recommender systems to highlight to, user might feel anxious about not finding what he or she was looking for. Recommender systems are valuable for coping with the

information overload and they have become one of the most powerful tools in e-commerce (Kantor et al., 2011, p.6).

Of all recommender systems paradigms, available, this thesis' focus is on CF recommender algorithms, which can be divided into smaller segments, in this case memory-based and model-based methods. Other popular recommender systems are content-based filtering and various hybrid techniques combining these two together.

The basic idea of collaborative filtering is to utilize user-item rating matrix to make predictions and recommendations (Wang et al. 2014). CF does not need previous information about users or items what makes its implementation far easier than content-based filtering, which requires proper domain knowledge.

Recommender systems have been a popular topic for the last two-three decades and it is becoming even more important. Collaborative filtering is considered to be the first automated recommender system (Konstan & Riedl. 2012) and it is the most popular one (Herlocker, Konstan, & Riedl, 2000). It has been studied the most and there are many different versions published

When a user makes a search in e.g. Google, he or she is most interested in the results locating on top of the list and not so interested about the results below. In most cases, providing a ranked list of items in user preference order supports his or her decision-making. Ranked list can be produced by using traditional rating-oriented approach which first predicts the potential ratings a target user would assign to the items and then rank the items according to the predicted ratings (Liu & Yang, 2008). However, ranking-oriented CF algorithms can directly generate a ranked list of items for a target user without the intermediate step of rating prediction (Huang et al., 2015). This thesis' focus is in the ranking problem and the research questions for this paper are as follows:

- Why ranking-based algorithms should provide better results than traditional rating-oriented algorithms?
- Do the proposed algorithms perform better than rating-based algorithm in real-world benchmark datasets?

The contribution of this thesis will be the results of the three ranking-based CF algorithms compared to the traditional rating-based CF. These have not been compared to each other before but once, in publication of one of the algorithms. Therefore, this thesis provides more objective approach for the comparison. This thesis is divided into six chapters: Introduction, Recommender Systems, Selection of Collaborative Filtering Algorithms, Methodology, Results and Discussion.

Chapter 2: Recommender Systems, describes the recommender systems in general level. A brief background about recommender systems is explained with the information about different types of recommender systems. Since this thesis is about CF, other types of recommender systems are not explained in detail. CF is divided into memory-based (chapter 2.2) and model-based (chapter 2.3) approaches.

The ranking-based CF algorithms used in this thesis are memory-based. The more detailed explanation about the functionalities of selected algorithms are in chapter 3, which provides reader a basic understanding on how they work and differ from each other. For in-depth specifications about the selected algorithms, one should consider the research papers where they were introduced.

Chapter 4 is about the methodology about this thesis. The selected methodology is Design Science Research Methodology (DSRM). The background and details about DSRM are explained based on paper by Hevner, March, Park and Ram (2004). The implementation of DSRM for this thesis is explained. Chapter 334.1 introduces the real-world datasets that will be used in the experiment. The tool developed and used for the experiment is explained in chapter 4.2. The results are evaluated using two different evaluation methods, which are explained under chapter 4.3

The analysis of the results is divided into three subchapters following with the summary under chapter 5. Chapter 5.1 is about training the algorithm with training data and calculating the similarities. Chapter 5.2 about analyzing the runtime of prediction calculation using the test-data of the dataset. The prediction accuracy is evaluated using two different evaluation methods in chapter 5.3. Finally, chapter 5.4 concludes this chapter and the test results.

Chapter 6 concludes the thesis with key findings, contribution and limitations of the research. The discussion about how well this thesis answered the research questions is in chapter 6.1., following with the contribution in chapter 6.2. Limitations and evaluation of the research are explained in chapter 6.3. Chapter 6.4 concludes chapter 6 and the whole thesis.

2 RECOMMENDER SYSTEMS

In this chapter recommender systems are explained in general level. Chapter 2.1 provides background of recommender systems in general, provides basic knowledge for reader to understand the concept better and explaining why recommender systems are implemented. Chapters 2.2 and 2.3 provides basic level information about memory-based collaborative filtering and model-based collaborative filtering, respectively.

2.1 Background

Recommender systems are software tools and techniques that provide suggestions for target user (Kantor et al, 2011, p. 28). Recommender systems assist user in information-seeking tasks by suggesting items, e.g. products, services, information, that best suit their needs (Mahmood & Ricci, 2009). Recommender systems have become very popular and they are applied broadly in e-commerce and streaming services, like Netflix. E-commerce services might have hundreds of thousands, even millions, of products in their portfolio. While vast product portfolio is generally a good thing, customer might find itself surrounded by products not useful to him or her or worse, customer won't find the product that he or she is interested in, making customer frustrated and motivated to exit the online store. Recommender systems are used to suggest products customer is interested in, based on various types of customer data that can be gathered in several ways. Type of data gathered and used in recommendation process depends on recommender system paradigm that has been used in the situation. Nowadays recommender systems are so popular that more often than not user does not even notice using one.

In order for recommender system to function properly, it has to predict correctly the potential items user might want to see. The system must be able to predict the utility of some of the items and then decide what items to

recommend based on item comparison. (Kantor et al., 2011). If recommender system fails to do so, user sees recommendations annoying rather than useful. One example is that user gets product recommendations that suits users interest, but recommender system do not know that user have these products already, making recommendations pointless. Various user data must be gathered to avoid these kinds of situations. One option to avoid false recommendations is to implement filtering tools. Users rate items they have experienced to establish a profile of interest (Herlocker, Konstan & Riedl, 2000). If recommender system knows user's profile, shopping history and/or possible reviews user has given, it should function far more accurate. In several cases user interaction is needed also to mark products as "not relevant".

Before describing how different recommender systems work, one should know the basic terms concerning the subject. Kantor et al (2011, p. 35) describes that data used by recommender systems refers to three kinds of objects: items, users and transactions.

Items

Items are the objects that are recommended (e.g. products in online store). Items are represented by a set of features. For example, movie and TV-series recommender describes items with following features: actors, directors, genres, subject, year of production etc. The value of an item may be positive if the item is useful for the user, or negative if the item is not appropriate and the user made a wrong decision when selecting it. When a user is acquiring an item there will always incur a cost. The cost is a cognitive cost of searching the item and monetary cost of paying the item. This should be taken into consideration when implementing recommender systems into a service. There is always a cost for the user even if user is not buying it. If searching and eventually finding the item does not end up buying the item, there have been cognitive cost, thus the value of the item is negative. If the item is useful for the user and he or she will buy it, the value of the item is positive.

Users

Users are more challenging to define since everyone is an individual with individual needs and goals. In order to personalize recommender systems to user, a lot of information about the user must be gathered. User information can be structured in various ways and the selection of what information to model depends on the recommendation paradigm. For instance, in collaborative filtering user profile is basically a simple list of ratings user has provided to items while content-based filtering requires far more complex user profile in order to generate accurate predictions. Demographic recommendation uses sociodemographic attributes such as age, gender, profession and education to form a user profile.

Managing a user profile contains a lot of challenges. Once a user's profile has been established, it is difficult to change one's preferences. A meat-eater who becomes vegetarian will continue to get meat-related recommendations for some time, before preferences have changed enough. This occurs especially in

memory based collaborative filtering and content-based filtering. Many recommender systems have functions to weight older ratings to have less influence but it risks the system to lose user's long-term interests that are not in frequent enough use. (Burke, 2007.).

Transactions

Transactions are referred to a recorded interaction between a user and the recommender system. Transactions are log-like data for which purpose is to store important information during the interaction process and which are useful for the recommendation generation algorithm that the system is using. One example of transaction data is rating that user has given to a certain item. Ratings are in fact the most popular form of transaction data. Ratings can be collected either explicitly or implicitly. The explicit collection relates to situation where the user is asked to provide an opinion about an item on a rating scale. Ratings can take a variety of forms:

- Numerical ratings e.g. 1-5 stars
- Ordinal ratings, such as "strongly agree, agree, neutral, disagree, strongly disagree"
- Binary ratings where user is asked to decide if a certain item is good or bad
- Unary ratings that can indicate if a user has observed or purchased an item. For example, browsing behavior or reading an article (staying on one page for a certain amount of time) is a form of unary rating.

Recommender systems as a research area is relatively new. Earliest scientific publications about recommender systems are from early 1990s (Konstan & Riedl, 2012). The interest in recommender systems has increased significantly in recent years. Kantor et al (2009, p. 30) point out facts to indicate the rising popularity of recommender systems as a research area. Few of these mentions are listed as follows:

- Recommender systems play an important role in such highly rated Internet sites as Amazon.com, YouTube, Netflix, Yahoo, TripAdvisor, Last.fm, and IMDb
- There are dedicated conferences and workshops related to the field. For example, ACM Recommender Systems (RecSys), established in 2007. Sessions dedicated to RSs are frequently included in the more traditional conferences in the area of data bases, information systems and adaptive systems.
- At institutions of higher education around the world, undergraduate and graduate courses are now dedicated entirely to RSs; tutorials on RSs are very popular at computer science conferences; and recently a book introducing RSs techniques was published.

- There have been several special issues in academic journals covering research and developments in the RS field.

Two most popular recommender system types are called collaborative filtering and content-based filtering. In addition, there are also demographic filtering and knowledge-based filtering. There are also hybrid variations, combining two or more of these paradigms. Collaborative filtering is considered to be the first automated recommender system (Konstan & Riedl, 2012) and it is the most popular and widely implemented recommendation technique (Kantor et al., 2011). The very first recommender system, called Tapestry, was based on collaborative filtering and was designed to recommend documents drawn from newsgroups to a collection of users (Goldberg, Nichols, Obi & Terry, 1992). CF predicts item recommendations to the user based on collected information about item ratings user has provided, and then comparing this information to peer users rating-data (Herlocker, Konstan, Terveen & Riedl, 2004). User's rating data is compared to other users' data, and by finding a user with similar tastes with the target user, CF can predict items for the target user. The assumption is that a user would be interested in those items preferred by other users with similar interests (Liu & Yang, 2008). CF brings together the opinions of large interconnected communities on the web (Schafer et al., 2007). In other words, CF is based on "wisdom of the crowd". In this process, CF uses the neighborhood approach, which focuses on relationships between items or between users (Kantor et al., 2011, p. 146)

One of the benefits CF has compared to other techniques is its ability to recommend items regardless of the type or content, what makes it practical in various applications. However, there are some properties that needs to be fulfilled to get CF function properly. Schafer et al. (2007) lists following required properties in table 1:

TABLE 1 :Required properties to get CF function properly (adapted from Schafer et al. 2007)

Feature	Explanation
There are many items	If there are few items to choose from, the user can learn about them all without need for computer support.
There are many ratings per item	If there are few ratings per item, there may not be enough information to provide useful predictions or recommendations.
There are more users rating than items to be recommended	A corollary of the previous paragraph is that often you will need more users than the number of items that you want to be able to capably recommend. As an example, with one million users, a CF system might be able to make recommendations for a hundred thousand items, but may only be able to make confident predictions for ten thousand or fewer, depending on the distribution of ratings across items. The ratings distribution is almost always very skewed: a few items get most of the ratings, creating a long tail of items that get few ratings. Items in this long tail will not be confidently predictable.

Users rate multiple items	If a user rates only a single item, this provides some information for summary statistics, but no information for relating the items to each other.

There are several ways to categorize different CF methods. Another policy is to split CF techniques into memory-based and model-based methods. Memory-based methods include both item-based and user-based CF methods and this is widely used for e.g. e-commerce sites, often with domain-specific variations (Su & Khoshgoftaar, 2009). Chapters 2.2 and 2.3 will provide more detailed information about memory-based and model-based CF.

2.2 Memory-based Collaborative Filtering

Memory-based CF (also called as neighborhood-based or heuristic-based) is considered one of the most popular recommendation approaches. (Kantor et al., 2011, p.111). Memory-based CF can be divided into item-based or user-based methods. In user-based-methods focus for CF is to predict user similarities where item-based methods rely on item similarities (Jian & Qun, 2012). Popular memory-based technique is to use nearest-neighbor methods, which means searching for most similar user to a target user from a set of users. This is popular due to their simplicity, efficiency and ability to produce accurate and personalized recommendations (Kantor et al., 2011, p.107). User-item ratings stored in the system are directly used to predict ratings for new items.

This chapter is divided into four sub-chapters: Pearson Correlation Coefficient, Vector Space Model, Ranking-based Collaborative Filtering and Advantages and Drawbacks. Pearson Correlation Coefficient in chapter 2.2.1 is so common way to calculate similarities that it is good to explain in its own chapter. Vector Space-model is explained in chapter 2.2.2 and it is used in one of the algorithms by Wang, Su, Gao & Ma (2014). Ranking-based Collaborative Filtering in general is explained in chapter 2.2.3. The advantages and drawbacks of memory-based CF is discussed in chapter 2.2.4.

2.2.1 Pearson Correlation Coefficient

One way to measure similarities is to use Pearson correlation coefficient. Pearson coefficient calculates the correlation between target user and its neighboring users. For example, Eric has rated four items out of five. User-Item matrix (Table 2) contains ratings from three other users as well. With this rating data, it is possible to predict whether Eric likes movie "Titanic" or not, using Pearson correlation coefficient, and what rating she will most likely provide. In Pearson correlation coefficient, the result is covariance value between [-1,1], value 1 representing complete dependence.

u,v: users
$r_{u,i}$: user u rating for item i
I_{uv}: items rated by both u and v

$$PC(u,v) = \frac{\sum_{i \in I_{uv}} (r_{ui} - \bar{r}_u)(r_{vi} - \bar{r}_v)}{\sqrt{\sum_{i \in I_{uv}} (r_{ui} - \bar{r}_u)^2 \sum_{i \in I_{uv}} (r_{vi} - \bar{r}_v)^2}}$$

TABLE 2 User-Item matrix (Kantor et al., 2011, p.126)

	The Matrix	Titanic	Die Hard	Forrest Gump	Wall-E
John	5	1		2	2
Lucy	1	5	2	5	5
Eric	2	?	3	5	4
Diane	4	3	5	3	

We can see that Eric's taste is the most similar to Lucy's, since both loved "Forrest Gump" and neither liked "The Matrix". It would seem like Eric would like "Titanic" because Lucy rated it for 5, but there might be differences in their taste in movies. Perhaps Lucy likes more drama movies than Eric. It is important to notice that table this small works well as an example but not well in reality. Neighbors with tiny samples, three to five co-rated items, are proved to be terrible predictors for the active user (Herlocker, Konstan & Riedl, 2002). In order to get accurate predictions, table size should be between 20 to 50 users, minimum (Herlocker, Konstan & Riedl., 2000). Nearest-neighbor algorithms trust that user's interests and tastes stays the same in the future than they are at the moment. However, there are systems that weight rating values depending how old ratings are, offering possibility to update user profile as time goes by.

User-based neighborhood recommendation methods predict the rating r_{ui} of a user u for a new item i using the ratings given to i by users most similar to u, called nearest-neighbors (Kantor et al., 2011, p.138). By using Pearson Correlation Coefficient, we can calculate user similarities. In table 3, we can see that covariance between Eric and Lucy is highest, meaning their taste in movies is the most similar.

TABLE 3 User-based Pearson correlation (Kantor et al., 2011, p.126)

	John	Lucy	Eric	Diane
John	1.000	-0.938	-0.839	0.659
Lucy	-0.938	1.000	0.922	0.994
Eric	-0.839	0.992	1.000	-0.659
Diane	0.659	-0.787	-0.659	1.000

Item-based recommendation relies on the ratings given to similar items. If we look at table 4, you can notice that people who liked "Forrest Gump" and "Wall-E" also liked "Titanic" and, obviously, vice versa. Since Eric liked

"Forrest Gump" and "Wall-E" (Table 2), we can presume that he would also like "Titanic". It is notable that these recommendations do not consider any domain knowledge e.g. genre, director, actors.

TABLE 4 Item-based Pearson correlation (Kantor et al., 2011, p.126)

	The Matrix	Titanic	Die Hard	Forrest Gump	Wall-E
Matrix	1.000	-0.943	0.882	-0.974	-0.977
Titanic	-0.943	1.000	-0.625	0.931	0.994
Die Hard	0.882	-0.625	1.000	-0.804	-1.000
Forrest Gump	-0.974	0.931	-0.804	1.000	0.930
Wall-E	-0.977	0.994	-1.000	0.930	1.000

Pearson Correlation Coefficient is only one from many similarity measurement methods. Mean Squared Difference, Spearman Rank Correlation have also been used, to name a few (Kantor et al., 2011. p. 127). The used data (e.g. size, type) has some significance which measurement is the most suitable for target application. However, Herlocker et al. (2002) note that Pearson Correlation Coefficient is the most accurate technique for computing similarity.

2.2.2 Vector Space Model

The vector space model is a standard algebraic model commonly used in information retrieval and it has been used in many applications, such as image processing, recommender systems, spam detection and song sentiment classification. It has been used in both collaborative-filtering and content-based filtering approaches. In content-based filtering, the descriptive user profiles can be reflected as documents and the vector space model can be applied to make recommendations based on user similarity. In rating-based CF, the vector space model can be used to transform vectors of users from the user space into the item space, and the similarity between users and items can be measured using cosine similarity. (Wang et al. 2014). User-user or item-item similarity can also be measured using vector similarity. The idea about vector similarity is to view each user as a vector in a high dimensional vector space based on user's ratings. The similarity between two vectors is calculated from the cosine of the angle of these vectors, which is a standard measure estimating pairwise document similarity in the vector space model (Liu & Yang, 2008, Wang et al. 2014).

$$S_{u,v} = \frac{\sum_{i \in I_u \cap I_v} r_{u,i} \cdot r_{v,i}}{\left[\sum_{i \in I_u \cap I_v} r_{u,i}^2 \sum_{i \in I_u \cap I_v} r_{v,i}^2\right]^{1/2}}$$

When calculating item-item similarity, the adjusted cosine similarity has proven to be the most effective (Liu & Yang, 2008, p.85). In item-item Vector Similarity each user's rating on an item is adjusted by user's mean rating.

$$S_{i,j} = \frac{\sum_{u \in U_i \cap U_j} (r_{u,i} - \overline{r_u})\ (r_{u,j} - \overline{r_u})}{\left[\sum_{u \in U_i \cap U_j} (r_{u,i} - \overline{r_u})^2\ \sum_{u \in U_i \cap U_j} (r_{u,i} - \overline{r_u})^2\right]^{1/2}}$$

2.2.3 Ranking-based Collaborative Filtering

This chapter explains the basic concept of ranking-based CF. The more detailed description about ranking-based CF can be obtained in chapter 3 where four different ranking-based CF algorithms are explained and hence work as an example.

In rating-based CF approaches, ranked lists are produced from rating predictions first and generating ranked list from those predictions. The problem in this approach along with weaker performance is that higher accuracy in item rating does not necessarily lead to better ranking effectiveness. This can be explained with following simple example. There are two items, *i* and *j* with true ratings of 3 and 4 respectively. Two different methods have predicted the ratings for *i* and *j* to be {2, 5} and {4, 3} respectively. As one can see, there is no difference between these two sets of predictions in terms of rating prediction accuracy since all the values are one unit away from the correct rating. The difference between these two predictions is that using the prediction {4, 3} will put items *i* and *j* in incorrect order while {2, 5} puts items in correct order.

Liu & Yang (2008) note that the problem in rating-oriented CF is focusing on approximating the ratings instead of rankings, which is a more important goal for recommender systems. In addition, most existing methods predict the ratings for each individual item independently instead of considering the user's preferences regarding pairs of items.

The idea behind ranking-based CF is to produce an ordered list of Top-N recommended items where the highest ranked items are predicted to be most preferred by the user. Assumption is that the user examines the items in the list starting from the top positions. (Liu & Yang, 2008.). It is easy to understand and accept this assumption if one thinks user behavior while e.g. making web-searches, browsing media-streaming services like Netflix or browsing e-commerce sites with tens of thousands of articles.

Ranking-based CF is considered to be more effective recommender system than rating-based CF since it has no need to calculate the ratings of the items for target user first. The popularity of ranking-based CF has increased over the years. Building of recommendation problems is changing away from rating-based to ranking based (Wang et al., 2014). Like rating-based CF, also ranking-based CF has multiple different variations on how to implement the process. Common for all ranking-based CF is that they are able to capture the preference

similarity between users even if their rating scores differ significantly. First ranking-based CF is considered CoFiRank (Weimer et al, 2007). CoFiRank uses maximum margin matrix factorization to optimize ranking of items for CF. EigenRank, that is included in empirical part of this research, was introduced in 2008 by Liu and Yang. EigenRank measures the similarity between users using Kendall tau rank correlation for neighborhood selection, much like VSRank which function is also explained later in this paper.

The rankings are derived from the rating matrix. Popular method to calculate similarity between users u and v is Kendall Tau rank correlation coefficient (Wang et al., 2014). Kendall tau rank correlation coefficient is a non-parametric statistic tool to measure correlation between two ordinal scale values. In case of ranking-based CF, the values are the two rankings from users u and v on their common item set:

$$\tau_{u,v} = \frac{N_c - N_d}{\frac{1}{2} N(N-1)}$$

where N_c are the numbers of the concordant pairs and N_d are the numbers of discordant pairs. *1/2N(N-1)* represents the total number of pairs. The value of Kendall tau is between [-1, 1].

2.2.4 Advantages and Drawbacks

Kantor et al. (2011, p. 135) have listed main advantages for memory-based CF as follows:

TABLE 5 : Main advantages for memory-based CF (Kantor et al., 2011, p. 113)

Attribute	Explanation
Simplicity	Neighborhood-based methods are intuitive and relatively simple to implement. In their simplest form, only one parameter (the number of neighbors used in the prediction) requires tuning
Justifiability	Such methods also provide a concise and intuitive justification for the computed predictions. This helps to provide recommendation transparency and hence, more trust. For example, in item-based recommendation, the list of neighbor items, as well as the ratings given by the user to these items, can be presented to the user as a justification for the recommendation.
Efficiency	One of the strong points of neighborhood-based systems is their efficiency. Unlike most model-based systems, they require no costly training phases, which need to be carried out at frequent intervals in large commercial applications. While the recommendation phase is usually more expensive than for model-based methods, the nearest-neighbors can be pre-computed in an offline step, providing near instantaneous recommendations. Moreover, storing these nearest neighbors requires very little memory, making such approaches scalable to applications having millions of users and items.
Stability	Another useful property of recommender systems based on this approach

	is that they are little affected by the constant addition of users, items and ratings, which are typically observed in large commercial applications. For instance, once item similarities have been computed, an item-based system can readily make recommendations to new users, without having to re-train the system. Moreover, once a few ratings have been entered for a new item, only the similarities between this item and the ones already in the system need to be computed

Investigations have shown that model-based approaches (e.g. latent factor model) are better than memory-based methods in prediction accuracy (Koren, 2008). However, good prediction accuracy alone does not guarantee users an effective and satisfying experience (Good et al., 1999). In fact, a very important role in appreciation of users for the recommender system is serendipitous recommendations (Good et al., 1999). For instance, a huge fan of Star Wars-saga does not get excited about movie recommendations of Star Wars-movies. Serendipitous recommendations help user find an interesting item that would not have otherwise been discovered.

Memory-based CFs one drawback is its poor scalability to larger systems, since managing CF requires managing large data tables, which tend to be inefficient. For example, big e-commerce providers could have millions of users and items and CF provides predictions based on user-item matrix. Memory-based CF also focuses more on recommending the most popular items since they have more rating-information available. (Sarwar, Karypis, Konstan & Riedl, 2001.). Almost all practical algorithms use some form of pre-processing to reduce run-time complexity and to help memory-based CF to scale better (Schafer et al., 2007).

Liu & Yang (2008) state that there are several difficulties when user-user approach has been selected for measuring. Firstly, raw ratings may contain biases, meaning that users tend to rate items differently. For example, some users may tend to give high ratings for most of the items. This can be corrected by using data normalization or centering the data prior to measuring user similarities, for example. Secondly, user-item ratings data is typically highly sparse. This challenges the system to find highly similar neighbors for creating accurate predictions. To fix the data, unknown ratings in the user-item matrix must be handled.

Another common CF related problem is the cold start issue. CF requires rating data in order provide predictions. When a new item is added to the system, it doesn't have any rating info, naturally. This item can't be recommended to anyone before it gets enough ratings. Same problem bothers new user. CF can't generate user profile before target user has given enough ratings for items. In other words, new user doesn't have neighborhood of similar users. There are some solutions that help reduce the cold start issues (Schafer et al., 2007.):

- have the user rate some initial items before they can use the service

- display non-personalized recommendations (e.g. most popular items in the service) until the user has rated enough items
- ask the user to describe their taste in aggregate, e.g., "I like science fiction movies"
- ask the user for demographic information
- using ratings of other users with similar demographics as recommendations

2.3 Model-based Collaborative Filtering

Memory-based CF algorithms maintain a database of all users' ratings for all items and each prediction performs computation across the entire database. Model-based algorithms first compile the user's preferences into a descriptive model of users, items, and ratings. Recommendations are generated by appealing to the model. Model-based approach may offer added value beyond its predictive capabilities by highlighting certain correlations in the data. Prediction can be calculated quickly once the model is generated and it doesn't require as much performance as memory-based CF. However, complexity to compile the data into a model may be challenging and adding new item may require a full recompilation. (Pennock, Horvitz, Lawrence & Giles, 2000.)

Model-based CF uses probabilistic models to predict rating values of unobserved items for the target user. These probabilistic models used in model-based CF are for example cluster models and Bayesian Network model. (Breese, Heckerman & Kadie, 1998.). The Bayesian network model, the most popular probabilistic model (Schafer et al., 2007), is explained in chapter 2.3.1. Cluster model is briefly described in chapter 2.3.2. Model-based approach is not discovered more specifically as none of the selected algorithms represent model-based CF.

2.3.1 Bayesian-Network Model

Bayesian-network model derives probabilistic dependencies among users or items. Breese et al. (1998) describes a method for deriving and applying Bayesian networks using decision trees to compactly represent probability tables. A decision tree (FIGURE 1) shows that users who do not watch "Beverly Hills 90210" are very likely to not watch "Melrose Place". There is a separate tree for every recommendable item. The branch chosen at a node in the tree is dependent on the user's rating for a particular item. Every node stores a probability vector for user's ratings of the predicted item. (Schafer et al., 2007.) Bayesian-network is proven to be more scalable compared to memory-based CF methods but prediction accuracy is weaker. In addition, model learning and updating is considered expensive if the number of users is large.

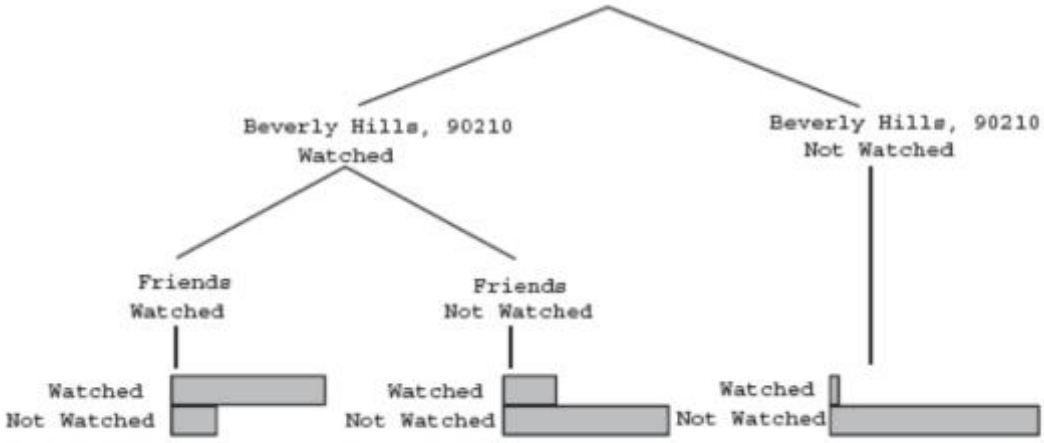

FIGURE 1 A decision tree regarding whether a user watches" Melrose Place". (Breese et al., 1998)

2.3.2 Cluster models

Cluster models provide facilitation to CF scaling problem by clustering items in groups." Clustering consists of assigning items to groups so that the items in the same groups are more similar than items in different groups: the goal is to discover natural (or meaningful) groups that exist in the data" (Kantor et al., 2011, p.86). Clustering is often considered as an intermediate step, for example to help memory-based CF scale better. The actual prediction is then made using e.g. Pearson correlation coefficient. With clustering models, the recommendation process can be done in smaller parts (clusters) rather than the entire database, improving performance. This complex and expensive clustering computation is run offline. Cluster model's recommendation quality is generally low. The quality can be improved by using numerous fine-grained segments. However, the online user-segment classification could become almost as expensive as just using memory-based CF only (Su & Khoshgoftaar, 2009.).

3 SELECTION OF COLLABORATIVE FILTERING ALGORITHMS

Algorithms applied and used in this paper are traditional rating-oriented CF, EigenRank (Liu & Yang, 2008), VSRank (Wang et al., 2014) and ListCF (Huang et al., 2015). These articles are the main sources for these algorithms in their own chapters. If not separately mentioned, the text refers to these publications accordingly.

3.1.1 Rating-oriented CF

Traditional collaborative filtering algorithms are based on predicting the potential ratings that a user would assign to the unrated items. Liu and Yang (2008) divide traditional CF into two classes. In the first class, user is presented with one individual item at a time along with a predicted rating that indicates the user's potential interest in the item. The second class produces an ordered list of Top-N recommended items. The items are ranked and highest-ranked items are most preferred by the user. The user is expected to browse predicted items from top of the list heading downwards. The latter one appears to be more popular, at least in e-commerce (Liu & Yang, 2008).

The computation of Top-N item list is generated using a rating-oriented approach. Rating-oriented approach first predicts the potential ratings a target user would assign to the items and then rank the items according to the predicted ratings. (Liu & Yang, 2008). Liu and Yang (2008) present a solid example about ranking effectiveness in rating-oriented approach:

> Suppose we have two items i and j for which the true ratings are known to be 3 and 4 respectively and two different methods have predicted the ratings on i and j to be {2, 5} and {4, 3} respectively. In terms of rating prediction accuracy as measured by the absolute deviation from the true rating, there is no difference between the two sets of predictions. However, using the predictions {4, 3}, item i and j will be incorrectly ordered while the predictions {2, 5} ensures the correct order.

Rating-oriented CF approach focuses on approximating the ratings rather than the rankings. The similarity measures in rating-oriented CF are e.g. Pearson Correlation Coefficient and Vector Similarity (see Chapter 2.2.1. and 2.2.2).

3.1.2 EigenRank

EigenRank represents one of the first ranking-based CF algorithms and it is the oldest ranking-based recommendation algorithm presented in this paper. EigenRank's main contribution can be considered similarity measure, greedy order algorithm and random walk algorithm. This chapter presents the main idea behind EigenRank and thus the only reference is the article from Liu and Yang (2008). EigenRank challenges the problem in rating-oriented CF about computing Top-N item list by using a ranking-based approach.

EigenRank approach describes a user-user similarity measure, which is based on two users' preferences over the items and after that, present two methods for ranking items based on the preferences of the set of neighbors of the target user. These methods are greedy order algorithm and random walk model. EigenRank is not trying to predict user's ratings on unrated items which is an intermediate step in traditional rating oriented CF. User-user similarity measurement using Pearson Correlation Coefficient or Vector Similarity are rating-based measures and hence, not ideal for ranking-based algorithms. In ranking-based algorithms, the similarity between users is determined by their ranking of the items.

Since the goal for EigenRank, and all other ranking-based algorithms, is to produce a ranking of the items for a user and not to predict the rating values first, user's preference function needs to be evaluated. Liu & Yang (2008) use the following form to model a user's preference: Ψ: I x I $\rightarrow \mathbb{R}$, where $\Psi(i,j) > 0$, meaning that item i is more preferable to j for user u and vice versa. The magnitude of this function $|\Psi(i,j)|$ explains the strength of preference between items i and j, value zero meaning no preference between these two items. If user u's rates items i and j with values 5 and 3 respectively, it indicates that $\Psi(i,j) > 0$ and $\Psi(j,i) < 0$. This leads to a broader definition of preference function where the basic idea is the same as in neighborhood-based collaborative filtering, where we need to find users with similar preferences to the target user. In the following formula, the set of users is noted N_u:

$$\Psi(i,j) = \frac{\sum_{v \in N_u^{i,j}} S_{u,v} \cdot (r_{v,i} - r_{v,j})}{\sum_{v \in N_u^{i,j}} S_{u,v}}$$

The more often the users in N_u assign i a higher rating than j, the stronger the evidence is for $\Psi(i,j) > 0$ and $\Psi(j,i) < 0$. The summation of $N_u^{i,j}$ is the set of neighbors of u who have rated both items i and j.

This preference function presented above assigns a score to every pair of items $i, j \in I$. The goal is to choose a ranking of items in I that approves with the

pairwise preferences defined by Ψ as much as possible. Value function $V^{\Psi}(p)$ is defined such as p is a ranking of item in I when $p(i) > p(j)$ if and only if i is ranked higher than j. Value function measures how consistent is the ranking p with respect to the preference function Ψ as follows:

$$V^{\Psi}(p) = \sum_{i,j:p(i)>p(j)} \Psi(i,j)$$

EigenRank value function suggests that solving the ranking problem requires searching through the optimal ranking p^* that maximizes the function. Finding the optimal ranking p^* is a NP-complete problem, meaning that the resolution time for this problem is exponential.

Ranking-based filtering needs a way to rank items without ratings and that is why preference functions are used. It is challenging to obtain preference information about items that the target user has not yet rated. EigenRank explains two approaches to solve the item-ranking problem. First one is called Greedy Order Algorithm and second one is named Random Walk Model. Greedy Order Algorithm is explained in FIGURE 2:

Algorithm 1 Greedy Order

INPUT: an item set I; a preference function Ψ

OUTPUT: a ranking $\hat{\rho}$

1: **for** each $i \in I$ **do**
2: $\pi(i) = \sum_{j \in I} \Psi(i,j) - \sum_{j \in I} \Psi(j,i)$
3: **end for**
4: **while** I is not empty **do**
5: $t = \arg\max_{i \in I} \pi(i)$
6: $\hat{\rho}(t) = |I|$
7: $I = I - \{t\}$
8: **for** each $i \in I$ **do**
9: $\pi(i) = \pi(i) + \Psi(t,i) - \Psi(i,t)$
10: **end for**
11: **end while**

FIGURE 2 Greedy Order algorithm (Liu & Yang, 2008)

Each item $i \in I$ have a potential value $\pi(i)$. Second line of the algorithm indicates that the more items that are less preferred than i (i.e. $\Psi(j,i) > 0$) the higher the potential of i. The item t evaluated in line five presents the current highest ranked item. The Greedy Order algorithm picks the item t and assigns a rank to it, which is equal to the number of remaining items in I so that it will be ranked above all the other remaining items (line six). Item t is then deleted from items in I and the potential values of the remaining items are updated by

removing the effects of *t*. Greedy order algorithm has a time complexity of $O(n^2)$, where *n* denotes the number of items.

Since Greedy Order algorithm is not considered as effective way, another approach is presented in the paper. Random Walk model is based on the stationary distribution of Markov chain and it is ultimately close to PageRank, introduced by Brin and Page (2012; originally presented in 1998). Markov chain is used as a base for Random Walk since it is effective for aggregating partial and incomplete preference information from many users. Random Walk model used in EigenRank derives implicit links between items based on the observed preference information so that a less preferred item *j* would link to a more preferred item *i* and the transition probability $p(i|j)$ would depend on the strength of the preference which can be told from the value $\Psi(i,j)$. For example, a user is trying to find his or her favorite item (e.g. a movie) and he or she has some preferences over the items. The user first chooses an item *j* randomly and based on the preferences, he or she would switch to another item *i* based on the conditional probability $p(i|j)$ which would be higher for items that are more preferred than *j* and naturally lower for items that are less preferred than item *j*. Eventually the user should select his or her favorite item most often, explaining how the stationary distribution could be used to rank items based on preferences. The transition probability $p(i|j)$ of switching to another item *j* given the current item *i* is defined in next figure:

$$p(j \mid i) = \frac{e^{\Psi(j,i)}}{\sum_{j \in I} e^{\Psi(j,i)}}$$

3.1.3 VSRank

VSRank is a ranking-based recommender system approach introduced in 2014 by Wang et al. Their article is the main source for this chapter since this is a summary of their publication. Idea behind VSRank is to improve recommendation accuracy for ranking-based CF by adapting vector space model and considering each user as a document and user's pairwise relative preferences as terms. Vector space model have been implemented in content-based filtering but it has not been investigated before in the context of CF. The terms are weighted using degree-specialty weighting scheme that in this case is TF-IDF (term frequency-inverse document frequency) resemblance. After users are represented as vectors of degree-specialty weights, ranking-based CF techniques are adopted for making recommendations for a given user.

Conventional ranking-based algorithms, e.g. EigenRank, are based on similarity measures between two users on the same set of items. These algorithms treat pairwise relative preferences equally, without considering any weighting scheme for preferences in similarity measures. That is, if two users give ratings for same items but other user has given bigger rating for item he or she liked more, users' preferences are different although both liked the same

item over the other. Weighting is one of the techniques used in data analysis and machine learning that eliminates irrelevant, redundant and noisy data.

The key component in VSRank is the degree-specialty weighting scheme. Wang et al. provide following simple example to demonstrate why relative preferences need weighting; there are two users, u and v, and both have rated two items $\{i_1, i_2\}$ to be {5,1} and {2,1}, respectively. Ratings indicate that both users have rated item i_1 higher than i_2 and the scores show that user u prefers i_1 over i_2 more strongly than user v does. Stronger preferences with larger score differences should be noted while creating recommendations. Stronger the score difference between two items means larger degree between them. This degree between two comparable items resembles *term frequency* (TF) in TF-IDF scheme. High degree for a preference term from a user can be understood in a way the user frequently confirms his or her preference.

Since TF-IDF is originally a numerical statistic tool that reflects how important a word is to a document, it is naturally not suitable for CF out-of-the-box. One of the problems TF has is that in original use the value is textual and undirectional. When implementing TF-IDF in CF, TF-value is directional. This means it has a value to compare to, which is its opposite preference. Instead of a literal word for word translation, IDF in VSRank measures the rarity of the preference in users who hold the same or the opposite preferences on the same items.

Vector space model is a standard algebraic model commonly used in information retrieval. Vector space model treats a textual document as a bag of words. Each document is represented as a vector of TF-IDF weights. Cosine similarity is used to compute similarity between document vectors and query vectors. Larger the similarity, higher the relevancy.

ALGORITHM 1: The VSRank Framework

Input: An item set I, a user set U, and a rating matrix R
Output: A set of rankings $\{\tau_u\}_{u \in U}$ of items for each user $u \in U$

1 $T \leftarrow \varnothing$;
2 **for each** $u \in U$ **do**
3 $\quad T_u \leftarrow \texttt{ExtractTerms}\ (u, I, R)$;
4 $\quad T \leftarrow T \cup T_u$;
5 **end**
6 **for each** $t \in T$ **do**
7 $\quad w_t^{(S)} \leftarrow \texttt{ComputeSpecialty}\ (T)$; // Eq. 7
8 **end**
9 **for each** $u \in U$ **do**
10 $\quad$ **for each** $t \in T_u$ **do**
11 $\quad\quad w_{u,t}^{(D)} \leftarrow \texttt{ComputeDegree}\ (T_u)$; // Eq. 5
12 $\quad\quad w_{u,t} \leftarrow w_t^{(S)} \times w_{u,t}^{(D)}$; // Eq. 8
13 $\quad$ **end**
14 **end**
15 **for each** $u \in U$ **do**
16 $\quad$ **for each** $v \in U$ **and** $u \neq v$ **do**
17 $\quad\quad s_{u,v} \leftarrow \texttt{ComputeSimilarity}\ (w_u, w_v)$
18 $\quad$ **end**
19 $\quad U_u \leftarrow \texttt{SelectNeighbors}\ (\{s_{u,v}\}_{v \in U})$
20 **end**
21 **for each** $u \in U$ **do**
22 $\quad \tau_u \leftarrow \texttt{Aggregate}\ (\{T_v\}_{v \in U_u})$
23 **end**

FIGURE 3 VSRank-algorithm (Wang et al., 2014)

First part of the algorithm (FIGURE 3) (lines 1-14) represent each user as a vector of relative preference terms based on the vector space model. For each user u relative preference terms T_u are extracted, forming a preference terms T (lines 1-5). Next block (lines 6-8) computes the specialty weight for each term $t \in T$. Lines 9-14 compute the degree weights and obtain a vector of degree-specialty weights for each user.

The latter part (lines 15-23) contains CF procedure to make recommendations for each user. Similarity between u and the rest of the users are computed in lines 16-18. The neighborhood users U_u are selected in line 19. The rest of the algorithm aggregate preferences of the neighborhood users into a total ranking of items τ_u for recommendation.

3.1.4 ListCF

ListCF stands for Listwise Collaborative Filtering and it represents a memory-based ranking-oriented CF approach, which measures the user-user similarity based on Kullback-Leibler divergence. What makes ListCF different from previous ranking-based CF algorithms is its function to directly predict a total order of items for each user based on similar user's probability distributions over permutations of commonly rated items.

ListCF predicts item rankings for each user by minimizing the cross-entropy loss between the target user and his neighboring users with weighted

similarities. The advantage in this approach, compared to algorithms that focus on predicting pairwise preferences between items, is reduced computational complexity in training and prediction procedures. Ranking performance should be at the same level or better than on previous memory-based CF algorithms. (Huang et al., 2015).

Where EigenRank uses Kendall's tau correlation and VSRank implies Vector Space model for position ranking, ListCF approach is to utilize Placket-Luce, widely used permutation probability model, to represent each user as a probability distribution over the permutations of rated items. The similarity between two users is measured based on the Kullback-Leibler divergence between their probability distributions over the set of commonly rated items. The neighborhood users for the target user are the ones that have higher similarity scores. ListCF infers predictions by minimizing the cross-entropy loss between their probability distributions over permutations of items with gradient descent.

Huang et al. (2015) have considered different variations for calculating probability of permutations in phase I. For a set of n items, the number of different permutations is n! and therefore too time-consuming to calculate the probabilities of all the permutations. This is a problem that effects performance far too heavily and forces to implement more efficient solutions. Huang et al. end up using Top-k permutation set, which focuses only on the permutations of the items within the top-k positions. The number of permutation sets to consider is n!/(n - k)! , each containing (n – k)! permutations of I (= set of items). The total probabilities of permutations are n!=(n - k)! x (n - k)! = n!, the same as in full permutations of all the items. For this problem, Huang et al. (2015) refer to the probability distribution over the top-k permutation sets as the top-k probability model, which uses the top-k permutation sets instead of the full permutations of all the items. Kullback-Leibler divergence based metric is used for similarity calculation as it is a common measure of the difference between two probability distributions in probability theory and information theory. The similarity between each pair of users is between {0, 1}.

In phase II, the goal is to predict a preference ranking of items. The assumptions are similar to traditional pairwise CF: to make predictions based on a set of neighborhood users with similar ranking preferences to the target user. The past ranking preferences affect to the upcoming ranking preferences, which is normal to memory-based recommender systems. The cross-entropy loss function, widely-used function for optimizing similarity or distance between probability distributions, is used for making predictions.

In FIGURE 4, we see ListCF algorithm in pseudocode. The lines 1 – 7 represents the phase I, where the similarities between users are calculated and neighborhood users discovered. Lines 8-23 represents the phase II where the ranking of items is predicted for making the recommendations.

It is possible to modify the k-value in top-k permutation set in ListCF. Huang et al. (2015) tested values 1,2 and 3 with 1000 randomly selected users from MovieLens-1M dataset. According to the test, the recommendation

accuracy improved a little when increasing the *k*-value. However, the efficiency suffered a tremendous fall. The calculation time in phase I and phase II of ListCF algorithm is 433 and 488 times longer, respectively, when k-value was set to 3 instead of 1. It is safe to say that improved accuracy is not worth the degenerated efficiency.

When comparing the accuracy of ListCF to traditional pointwise CF and pairwise CF (e.g. EigenRank), Huang et al. (2015, p.6) demonstrate with a following example:

> Suppose there are three users $U = \{u_1, u_2, u_3\}$ and three items $I = \{i_1, i_2, i_3\}$, where u_1 assigned ratings of {5, 3, 4} to items, u_2 assigned {5, 4, 3}, and u_3 assigned {4, 3, 5}. In pointwise CF, the similarity (Pearson correlation coefficient) between u_1 and u_2 and the similarity between u_1 and u_3 are $\rho(u_1, u_2) = \rho(u_1, u_3) = 0.5$. In pairwise CF, the similarity (Kendall's τ correlation coefficient) between u_1 and u_2 and the similarity between u_1 and u_3 are $\tau(u_1, u_2) = \tau(u_1, u_3) = 0{:}333$. In ListCF, according to Equation (1), the top-1 probability distributions of users u_1, u_2 and u_3 are $P_{u1} = (0{:}665, 0{:}090, 0{:}245)$, $P_{u2} = (0{:}665, 0{:}245, 0{:}090)$, and $P_{u3} = (0{:}245, 0{:}090, 0{:}665)$. According to Equation (2), $s(u_1, u_2) = 0{:}776$ and $s(u_1, u_3) = 0{:}395$, and thus $s(u_1, u_2) > s(u_1, u_3)$.

If we look at the ratings users have given, we can see that user u_1 and u_2 are more similar to each other than users u_1 and u_3. While the rating values are the same for each user, Pearson correlation coefficient and Kendall's τ indicate, that the users are similar to each other. As one can see from the example above, ListCF is the only one that calculates the user similarities correctly.

Input: An item set I, a user set U, and a rating matrix $R \in \mathbb{R}^{M \times N}$. A set of rated items $I_u \subseteq I$ by each user $u \in U$. The maximal number of iterations $maxIteration$ and error threshold ϵ.

Output: A ranking $\hat{\tau}_u$ of items for each user $u \in U$.

for $u \in U$ **do**
  for $v \in U$ *and* $u \neq v$ **do**
    $P_u, P_v \leftarrow \text{TopKProDist}(I_u, I_v, R)$ `/* Eq.1 */`
    $sim(u, v) \leftarrow \text{Similarity}(P_u, P_v)$ `/* Eq.2 */`
  end
  $N_u \leftarrow \text{SelectNeighbors}(\{sim(u, v)\}_{v \in U/u})$
end
for $u \in U$ **do**
  $t = 1$
  repeat
    $\varepsilon = 0$
    $\text{Initialize}(\varphi_u^0)$
    for $g \in \mathcal{G}_k^{T_u}$ **do**
      $\varphi_{u,g}^t \leftarrow \text{Update}(N_u, sim, R)$ `/* Eq.8 */`
      $\varepsilon += \sqrt{\sum (\varphi_{u,g}^t - \varphi_{u,g}^{t-1})^2}$
    end
    $t \leftarrow t + 1$
  until $t > maxIteration$ *or* $\varepsilon < \epsilon$;
  for $t \in T_u$ **do**
    $P(t) \leftarrow \text{Aggregation}(\{\varphi_{u,g}\}_{g \in \mathcal{G}_k^{T_u}})$
  end
  $\hat{\tau}_u \leftarrow \text{Ordering}(\{P(t)\}_{t \in T_u})$
end

FIGURE 4 ListCF-algorithm (Huang et al., 2015)

4 METHODOLOGY

Since the approach to the subject of recommender systems in this research is rather technical, I have chosen design science research methodology as a framework. Design science is fundamentally a problem-solving process (Hevner et al, 2004). Improving both the effectiveness and prediction accuracy in recommender systems is without a doubt a problem-solving process as well. However, the subject in this research is not to construct an artifact but to compare artifacts to each other.

It was until early 1990s, when DS was considered as a research method for development in Information Systems (IS). One might note that IS research itself is only about one-third of a century old (Peffers, Tuunanen, Rothenberger & Chatterjee, 2008.). DS is relatively young research method for IS but it has got a solid ground in researches where a design artifact is developed for extending the boundaries of human and organizational capabilities. "Design science, as the other side of the IS research cycle, creates and evaluates IT artifacts intended to solve identified organizational problems." (Hevner et al., 2004.). In IS research, technology and human behavior are inseparable and hereafter scientific research should be assessed considering its practical implications (Hevner et al., 2004).

Hevner et al. (2004) present guidelines for design-science research (table 6). First guideline is to design a purposeful IT artifact to address an important organizational problem. Instead of a whole information system, the artifact can be one crucial part of it. For example, IT artifact can be a software tool for improving the process of information system development.

The aim is to do an experiment about the artifacts that are already created by others. Hence, we are implementing parts of the DSRM approach as a guideline to fit this experiment under IS research. To be more precise, I am using guidelines from three to seven of the Design-science research guideline (table 6).

TABLE 6 Design-science research guidelines (adapted from Hevner et al., 2004)

Guideline	Description
Guideline 1: Design as an Artifact	Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation.
Guideline 2: Problem Relevance	The objective of design-science research is to develop technology-based solutions to important and relevant business problems.
Guideline 3: Design Evaluation	The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.
Guideline 4: Research Contributions	Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies.
Guideline 5: Research Rigor	Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.
Guideline 6: Design as a Search Process	The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.
Guideline 7: Communication of Research	Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.

Guideline 3 describes the utility, quality and efficacy of a design artifact. These attributes of algorithms used in this paper, except PointCF which is used as a benchmark, are explained in needed detail on chapter 3. Hevner et al. (2004) describes five different evaluation methods for design science in IS: observational, analytical, experimental, testing and descriptive. Of these five methods, experimental is suitable for evaluating recommender systems algorithms since it is a test with fixed data and platform.

One example of an artifact in IS-research and DSRM is an algorithm. To be more precise, algorithm can be defined as a method (Hevner et al., 2004). Since we are comparing four different algorithms designed to provide a best solution for predicting recommended items to target user, we are comparing four different methods of achieving the best viable outcome.

Guideline 4 is about the contributions the artifact should provide to research community. The goal in ranking-based algorithms is to improve the predicted item list ranking from the results of a one generated by rating-based algorithm. When the focus is on a list of predicted items, say top-10 list of movies, the effect is more usable to real-world scenarios, compared to predicting the best rating of an item for the target user. Not only should ranking-based CF provide more accurate predictions for user, but it also brings up a new way of approaching the recommendation problem itself. For most cases, it is more important for user what item user likes most, rather than what rating might be.

The definition of algorithms in chapter 3 along with the research papers, where the algorithms were published, provide a rigor approach that is required in guideline 5. All the algorithms are designed to improve the previous

approaches in CF-context, which has required a search process for problem survey.

Design as a search process in guideline 6 fits this scenario well, since the goal for all algorithms is to be better than previous CF approaches. By comparing the algorithms to each other with same data, we are searching for the best solution to a problem about item recommendation.

The benchmark consists of two phases: training phase and test phase. In training phase, the selected algorithms are trained for prediction using data-set-specific training data. The algorithms calculate the similarities between users to be able to predict the ranked lists. The training phase produces a table of similarities between users which is then used in the test phase. The predictions are calculated in test-phase. Selected algorithm predicts the preference ordering of unrated items for each target user using data-set-specific test data. Lastly, the prediction accuracy is evaluated using specific evaluation methods.

The methodology chapter is divided into subchapters as follows: chapter 4.1 introduces selected datasets, chapter 4.2 explains the tool for the experiment, chapter 4.3 introduces the selected evaluation methods.

4.1 Data collection and use

Selected real-world data sets are MovieLens (Harper & Konstan, 2016), EachMovie and Netflix. The datasets used in this work are given to me as part of the assignment. All three datasets are popular among research articles about recommender systems. For keywords EachMovie, MovieLens and Netflix-data, Google Scholar finds 16800, 11600 and 1260 results, respectively. These datasets have also been used in algorithm papers that are in use in this research. EachMovie and MovieLens datasets have been used in EigenRank (Liu & Yang, 2008) and VSRank (Wang et al., 2014). ListCF (Huang et al., 2015) used EachMovie, MovieLens and Netflix data to test the performance of the algorithm.

The data format in test sets are similar, each containing two files: training set and test set. The data files consist of user Id, item Id, rating value between 1-5 and timestamp. Timestamp is not needed and therefore it is not loaded into the program. Files are simple text files in txt-format and values are tab-separated from each other. More details in table 7 below:

TABLE 7 Details about datasets

	MovieLens – 1M	**EachMovie**	**Netflix**
users	6 040	36 656	429 584
items	3 952	1 623	17 770
ratings	1 000 209	2 580 222	99 884 940
ratings/user	165.6	70.4	232.5
ratings/item	253.1	1 589.8	5 621
sparsity	95.8%	95.7%	98.7%

The selected three datasets provide a good data variation since the number of users, items and ratings differs a lot. MovieLens data represents the smallest dataset with approximately one million ratings. However, the ratings per user ratio is significantly higher than in EachMovie-dataset with approximately 2,6 million ratings. EachMovie do have higher ratings per item ratio, though. Netflix-dataset is the largest in every category with approximately 100 million ratings. The data sparsity is calculated as follows:

$$sparsity = 1 - \frac{number\ of\ ratings}{number\ of\ users\ \times\ number\ of\ items}$$

4.1.1 EachMovie

EachMovie is user-movie ratings data set provided by the Compaq Systems Research Center (Melville, Mooney & Nagarajan, 2002). EachMovie was shut down in 2004 and dataset became available to public after that. The original dataset contains 2,811,983 ratings entered by 72,916 for 1628 different movies (http://grouplens.org/datasets/eachmovie). EachMovie dataset is hugely popular despite it is not anymore available. Google Scholar returns 15800 references to search term "eachmovie".

Originally MovieLens dataset was based on EachMovie dataset. At the time EachMovie was being shut down, it belonged to Digital Equipment Corporation (DEC). DEC contacted the recommender systems community for finding an organization to continue maintaining and developing the data set. GroupLens volunteered for the task. However, legal issues blocked directly transferring user accounts and DEC transferred an anonymized dataset to GroupLens. GroupLens used this dataset to train the first version of the MovieLens (Harper & Konstan, 2016.).

4.1.2 MovieLens

The MovieLens datasets was first released in 1997. Like Netflix and EachMovie, MovieLens describes people's expressed preferences for movies. The data take the form of *user, item, rating, timestamp.* Each line represents a person expressing a preference for a movie at a particular time. Only users with at least 20 ratings are included. (Harper & Konstan, 2016).

The data is collected from a web service movielens.org - a recommender system that asks its users to give movie ratings in order to receive personalized movie recommendations (Harper & Konstan, 2016). Basically, the web site works like Netflix, asking ratings for movies. However, MovieLens is non-commercial and the ratings are only used for scientific purposes.

MovieLens dataset is one of the most popular datasets in recommendation research. Google Scholar returns over 10300 references to "movielens".

MovieLens can be considered a successor for EachMovie. MovieLens is also easy to attain – there are four different-size datasets available, 100k, 1m, 10m and 20m, reflecting the approximate number of ratings per dataset.

4.1.3 Netflix

The largest dataset in this research is Netflix data with 100 million movie ratings. The dataset is released in October, 2006 as a part of The Netflix Prize competition. The idea of the competition was to challenge data mining, machine learning and computer science communities to develop systems that could beat the accuracy of, at that time, current system. (Bennett & Lanning, 2007).

The main prize for that competition was 1 million dollars for the winning team. Netflix's system to beat was called Cinematch. Cinematch used a variant of Pearson's correlation (see chapter 3.1.1.) to calculate item similarities (Bennett & Lanning, 2007). The goal was to produce a 10 percent reduction in the RMSE of test data compared to Cinematch score. Not one team achieved this goal. The best result was 8.43% from team KorBell of AT&T Labs-Research. (Bell & Koren, 2007).

4.2 Tool for the experiment

The tool that is used in this research measures the performance (resolution time) and accuracy of three different CF algorithms. The algorithms are: EigenRank, VSRank and ListCF. To give a baseline for the results, there is also a 'traditional' pointwise CF-algorithm, PointCF.

I programmed the benchmark tool using Java-language. As this software's purpose is to be a tool for algorithm comparison, the user interface is command-line based, which requests three different parameters: algorithm, dataset, evaluation method.

4.3 Results measurement

Evaluating prediction accuracy of recommender system algorithms is often executed using metrics like Mean Absolute Error (MAE), Root Mean Square Error (RMSE), Normative Discounted Cumulative Gain (NDCG) and Mean Average Precision (MAP). MAE and RMSE are used when evaluating rating-oriented algorithms where NDCG and MAP are used when dealing with ranking-based algorithms. This chapter briefly explains the function of these evaluations metrics to help understand the results of different algorithms. Chapter 4.3.1 explains NDCG and chapter 4.3.2 introduces MAP.

4.3.1 Normalized Discounted Cumulative Gain (NDCG)

Normalized Discounted Cumulative Gain measures the performance of a recommender system based on the graded relevance of the recommended items. Like in previous measurement methods, the result value is between [-1, 1], value 1 representing the perfect prediction.

The NDCG metric is evaluated over number k which represents the top items on the ranked list. The variable Q is the set of users used for testing. $R(u,p)$ is the rating that is assigned by u to the item at the p-th position on the ranked list produced for user u. The NDCG at the k-th position to the set of Q is shown in figure 13. Variable Z_u is a normalization factor which is calculated so that the NDCG of the optimal ranking has a value of 1. Value $log(1 + p)$ is a discounting factor, which increases with the position in the ranking. Discount is by position, so things at front are more important. These features make it desirable for measuring ranking quality in recommender systems. This is due to a fact that most users rarely look past the first few items on a recommendation list. The relevance of the top items in the list are more important than those at low positions. (Liu & Yang, 2008). Below is formula of NDCG.

$$NDCG(Q,k) = \frac{1}{|Q|} \sum_{u \in Q} Z_u \sum_{p=1}^{k} \frac{2^{R(u,p)} - 1}{\log(1+p)}$$

4.3.2 Mean Average Precision (MAP)

MAP is a popular performance measure for calculating the mean of average precisions scores for each query. MAP is the most commonly used single-value summary of a run over a set of queries (Agichtein, Brill & Dumalls, 2006).

Average Precision (AP) can be explained as follows: There are correct rankings {1,2,3,4,5} and predicted rankings {4,2,6,1,7}, then AveP(5), where number 5 is the threshold, is 3/5 = 0,6. That is because we found correct values 4,2 and 1 and incorrect values 6 and 7. AveP(3) returns 1 correct value, 2, and two incorrect values, 4 and 6, thus 1/3 = 0,33. MAP is simply an average of the sum of these queries.

The formula for MAP is:

$$MAP = \frac{\sum_{q=1}^{Q} AveP(q)}{Q}$$

where AveP is average precision, q is current query and Q is the number of queries. It is good to acknowledge that MAP is macro-averaging measure, meaning each query is counted equally. MAP also assumes user is interested in finding many relevant documents for each query.

5 RESULTS

The results chapter is divided into four subchapters: Similarity and neighbor search comparison, runtime comparison and ranking accuracy comparison following with conclusions. In first two chapters the results are evaluated with processing speed. The faster the time, the better the performance. In ranking accuracy comparison, Normative Discounted Cumulative Gain and Mean Average Precision are used to compare the accuracy of the rankings. These evaluation methods are explained in chapters 4.3.1. and 4.3.2., respectively.

The tests were run on a computing machine provided by University of Jyväskylä. The benchmarks were run once per algorithm. Few algorithms were benchmarked twice to test that the results remained the same. These test benchmarks are not included in the results. The datasets are MovieLens, EachMovie and Netflix-data. MovieLens and EachMovie datasets were used as is but Netflix-dataset was too large to process in its original size. A sample set was created from Netflix-dataset by selecting only the users that had rated 50 or more items. The sparsity of the sample set is 99.8%.

5.1 Algorithm Training and Similarity Calculation runtime

Due to the size differences on data sets, results are shown either in seconds or minutes, depending on a data set. MovieLens data, being the smallest in size, is measured in seconds where EachMovie and Netflix are measured in minutes.

In MovieLens-dataset (FIGURE 5) the fastest performing algorithm is PointCF with 180.49 seconds. The second fastest is ListCF with 237,86 seconds. Clearly the slowest algorithms were EigenRank and VSRank with times 1729.20 seconds and 1771.92 seconds, respectively. Wang et al. (2015) measured PointCF, EigenRank and ListCF in their paper. Compared to their results, the ranking of these three algorithms is the same, although times are not similar. Compared to results made by Wang et a. (2015), PointCF and ListCF performed faster in our

test but EigenRank is multiple times slower. That excludes the influence of better computational power as all the results should have been faster in our test.

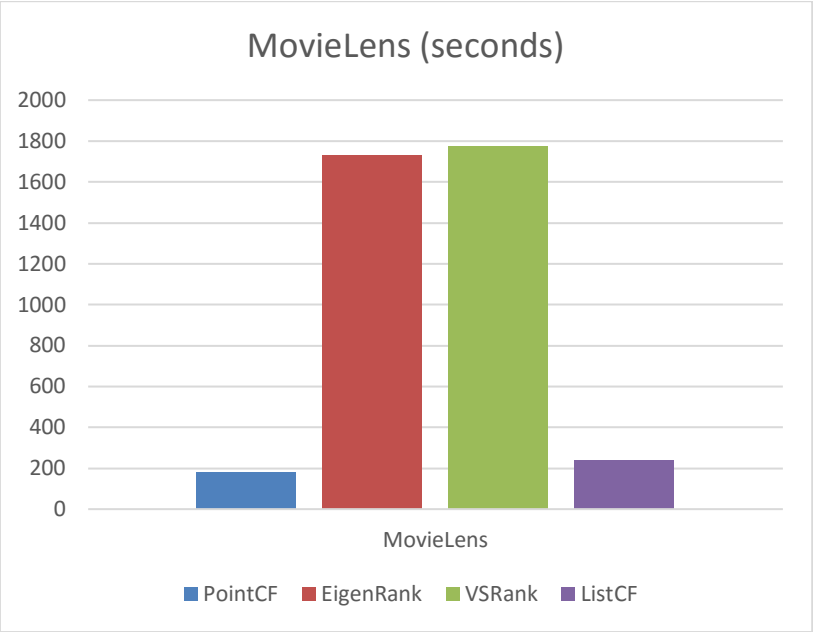

FIGURE 5 Movielens similarity and neighbor search

EachMovie-dataset provided contrary results to MovieLens as PointCF proved to be the slowest of the algorithms and VSRank the fastest (FIGURE 6FIGURE 7). Execution time of VSRank is 191.52 minutes, EigenRank 664.25 minutes, ListCF 918.25 minutes and PointCF 1096.85 minutes.

One possible explanation to this is in dataset details. Ratings per user is higher in MovieLens but ratings per item in EachMovie is six-fold compared to MovieLens. In this type of data, VSRank proved itself to be fastest performing algorithm.

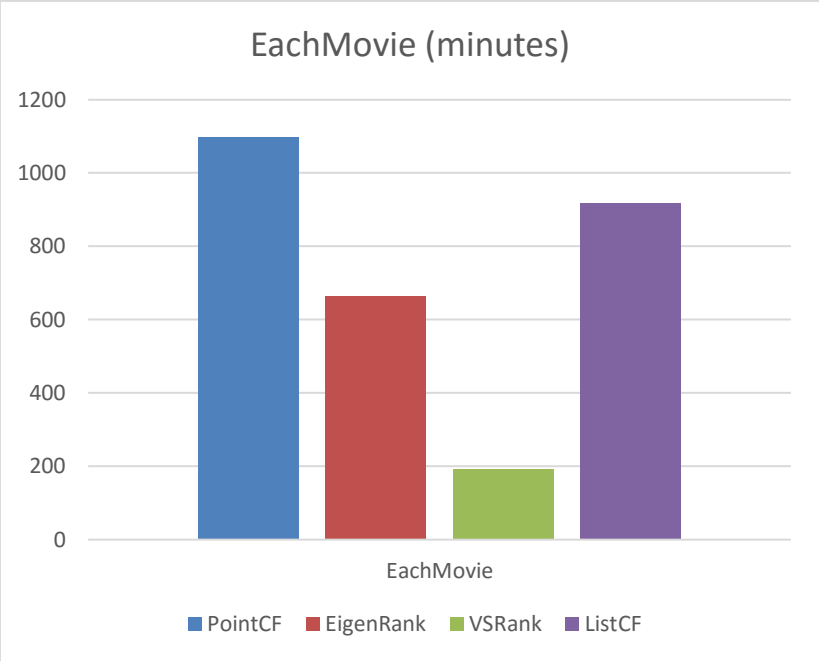

FIGURE 6 EachMovie similarity and neighbor search

The ranking of the algorithms stays the same while testing them with Netflix-data (FIGURE 7). Computation times for PointCF, EigenRank, VSRank and ListCF are 463.52, 307.85, 20.35 and 334.22 minutes, respectively. With Netflix-data, VSRank is the fastest algorithm, being 15 times faster than EigenRank which resulted second.

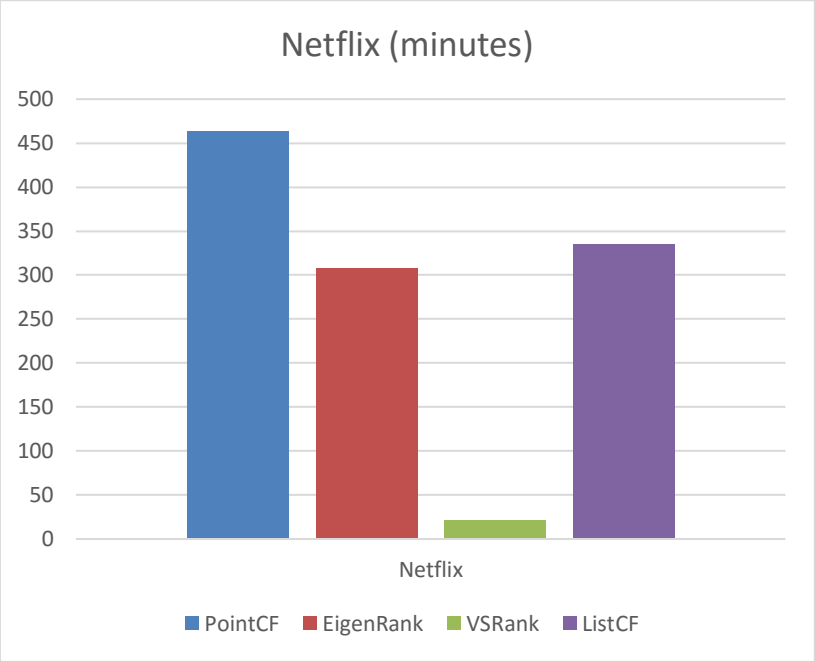

FIGURE 7 Netflix similarity and neighbor search

It is no surprise that the results of EachMovie and Netflix are close to each other. The user counts are 36656 and 32421, respectively. EachMovie has 2580222 ratings where Netflix-sample data has 906758, which is 2.8 times less. The factor in time difference between EachMovie and Netflix-benchmarks are approximately the same.

The vast difference between VSRank and other algorithms is a signal that VSRank is remarkably efficient on handling data where ratings per user is low and ratings per item is high. This notes that vector space approach with TF-IDF weighting scheme is good performance-wise in CF applications. On a sample set of Netflix-data there is only 28 ratings per user whereas EachMovie has 70.4. Both ratios are significantly lower compared to Movielens' 165.6 ratings per user. VSRank also has an advantage in runtime when datasets are bigger in size, according to these three datasets.

5.2 Prediction Runtime

In prediction phase, rating-based PointCF is the fastest with a huge margin to ranking-based solutions on every data set (FIGURE 8). The results times with MovieLens data for PointCF, EigenRank, VSRank and ListCF are 2816, 7429, 17570 and 7915 milliseconds, respectively. With MovieLens data, VSRank is significantly slower than the others. Runtime speed is virtually the same between EigenRank and ListCF, latter being 6% slower.

The results with EachMovie data differs from MovieLens. PointCF is the fastest with the runtime of 5212 milliseconds. EigenRank and VSRank share almost the same runtime of 32970 and 33924 milliseconds, VSRank being 3% slower. ListCF is clearly the slowest with the time of 56 828 milliseconds, being 1090% slower than PointCF.

With Netflix-data, the ranking of last three algorithms has changed compared to EachMovie which is almost the same in size. The runtimes for PointCF, EigenRank, VSRank and ListCF are 4223, 26163, 21141 and milliseconds, respectively. VSRank finished second with Netflix-data with a clear margin before EigenRank and ListCF. The latter two however predicted the ranking in almost same time, with only 57 millisecond difference in favor for ListCF.

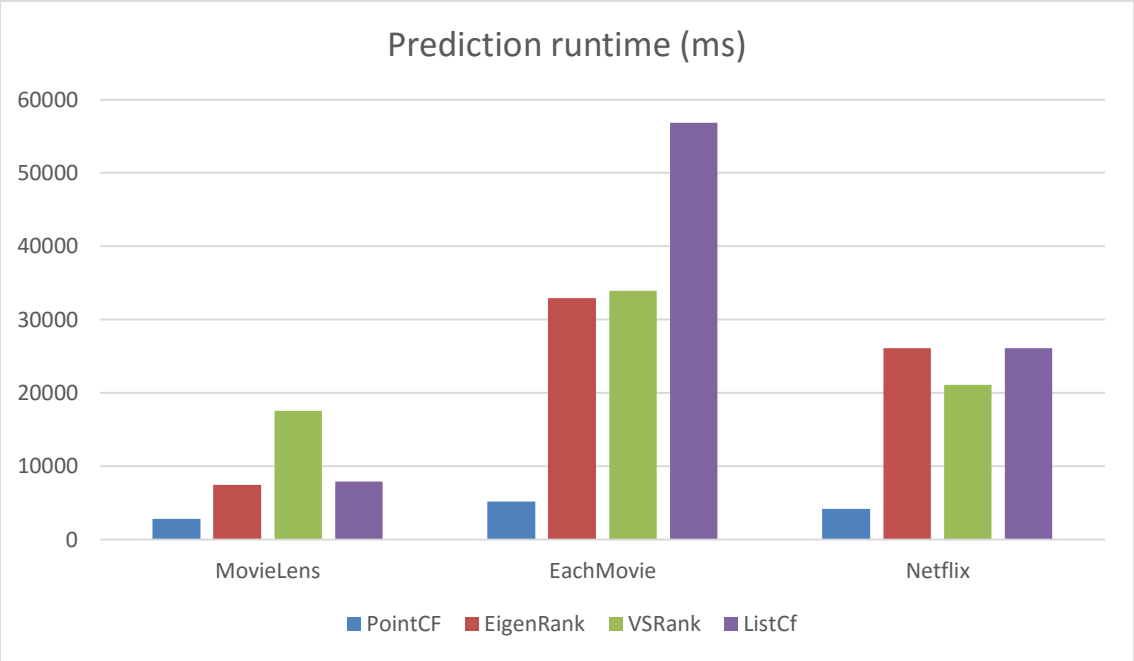

FIGURE 8 Prediction runtime

It is good to acknowledge that ranking-based algorithms are not designed for performing faster but to provide more accurate predictions. Simply put, the functionalities of these are much more complex.

The results for prediction runtime shows that the details of a dataset affect a lot for the results, which. PointCF manages to predict the fastest, no matter the dataset. However, the ranking based algorithms are more sensitive about the dataset. The approaches for the prediction differs a lot between ranking-based algorithms. If the performance time is an important factor when selecting the algorithm, fastest option is PointCF. If the details of a target data-set are similar and previously known, one could select the best performing ranking-based algorithm based on that.

5.3 Prediction accuracy comparison

The results for accuracy are measured with NDCG and MAP and the results are listed below in tables 8 and 9 and with graphs for better readability (Figures 10-15). The number after the evaluation method shortening means the position of the prediction. For example, NDCG@3 tells how accurately algorithm has predicted the 3rd ranking for the set of users in data-set. The value is the mean value over the set of users. The scale is [-1, 1], number 1 describing the perfect ranking. The best results between algorithms are bolded in tables 8 and 9.

TABLE 8 Ranking performance measured in NDCG (bolded values present the best result)

Dataset	Metric	PointCF	EigenRank	VSRank	ListCF

MovieLens	NDCG@1	0,6749	0,6238	**0,7266**	0,7196
	NDCG@3	0,7103	0,6504	**0,7477**	0,7369
	NDCG@5	0,7530	0,6946	**0,7838**	0,7639
	NDCG@10	0,8777	0,8517	**0,8932**	0,8872
EachMovie	NDCG@1	0,5762	0,6079	0,7280	**0,7450**
	NDCG@3	0,6253	0,6391	0,7504	**0,7653**
	NDCG@5	0,6808	0,6874	0,7898	**0,8034**
	NDCG@10	0,8289	0,8350	0,8876	**0,8950**
Netflix	NDCG@1	0,6464	0,6494	0,6726	**0,7028**
	NDCG@3	0,6744	0,6677	0,6848	**0,7114**
	NDCG@5	0,7128	0,7040	0,7172	**0,7421**
	NDCG@10	0,8591	0,8561	0,8633	**0,8758**

By evaluating the predictions using NDCG, VSRank provided the most accurate predictions in MovieLens-dataset. However, the results are not far ahead from the accuracy of ListCF. Rating-based PointCF managed to predict with better accuracy than ranking-based EigenRank. It is good to mention that runtimes of PointCF were also tremendously faster both in similarity and neighbor search (FIGURE 5) and prediction runtime (FIGURE 8).

With EachMovie-dataset, the best results were provided by ListCF. Second is VSRank following with EigenRank and PointCF. Again, ListCF and VSRank are close to each other and have a big margin for EigenRank and PointCF, which also are close to each other. This is clearly visible on FIGURE 10. Although ListCF provided slightly better accuracy, the runtimes compared to VSRank are much longer.

The order of algorithms is the same with Netflix-data than with EachMovie-data. However, the difference between results are bigger between ListCF and VSRank. ListCF is the only algorithm that exceeds 0.7 accuracy in NDCG@1 and NDCG@3. EigenRank is better than PointCF in NDCG@1 but worse in other measurement points.

TABLE 9 Ranking performance measured in MAP (bolded values present the best result)

Dataset	**Metric**	**PointCF**	**EigenRank**	**VSRank**	**ListCF**
MovieLens	MAP@1	0,7800	0,7200	**0,8298**	0,8195
	MAP@3	0,7584	0,6911	**0,7913**	0,7770
	MAP@5	0,7306	0,6747	**0,7521**	0,7257
EachMovie	MAP@1	0,8146	0,8246	0,9156	**0,9174**
	MAP@3	0,7983	0,7986	0,8812	**0,8818**
	MAP@5	0,7799	0,7749	0,8457	**0,8466**
Netflix	MAP@1	0,7052	0,7111	0,7347	**0,7668**
	MAP@3	0,6825	0,6741	0,6910	**0,7164**
	MAP@5	0,6588	0,6589	0,6577	**0,6770**

The second evaluation method is MAP (Table 9). The selected measurement points are MAP@1, MAP@3 and MAP@5. For technical reasons MAP@10 is not selected as the results were identical between algorithms. On the contrary to NDCG, the results in MAP are higher in top positions and decreasing when the predicted position increases.

Like with NDCG, VSRank is the most accurate with MovieLens-data. ListCF is second, PointCF third and EigenRank last. Interestingly, PointCF outperforms ListCF in MAP@5. ListCF provided most accurate predictions with EachMovie-data. The results of VSRank are less accurate by 0.2% or less compared to ListCF in MAP@1,3,5. This minimal improvement comes to a cost of much longer prediction runtime, which is clearly visible in FIGURE 8. FIGURE 13 shows how the results are split into two groups. First group includes PointCF and EigenRank, second VSRank and ListCF. Netflix-data is the one where ListCF stands out from other algorithms. The results are 2.85%-4.18% better than the second-best algorithm, VSRank.

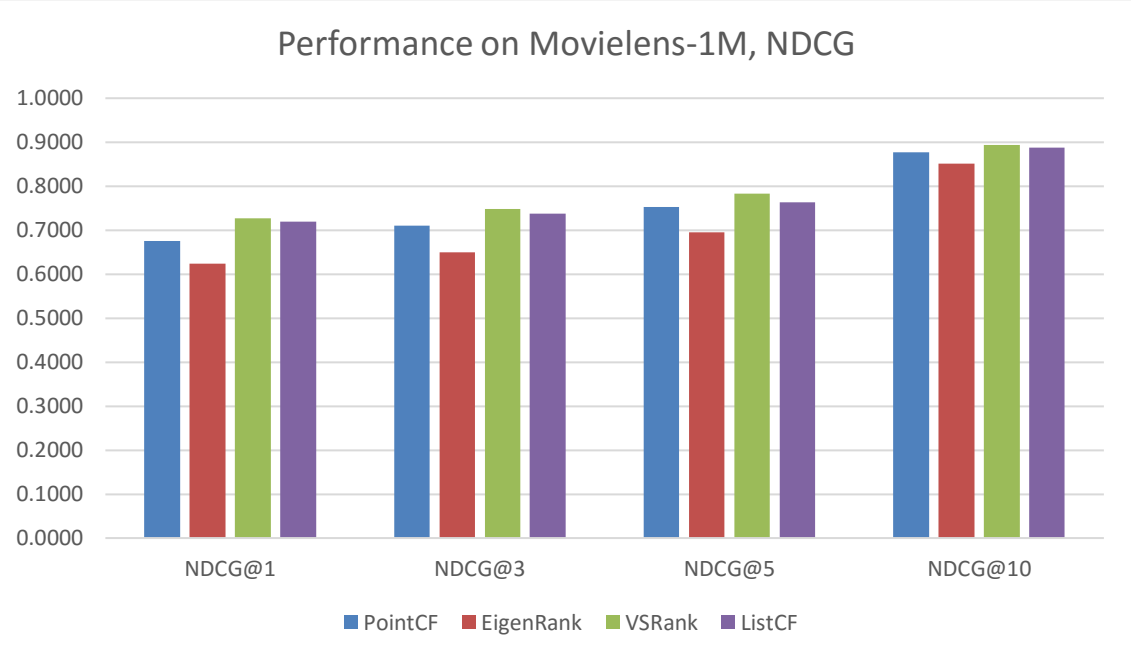

FIGURE 9 Performance on MovieLens data, measured in NDCG

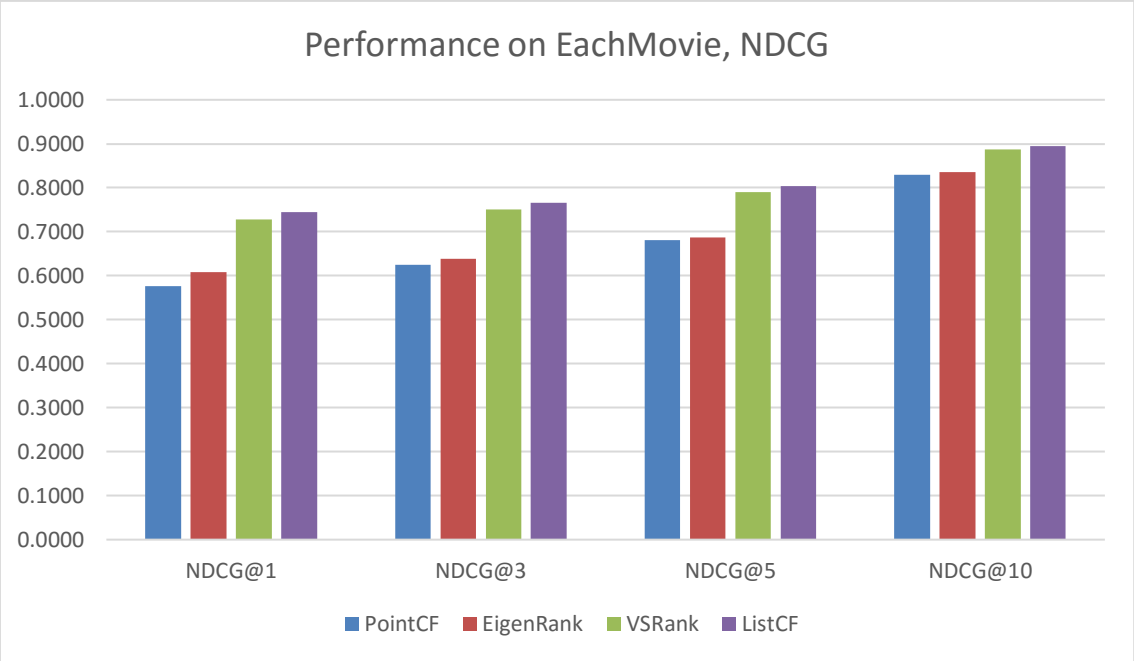

FIGURE 10 Performance on EachMovie data, measured in NDCG

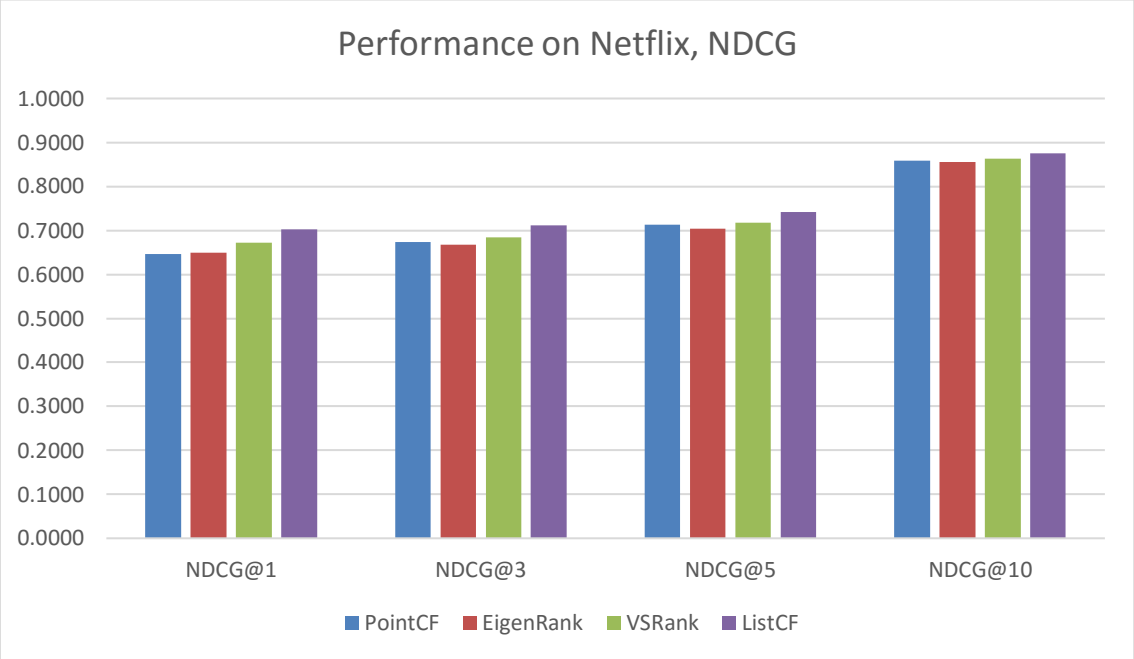

FIGURE 11 Performance on Netflix data, measured in NDCG

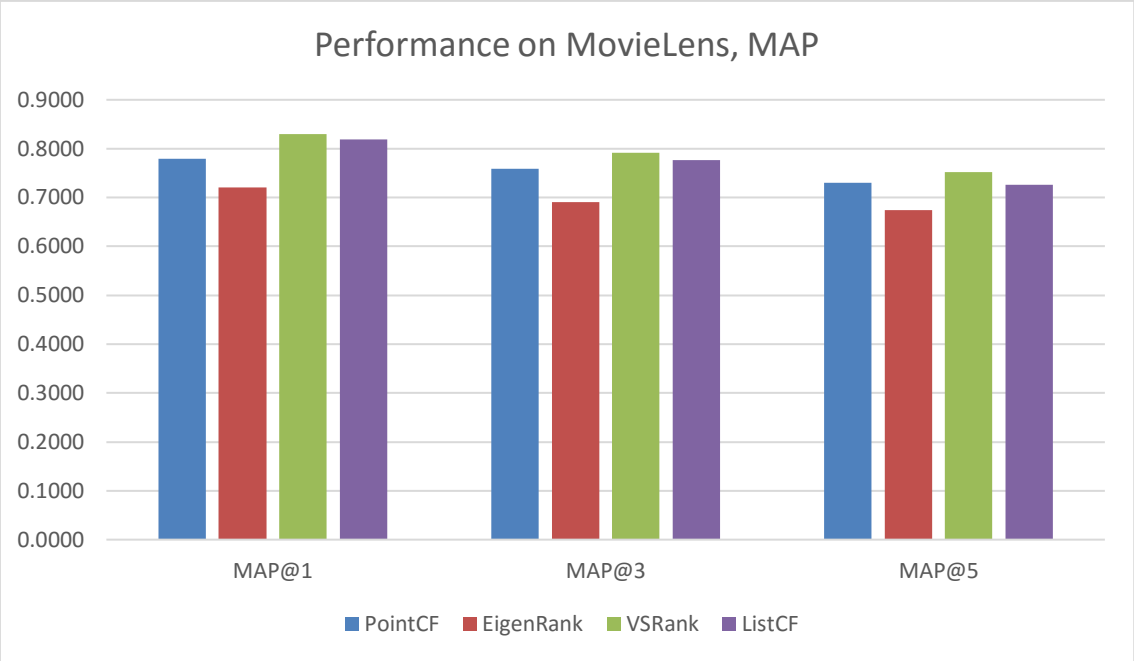

FIGURE 12 Performance on EachMovie data, measured in MAP

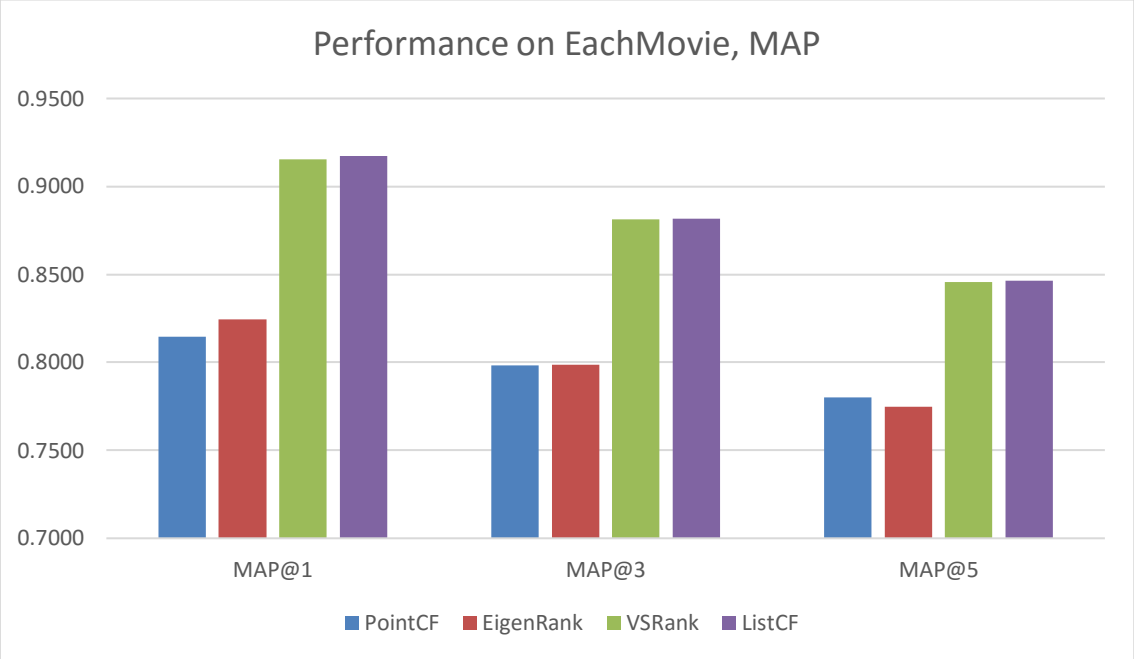

FIGURE 13 Performance on EachMovie data, measured in MAP

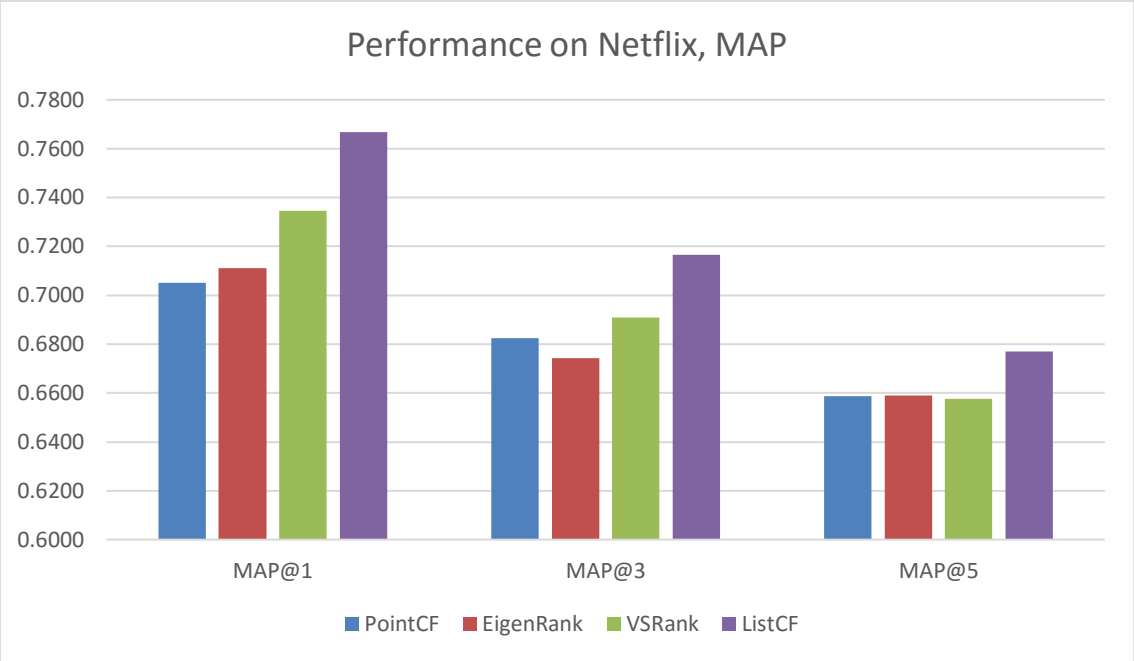

FIGURE 14 Performance on Netflix data, measured in MAP

5.4 Summary of the results

EigenRank is one of the first ranking-based algorithms and it is the oldest algorithm in this study. It was designed to outperform traditional rating-based CF. When reading the results of the benchmarks run in this research, one can notice that EigenRank competes with PointCF. However, both EigenRank and PointCF are systematically left behind of VSRank and ListCF in a comparison of prediction accuracy.

VSRank and ListCF represents the newer adaptations of ranking-based CF. The accuracy results indicate that ranking-based algorithms have been improved significantly in a last decade. When it comes to prediction accuracy, rating-based CF performs worse than ranking-based CF. However, the prediction runtime of rating-based CF is generally much better than in ranking-based CF approaches. The similarity calculation and neighbor search-phase in PointCF is faster with MovieLens compared to other algorithms, but slower with EachMovie and Netflix.

If prediction runtime is not an issue and the number one priority is prediction accuracy, one should select modern ranking-based CF approaches like VSRank or ListCF to their applications. That is of course when the usage of recommender system is to provide ranked list.

6 DISCUSSION

From a wide range of recommender systems, this study focused on one of the first and still popular implementations, the Collaborative Filtering. CF algorithms have improved both in performance (runtime) and accuracy over the years. As explained in chapter 2, CF implementations can be divided into many sub-categories. This study divided CF into memory-based and model-based filtering and focuses on the memory-based CF. Memory-based CF is again divided into rating-based and ranking-based approaches.

Rating-based CF is the older of these two approaches and ranking-based was developed to challenge the recommendation performance. The ranking-based CF is designed to provide a Top-N list of recommended items, instead of predicting the ratings for items first and ordering them to descending list according to predicted rating values second.

The performance of the selected three algorithms in addition to rating-based PointCF, which worked as a baseline algorithm, were tested using three real-world datasets: EachMovie, MovieLens and Netflix. These datasets vary in details from each other, as one can see in table 7. This provided a good base for comparison as one algorithm might be better in smaller data and another in larger data. The rating per user-ratio, rating per item-ratio and data sparsity also affected in performance.

The performance was evaluated by comparing two attributes: runtime and accuracy. Runtime comparison is simple, the faster the better. For accuracy, I used two evaluation methods: Mean Average Precision and Normative Discounted Cumulative Gain. With these three evaluations, we received rigor results.

This study was executed as DSRM since the focus was on artifacts which purpose is to solve a problem with prediction accuracy. This study did not create a new artifact. Instead, we were comparing already made artifacts and their performances. Hence, only selected parts of DSRM were applied to this research.

The literature used in this study was conducted from the most common publications related to this subject. The main sources were given in assignment

and most of the material was acquired using Google Scholar. Where it was necessary, as new as possible sources were used. Since the subject was quite limited, there were difficulties in finding academic sources to some of the chapters. For example, chapter 3 about selected algorithms is almost entirely based on the publication papers about the algorithms itself.

6.1 Key findings

The research questions are as follows:

- Why ranking-based algorithms should provide better results than traditional rating-oriented algorithms?
- Do the proposed algorithms perform better than rating-based algorithm in real-world benchmark datasets?

The fundamental difference between rating-based CF and ranking-based CF is that ranking-based CF does not provide a predicted rating to items at all. Traditional CF algorithms are based on predicting the potential ratings that a user would assign to the unrated items so that they can be ranked by the predicted ratings to produce a list of recommended items. Ranking-based CF addresses the item ranking problem directly by calculating user preferences derived from the ratings users have given before (Liu & Yang, 2008). Ranking-based CF approaches differ from each other by using different methods on similarity calculation between users. The selected three ranking-based CF algorithms, EigenRank, VSRank and ListCF are presented in more detail in chapter 3.

The results of the benchmarks executed in this thesis are analyzed in chapter **Error! Reference source not found.**. The outcome of the benchmarks is that ranking-based CF algorithms provide better accuracy compared to rating-based CF, named PointCF in this study. In some cases, ranking-based CF provided also faster processing times, although the implementations of ranking-based CF tend to be more complex than in rating-based CF variants.

To answer to the first research question, I did investigation of ranking-based CF in comparison with rating-based CF and discovered why ranking-based CF should provide better results than rating-based CF. The summary of the benchmark results discussed in chapter 5.4 prove that ranking-based CF provides better accuracy than rating-based CF, giving answer to the second research question.

6.2 Contribution

The contribution for this study is to show that, when it comes to predicting ranked lists, ranking-based CF outperforms rating-based CF. The outcome is a result of comparing four algorithms to each other, which has only been done in one research paper before this thesis. This study also provided how much ranking-based CF have improved from 2008, when EigenRank was introduced.

Ranking-based CF is one limited subject in recommender systems research and there are relatively small amount of research papers regarding this subject. Most of the papers are publications of a new ranking-based CF approach and the performance is measured by comparing it to rating-based CF and in some cases to an older ranking-based CF, e.g. EigenRank. EigenRank is the most popular ranking-based CF algorithm in this thesis where VSRank and ListCF are less known, partly because they are relatively new. The former is introduced in 2014, the latter in 2015 (Wang et al.,2014, Huang et al., 2015).

I could find only one research paper where these four algorithms were compared to each other. The paper is the publication of ListCF, titled Listwise Collaborative Filtering (Huang et al., 2015). The difference between my results and the ones in paper by Huang et al. (2015) is in slightly different evaluation methods. In addition to NDCG, I evaluated the results also with MAP. Huang et al. (2015) provided more detailed results in some sections and also added two more algorithms, CoFiRank and ListRank-MF, to the comparison. What is interesting though is that I got different results than Huang et al. (2015) in some sections. Some differences can be explained by different platforms and their computational performance. However, the algorithms should be the same as are the datasets, apart from the sample-set of Netflix data used in this thesis.

6.3 Limitations and evaluation of the research

As briefly explained in chapter 2, there are many different types of recommender systems. The subject of this research was limited to cover only a small fraction of all the recommender system types available. Also, there are many different algorithms that are based on CF and we evaluated only four of these. The number of ranking-based CF algorithms is significantly lower than in CF in general, but there still is more than the ones covered in this thesis.

The evaluation methods used for algorithm performance are commonly used in similar scientific papers. Similar kind of evaluation had been used in scientific papers that published the algorithms used in this study.

6.4 Concluding summary

The ranking problem is common among recommender systems and it has been approached by multiple different ways. Instead of seeing a rating one might give to an item, end-user is more likely interested on what items he or she would like the most. Therefore, predicting top-N ranked lists the most accurate way is both challenging and important business-wise. Ranking-based CF algorithms are designed for predicting ranked lists to target user. Since users are interested on the items that are first on the ranked list, focusing on rankings instead of ratings is arguable.

This thesis provides thorough investigation on ranking-based CF and how it differs from traditional rating-based CF. To give a more detailed approach, three ranking-based CF algorithms were selected for performance benchmarks against rating-based CF.

REFERENCES

Agichtein, E., Brill, E. & Dumais, S. (2006). Improving web search ranking by incorporating user behavior information. Proceedings of the 29th Annual International ACM SIGIR Conference on Research andDevelopment in Information Retrieval, (1926).ACM.

Bennett, J. & Lanning, S. (2007). The netflix prize. Proceedings of KDD Cup and Workshop, (35).

Bell, R. M. & Koren, Y. (2007). Lessons from the netflix prize challenge. ACM SIGKDD Explorations Newsletter,9(2), 7579.

Breese, J. S., Heckerman, D. & Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, (43-52). Morgan Kaufmann Publishers Inc.

Brin, S. & Page, L. (2012). Reprint of: The anatomy of a largescale hypertextual web search engine. Computer Networks, 56(18), 38253833.

Burke, R. (2007). Hybrid web recommender systems. The adaptive web (s. 377-408) Springer.

Goldberg, D., Nichols, D., Oki, B. M. & Terry, D. (1992). Using collaborative filtering to weave an information tapestry. Communications of the ACM, 35(12), 6170.

Good, N., Schafer, J. B., Konstan, J. A., Borchers, A., Sarwar, B., Herlocker, J. & Riedl, J. (1999). Combining collaborative filtering with personal agents for better recommendations. Aaai/iaai, (439-446).

Harper, F. M. & Konstan, J. A. (2016). The movielens datasets: History and context. ACM Transactions on Interactive Intelligent Systems (TiiS), 5(4), 19.

Herlocker, J., Konstan, J. A. & Riedl, J. (2002). An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms. Information Retrieval, 5(4), 287-310.

Herlocker, J. L., Konstan, J. A. & Riedl, J. (2000). Explaining collaborative filtering recommendations. Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work, (241-250). ACM.

Herlocker, J. L., Konstan, J. A., Terveen, L. G. & Riedl, J. T. (2004). Evaluating co llaborative filtering recommender systems. ACM Transactions on Information Systems (TOIS) 22 (1), 5-53.

Hevner, R., March, S. T., Park, J. & Ram, S. (2004). Design science in information systems research. MIS Quarterly, 28(1), 75-105.

Huang, S., Wang, S., Liu, T., Ma, J., Chen, Z. & Veijalainen, J. (2015) Listwise collaborative filtering.

Jin, J. & Chen, Q. (2012). A trust-based top-K recommender system using social tagging network. Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on, (1270-1274). IEEE.

Kantor, P. B., Rokach, L., Ricci, F. & Shapira, B. (2011). Recommender systems handbook Springer.

Konstan, J. A. & Riedl, J. (2012). Recommender systems: From algorithms to user experience. User Modeling and User-Adapted Interaction, 22(1-2), 101-123.

Koren, Y. (2008). Factorization meets the neighborhood: A multifaceted collaborative filtering model. Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, (426-434). ACM.

Liu, N. N. & Yang, Q. (2008). Eigenrank: A ranking-oriented approach to collaborative filtering. Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, (83-90). ACM.

Mahmood, T. & Ricci, F. (2009). Improving recommender systems with adaptive conversational strategies. Proceedings of the 20th ACM Conference on Hypertext and Hypermedia, (73-82). ACM.

Melville, P., Mooney, R. J. & Nagarajan, R. (2002). Contentboosted collaborative filtering for improved recommendations. Aaai/iaai, (187192).

Peffers, K., Tuunanen, T., Rothenberger, M. A. & Chatterjee, S. (2007). A design science research methodology for information systems research. Journal of Management Information Systems, 24(3), 4577.

Pennock, D. M., Horvitz, E., Lawrence, S. & Giles, C. L. (2000). Collaborative filtering by personality diagnosis: A hybrid memory-and model-based approach. Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence, (473-480). Morgan Kaufmann Publishers Inc.

Sarwar, B., Karypis, G., Konstan, J. & Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. Proceedings of the 10th International Conference on World Wide Web, (285-295). ACM.

Schafer, J. B., Frankowski, D., Herlocker, J. & Sen, S. (2007). Collaborative filtering recommender systems. The adaptive web (s. 291-324) Springer.

Su, X. & Khoshgoftaar, T. M. (2009). A survey of collaborative filtering techniques. Advances in Artificial Intelligence, 2009, 4.

Wang, S., Sun, J., Gao, B. J. & Ma, J. (2014). VSRank: A novel framework for ranking-based collaborative filtering. ACM Transactions on Intelligent Systems and Technology (TIST), 5(3), 51.

Weimer, M., Karatzoglou, A., Le, Q. V. & Smola, A. (2007). Maximum margin matrix factorization for collaborative ranking. Advances in Neural Information Processing Systems, , 18.