DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2019

Comparing neighborhood and matrix factorization models in recommendation systems

Saving the user some clicks

ERIK TORSTENSSON

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Comparing neighborhood and matrix factorization models in recommendation systems

ERIK TORSTENSSON

Master in Computer Science
Date: October 20, 2019
Supervisor: Johan Gustavsson
Examiner: Örjan Ekeberg
School of Electrical Engineering and Computer Science
Host company: Radar Ecosystems AB
Swedish title: Jämförelse av grannskap- och matrisfaktoriseringsmodeller inom rekommendationssystem

Abstract

This thesis explores how different recommendation models based on machine learning can be implemented using customer activity data from an internet portal. The recommendation models have been evaluated on both error metrics and accuracy metrics. In addition, two data-sets are used, one open and widely available called MovieLens, and one proprietary from the principal of this thesis. Furthermore, the models evaluated are based on popular neighborhood methods and more advanced matrix factorization techniques. Based on both error and accuracy metrics, the matrix factorization models SVD and SVD++ outperform the neighborhood model in both data-sets. Moreover, the SVD++ algorithm performed the best of all the models but in the end, SVD was chosen for the principal. This was done because it proved to be a better overall choice when looking at all the parameters such as performance, speed and ease of use.

Sammanfattning

Denna uppsats undersöker hur olika rekommendationsmodeller baserade på maskininlärning kan implementeras med hjälp av kundaktivitetsdata från en internetportal. Rekommendationsmodellerna har utvärderats med både felmätning och hur noggrant modellen gör sina rekommendationer. Två datauppsättningar har använts i uppsatsen, en öppen och allmänt tillgänglig, kallad MovieLens och en från företaget där denna uppsats har skrivits. De utvärderade modellerna baseras på populära grannskap- och matrisfaktoriseringsmetoder. Baserat på både fel- och noggrannhetsmätningar så överträffar matrisfaktoriseringsmodellerna, SVD och SVD ++, grannskapsmodellen i båda datauppsättningar. SVD++-algoritmen presterar bäst av alla modeller, men i slutändan så rekommenderas ändå SVD till företaget. Detta gjordes eftersom SVD visade sig vara ett bättre val om fler parametrar togs i beaktande än bara prestanda, såsom hastighet och lätthet att implementera.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In today's fast-paced world the amount of data increases every day. According to a report published by Mckinsey 50% of the responding businesses say that analytics and big data have fundamentally changed business practices in their sales and marketing functions (Henke and Kaka 2019). To continue to expand, companies today will have to embrace data collection and explore methods on how to use this data to deliver value for their customers.

One way to become a data-driven company is to predict what the customer prefers given historical data using different recommendation techniques. Such techniques are at the core of many companies today such as Netflix, Amazon, and Spotify. Moreover, they are relevant for all companies that have products where the customers can make any type of choice or have a preference for something.

This thesis explores how different recommendation models can be implemented using customer activity data from an internet portal and compare their performance using standard metrics. The recommendation models evaluated are based on popular neighborhood methods and more advanced matrix factorization techniques. The work has been done at the business intelligence company Radar Ecosystems.

Radar Ecosystems has a web portal for its customers where the customers can do bench-marking, read reports and key measurements for their specific domain. They have customers in a lot of different segments like municipalities, IT providers, buyers of IT services and more. Not all data is relevant for everyone and the amount of data makes it hard for new customers to know what is relevant for them. Hence, Radar is creating a new web portal and is inter-

ested in better customer segmentation and recommendations to make the data more relevant for everyone. Users with a similar profile should get a similar experience and get recommendations based on that profile.

1.1 Objective

The academic objective is to compare and evaluate different recommendation models, based on machine learning, to see which model gives the best recommendations based on standard metrics.

On the other hand, the principal's objective is to provide their customers with a recommendation system for their internet portal that can recommend relevant articles based on what other customers with a similar profile download. Furthermore, the goal is to increase customer satisfaction by having customers find better material, something that they think is hard today.

To summarize, the objective of this report is to evaluate different recommendation algorithms while also suggesting a suitable solution for Radar Ecosystems to use on their internet portal.

1.2 Research question

The research question addressed in this thesis is:

- How does matrix factorization compare to neighborhood models using standard metrics for generating customer-specific content recommendations on an internet portal from activity data?

Apart from the main research question, the following question will also be answered:

- Which recommendation model is most suitable for Radar Ecosystems internet portal?

1.3 Delimitations

In the time of writing this thesis, the online portal for which this recommendation system is built for is not ready yet. This means that the system proposed in this thesis can not be live tested and will only serve as as a proof of concept, to be implemented on a later date.

The data used for most of the tests in this thesis is proprietary and owned by Radar Ecosystems AB. The data contains secret and sensitive information about Radar Ecosystems AB customers and providers. Therefore, the data is not available to look at. However, a description and an example of how the data looks like can be seen in section 3.2.2.

1.4 Social and ethical aspects

A risk when the material online is tailored according to the user's preferences with recommendation algorithms is that it may become an echo chamber. The user only sees the material of the same type that he or she has already seen, thus, not finding potentially interesting material outside of that scope. Some argue that this is a big problem in the world today where more and more people get their news from social media that employs recommendation algorithms to tailor all material to the user (Grimes 2017). Consequently, the users find themselves in a news bubble where all the news and political views are similar. In effect, this can lead to a more polarizing world where different opinions are not considered or perhaps not even tolerated.

When using recommendation algorithms, data about the users must be collected to be able to give good recommendations. As a result, this can be problematic from an integrity perspective as a company will know a great deal about the user and what he or she likes. GDPR, the new online privacy law from the European Union, has made the regulations in this area stricter (*Key Changes with the General Data Protection Regulation – EUGDPR* 2019). One important consequence of GDPR is the "Right to Access" where every user has the right to know exactly how their data is used which can be complicated in complex recommendation models. Likewise, there is also the "Right to be Forgotten" which means that every recommendation system must able to delete the user from the system which means the system must be designed to easily remove the user. Thus, the system must be designed from the start to

remove the user from all instances, for example in training data and different databases.

Chapter 2

Theory

2.1 Introduction

A frequently cited definition of machine learning, coined by renowned computer scientist Tom Mitchell, is:

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." (Mitchell et al. 1990)

In other words, machine learning, which can be said to be a subset of the larger field artificial intelligence, is when a computer program is created to perform a task and get better at it while doing it. A famous example is when Arthur L. Samuel created a machine learning algorithm that learned to play checkers (Samuel 1988).

Machine learning and data science are closely related to recommendation systems. For example, in a recommendation system in retail, the algorithm can learn from transaction data of what the customer has bought. Next, machine learning algorithms can be applied to create recommendations based on this data.

The study of recommendation systems is a relatively new field within machine learning and other classical information system techniques such as databases or search engines. The field emerged as an independent research area in the late 1990s (Anand and Mobasher 2005). In recent years, however, the field has become increasingly popular as the web today is full of different data and

choices for the users to make, leading to information overload. Accordingly, information overload is a problem that recommendation systems are trying to solve (Ricci 2011).

Recommendation systems are all over the internet today and we may not even notice them. Internet sites such as Amazon, YouTube, TripAdvisor and services such as Spotify and Netflix are using these systems as a part of their service to their customers (Ricci et al. 2011)

The goals of the recommendation systems can be summarized as (Ricci 2011):

- *Increase the number of items sold.* This is a very important goal for any company. A common example we see every day is additional suggestions we get when buying something online. If we, for example, buy a printer, a suggestion could be white papers to complement our purchase.
- *Sell more diverse items.* Often it is the most popular items that get the most attention. For Netflix, for example, the challenge lies in getting the user to explore other content.
- *Increase user satisfaction.* If a user finds a good recommendation that he or she likes it can increase user satisfaction.
- *Increase user fidelity.* If the company understands the user, it can tailor the experience for that particular user and increase fidelity.
- *Better understand what the user wants.* If the company deploying the recommendation algorithm better understands its customer, they can also improve their service. By doing so, they can also improve the experience for all customers.

There are three kinds of objects that are of importance in a recommendation system: *items*, *users* and *transactions*. Items are the objects that are recommended by the algorithm, examples can be movies, electronics or books. Users can have different goals and characteristics. As a result, the algorithm can take into account other user interactions or features of the users, for example, age or interests. Lastly, transactions are simply the recorded interaction between items and users.

Recommendation systems can be broadly classified into two categories, content-based and collaborative filtering. Content-based filtering is based on what the content of a particular item is. This algorithm recommends items which are similar to the ones that a user has liked in the past. For example, if a person

has downloaded a report, which is about birds, then this algorithm will recommend reports that fall under the same category, in this case, birds or perhaps animals in general (Ricci et al. 2011).

Collaborative filtering works by looking at what other users are downloading, no matter the content. If a person downloads three reports, "report A", "report B" and "report C", and another person downloads "report B", "report C" and "report D", they have similar interests. We can say with some certainty that the first person will also like report "report D" (ibid.).

The recommendation system in this thesis will recommend items on an internet portal depending on what other, similar users are downloading. Using content-based filtering will not work as it requires information about the items themselves, something that is not available. Therefore, collaborative filtering is the appropriate technique to use.

2.2 K-nearest neighbor

The most popular model in collaborative filtering is the K-nearest neighbor model (KNN) and it is the most used recommendation engine today (Sarwar, Karypis, Konstan, and Reidl 2001; Seo and Ho-Jin 2010). For instance, in one review of recommendation systems, the authors noted that its popularity has made K-nearest neighbor synonymous with collaborative filtering in general (Adomavicius and Tuzhilin 2005). The popularity comes mainly from its ease of use and performance. As a result, the model is easy to set up, train and produces good results.

2.2.1 Theory

K-nearest neighbor (KNN) is a relatively simple concept. If a point among other points needs to be classified, KNN finds the *K* closest points, where *K* is a number defined by the user, and then assigns the class label of the majority of its neighbors. Thus, the idea is that if a point is in a neighborhood of a certain class, chances are that the point belongs to that class.

A more mathematical explanation of KNN is the following: we are given a point *q* and we want to know its class *l*, and a training set $X = x_1, l_1 ... x_n$, where x_j is the j-th element and l_j is its class label, the k-nearest neighbors

model will find a subset $X = x_1, l_1 ... x_n Y = y_1, l_1 ... y_k$ such that $Y \in X$and $\sum_1^k d(q, y_k)$ is minimized. Y contains the K points, the nearest neighbors, in X which are closest to the point q. Then, the class label of q is $l = f(l_1 ... l_k)$ (Ricci 2011).

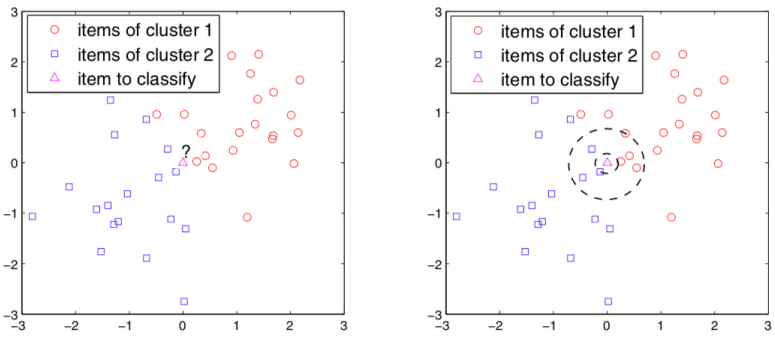

Figure 2.1: Example of K-nearest neighbor with K = 1 and K = 7 (Ricci 2011).

In Figure 2.1, two examples of KNN are shown. In the picture to the left, K is set to 1 and in the picture to the right, K is set to 7. We want to classify the triangle. Depending on how we set K in this case, the triangle can either be classified as a square or a circle. The answer is decided by a majority vote of its neighbors.

2.2.2 Advantages and challenges

The biggest advantage of the K-nearest neighbor model is its simplicity, it is relatively easy to set up and requires little maintenance. Also, the K-nearest neighbor model is conceptually very similar to how you would approach the problem manually if doing collaborative filtering because it is about grouping similar users, neighbors, together (ibid.).

Even though the model continues to be the most popular among recommendation systems, there exist several challenges with K-nearest neighbor. For example, one of the most challenging aspects of the model is the selection of the value K. If the value is too small, the model will be sensitive to noise. On the other hand, if the value is too large, the neighborhood will include points from other classes. This challenge is further illustrated in Figure 2.1 (ibid.).

In addition, the K-nearest neighbor model has been seen to scale poorly when data increases (Sarwar, Karypis, Konstan, and Reidl 2001; Seo and Ho-Jin 2010), highlighting one of the bigger challenges with collaborative filtering in general, the scalability problem (Degemmis et al. 2007). Furthermore, even though new users can easily be added into the model, the neighborhoods need to be recomputed, which takes time and computer power.

In conclusion, although the model is simple and produces good results, one can perhaps argue that it is too simple. This is why newer models focus on dimensionality reduction, which is more suited to today's increasing data requirements.

2.3 Matrix factorization

A common problem with recommendation engines is that the data-set is of high dimensionality. For instance, thousands of users can have interacted with hundreds of items, creating a very large matrix. Furthermore, the data is by definition sparse, the goal of the recommendation system is to fill in the blanks and predict what some users may like. In some data-sets the sparsity can be up to 99 percent (Degemmis et al. 2007; Karypis 2001). In Table 2.1, we see an example of how a data matrix can look like.

Table 2.1: Matrix R, an example of a user/item matrix with 1 meaning that an interaction between the user and item has occurred.

	Item A	Item B	Item C
User 1	-	1	-
User 2	1	-	1
User 3	-	-	1

One proposed solution to these problems is matrix factorization which recently has gained a lot of popularity and is becoming the dominant recommendation model within collaborative filtering (R. Bell et al. 2009; Portugal et al. 2018).

A big competition that Netflix arranged showed that matrix factorization was superior to more classical neighborhood models. They are fast and memory-efficient while also being relatively easy to use (R. Bell et al. 2009).

2.3.1 Theory

In matrix factorization, the goal is to reduce the dimensionality of the input matrix. Thus, if we have the matrix R, we can instead write it as R=PQ where R is the dot-product of P and Q. As a result, we have now reduced R to P and Q, without losing any information. The goal now is to compare the different user and item matrices and find similar users with slightly different data that can be used to estimate the missing information in the matrix. In short, this is done by using the dot-product again, but this time by only using the rows and columns of the value you are trying to estimate (R. Bell et al. 2009). In Figure 2.2, a simple example of matrix factorization is shown.

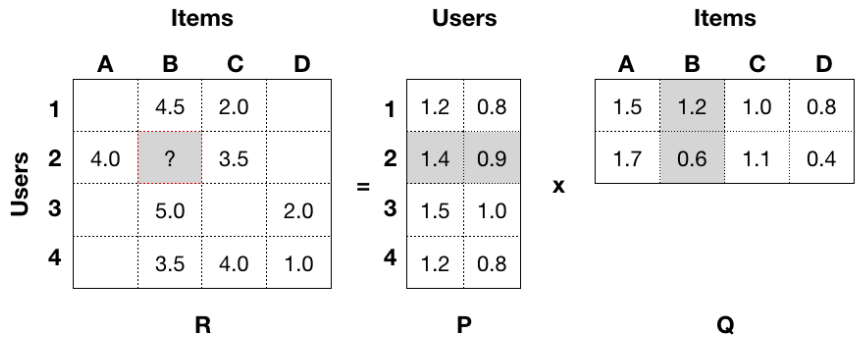

Figure 2.2: Example of matrix factorization with user rated items with the rating spanning from 1 to 5.

A more in-depth explanation of matrix factorization is the following: Matrix factorization map both users and items to a joint latent factor space of dimensionality f, such that user-item interactions are modeled as inner products in that space. Furthermore, all users denoted by the letter ***u*** is associated with the matrix

$$P_u \in \mathbb{R}^f$$

and all items denoted by the letter ***i*** is associated with the matrix

$$Q_i \in \mathbb{R}^f$$

Both P_u and Q_i measure to which extent the item has a positive or negative rating from the user. Furthermore, the resulting dot-product $P_u^T Q_i$, captures the interaction between user ***u*** and item ***i*** and thereby the overall user interest

in the item. Accordingly, this approximates user ***u*** interest in item ***i*** and we finally arrive at the estimate below (ibid.).

$$\hat{R}_{ui} = P_u^T Q_i$$

We can once again look at Figure 2.2 to see a simple example of this.

2.3.2 Advantages and challenges

The biggest advantage of matrix factorization is that it scales well with large amounts of data. As such, it partially solves the scaling problem which has been a big problem for recommendation algorithms in general. Furthermore, another advantage of matrix factorization is that there are incremental algorithms to compute an approximated decomposition. This means that new users and new items can be added without having to completely rebuild the model (Sarwar, Karypis, Konstan, and Riedl 2002).

Like many collaborative filtering techniques, matrix factorization has a transparency problem which means it is hard to know why something is recommended. This is partly because the input matrix is decomposed into smaller matrices. As a result, the numbers in the new matrices, which the recommendations are based on, are difficult to understand as they are decomposed from the actual ratings, see matrices P and Q in Fig 2.2. Matrix factorization also overfits easily, which is something that can be partially prevented by regularization techniques. It can also be computer-intensive to completely rebuild the model because it involves matrix decomposition which can take time if the input matrix is very big (Ricci 2011).

2.3.3 Implementations of matrix factorization

SVD

Singular value decomposition (SVD) is the most famous matrix factorization implementation. It was made famous in 2009 when it won the big Netflix competition in creating the best recommendation system, beating older K-nearest neighbor and statistical models (Koren, Robert Bell, and Volinsky 2009).

The model is a very straightforward implementation from the theory about matrix factorization:

$$\hat{R}_{ui} = \mu + b_i + b_u + Q_i^T P_u$$

Where μ is the standard deviation and b_i and b_u are set to zero in the first iteration.

Furthermore, to estimate the unknowns, the following regularized squared error is minimized:

$$\sum_{r_{ui} \in R_{train}} (r_{ui} - \hat{r}_{ui})^2 + \lambda(b_i^2 + b_u^2 + \|q_i\|^2 + \|p_u\|^2)$$

Where λ is the regularization term.

Lastly, the minimization is done by a stochastic gradient descent and repeated *n* number of epochs.

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

Where γ is the step size.

SVD++

SVD++ is an extension of the regular SVD model that not only looks at explicit feedback but also incorporates implicit feedback. Implicit feedback, in the case of this thesis, means that the model not only uses ratings but also user behavior regardless of what the ratings are (Koren and Robert Bell 2011). As a result, this means that we essentially use two models mixed together, one model that looks at ratings and one model that only looks at what others have downloaded. In addition, the model can also incorporate other implicit feedback if available.

The math for SVD++ is very similar to SVD, we only add a term which represents the implicit feedback, devoid of all ratings:

$$\hat{R}_{ui} = \mu + b_i + b_u + Q_i^T(P_u + |R(u)|^{-1/2} \sum_{j \in R(u)} y_j)$$

Where μ is the standard deviation, b_i and b_u are set to zero in the first iteration and the set R(u) contains the items rated by user u.

The regularized squared error is calculated the same as in SVD but we get a slightly different stochastic gradient descent as follows:

$$
\begin{aligned}
&b_u \leftarrow b_u + \gamma(e_{ui} - \lambda_1 b_u) \\
&b_i \leftarrow b_i + \gamma(e_{ui} - \lambda_1 b_i) \\
&p_u \leftarrow p_u + \gamma(e_{ui} \cdot q_i - \lambda_2 p_u) \\
&q_i \leftarrow q_i + \gamma(e_{ui} \cdot (p_u + |R(u)|^{-1/2} \sum_{j \in R(u)} y_j) - \lambda_2 q_i) \\
&\forall j \in R(u) : y_j \leftarrow y_j + \gamma(e_{ui} \cdot |R(u)|^{-1/2} \cdot q_i - \lambda_2 y_j)
\end{aligned}
$$

Where γ is the step size, λ_1 is the first regularization term and λ_2 is the second regularization term.

NMF

Non-negative matrix factorization is a variant of SVD that only uses positive values in its factorization. In theory, this means that the model will use fewer values for their computations as it only cares about the positive ones. Thus, this can lead to faster computation time while retaining the accuracy of other matrix factorization models. The model has increased in popularity recently and is especially popular in the field of computer vision. According to its authors, the model is fast, easy to implement and produces good results (Luo et al. 2014).

The model is very similar to the general matrix factorization implementation.

$$\hat{R}_{ui} = P_u^T Q_i$$

To remove all the negative values, a different kind of stochastic gradient descent must be used that adapts the step size accordingly:

$$p_{uf} \leftarrow p_{uf} \cdot \frac{\sum_{i \in I_u} q_{if} \cdot r_{ui}}{\sum_{i \in I_u} q_{if} \cdot \hat{r}_{ui} + \lambda_u |I_u| p_{uf}}$$

$$q_{if} \leftarrow q_{if} \cdot \frac{\sum_{u \in U_i} p_{uf} \cdot r_{ui}}{\sum_{u \in U_i} p_{uf} \cdot \hat{r}_{ui} + \lambda_i |U_i| q_{if}}$$

Where λ_u and λ_i are the two regularization terms.

2.4 Evaluation metrics

2.4.1 Error prediction

One of the most common ways of evaluating recommendation models is by using some kind of error prediction. The most popular error metric is the Root Mean Square Error (RMSE) which predicts the accuracy of the predicted ratings for items recommended to users (Ricci 2011). For instance, if the user has downloaded a set of movies, the rating of those movies can be said to be high. Thus, the RMSE will rate how well the recommended movies are rated based on rating of the movies the user have already downloaded.

RMSE by the predicted rating and the actual rating is given by:

$$\sqrt{\frac{1}{|T|}\sum_{(u,i)\in T}(\hat{r}_{ui} - r_{ui})^2}$$

Another popular alternative is Mean Absolute Error (MAE):

$$\frac{1}{|T|}\sum_{(u,i)\in T}|\hat{r}_{ui} - r_{ui}|$$

The main difference between the two is that RMSE has the difference squared, which means that it disproportionately penalizes large errors instead of many small.

2.4.2 Precision and recall

According to a review of the scientific literature on recommendation systems, one of the most common evaluation metrics is precision and recall, with around 50% of all literature using it to evaluate their algorithms (Portugal et al. 2018).

In Table 2.2, an overview of these evaluation metrics is presented. In short, it can be summarized as recall, what proportion of items that a user downloads

were actually recommended? Precision, out of all the recommended items, how many did the user actually download?

Table 2.2: Precision and recall.

	Recommended	Not recommended
Used	True-positive (tp)	False-negative (fn)
Not used	False-positive (fp)	True-negative (tn)

$$Precision = \frac{tp}{tp + fp}$$

$$Recall = \frac{tp}{tp + fn}$$

We can typically expect a trade-off between these two quantities in recommendation systems depending on how many recommendations that are being made. Long lists of recommendations improve recall but will most likely also reduce precision. Furthermore, if the presented recommendation list is fixed, say for example top five recommendations, the most useful metric is precision. However, if the list is not fixed, the most common way of presenting the results is by using comparison curves between the two, so-called precision-recall curves (Ricci 2011).

2.5 Cross-validation

Cross-validation in recommendation systems is not very different from the general machine learning field, to estimate the quality of the model the data-set needs to be properly partitioned into a training set and a test set. This is because it is vital that the performance is estimated on data which take no part in the formulation of the model. In recommendation systems, the data-set is usually split by using one of the following methods: *holdout*, *leave-one-out* or *m-fold cross-validation* (Cremonesi, Turrin, et al. 2008).

The holdout method is perhaps the most straightforward. It is a method that splits the data-set into two parts: a training set and a test set. The exact proportions can be decided by the user but the test proportion is usually between

10% and 30% (Sarwar, Karypis, Konstan, and Riedl 2000b). For instance, in an article about recommendation systems in E-commerce by Sarwar et al, the proportions were set to 20% test and 80% train (Sarwar, Karypis, Konstan, and Riedl 2000a). Overall, in the context of a recommendation system for movies, for example, the partitioning is performed by randomly selecting ratings from the users on the movies and dividing them into a train and test set.

The leave-on-out method works by withholding one rated item every turn while the algorithm is trained on the remaining data. The withheld item is used to evaluate the correctness of the remaining data. Furthermore, this is iteratively done for all the data, withholding different items every turn. Lastly, the results of all evaluations are averaged to compute the final result. This method was used by Karypis in their study (Karypis 2001) but has been noted to have disadvantages such as overfitting and high computational complexity (Cremonesi, Turrin, et al. 2008).

A popular variant of the holdout method described earlier, is the m-fold cross-validation. It works by dividing the data-set into m folds, independent of each other so that they do not overlap. Furthermore, each fold is used once every iteration as a test set and the remaining as training data. The number of folds is usually between 5 and 10 depending on how big the data-set is. The popularity of the model comes from its simplicity and because it can produce results that are less biased than the traditional holdout method that only splits the data set into training and testing sets (James et al. 2013).

2.6 Related work

2.6.1 Models

Schafer et al. 2001 analyze E-commerce recommendation systems in general with a focus on six leading sites including Amazon and eBay. According to the authors, the most popular systems are either based on K-nearest neighbor collaborative filtering which is easy to set up but tends to be slower with larger databases or Bayesian networks that provide the same accuracy but are faster. Furthermore, a problem with Bayesian networks is that the model needs to be rebuilt if the data changes too quickly as it is not a very general approach. One approach can also be to cluster the users beforehand and only analyze one cluster at a time. This can dramatically speed up computations but can hurt

accuracy, especially for users near the edges of the cluster.

In a research paper on how YouTube do their recommendations, Davidson et al. 2010 found that collaborative filtering neighborhood models based on what users are viewing are performing good for their unique data-set. Furthermore, they found that recommendations based on what others are viewing are outperforming top-rated and most-viewed by a large margin in customer satisfaction. YouTube's data-set is unique because the content on the site is user created which means that data about the actual video is often lacking or even missing.

According to Sarwar, Karypis, Konstan, and Reidl 2001, K-nearest neighbor is one of the most successful and widespread models on the web. In addition, K-nearest neighbors have been synonymous with collaborative filtering in general for many years, per a review of recommendation systems (Adomavicius and Tuzhilin 2005). As a result, many articles that describe their use of collaborative filtering are really describing neighborhood models such as K-nearest neighbor.

Cremonesi, Koren, et al. 2010 test the performance between different recommendation systems while using different evaluation metrics. The authors perform tests with the popular open data-sets "MovieLens" and one from Netflix using different algorithms. They choose to exclude the most popular movies so that the recommendations will not only be among them, which is usual in these kinds of data-sets. They test a variety of methods divided into the subcategories neighborhood models (Pearson Correlation, Cosine Similarity) and latent factor models (Singular Value Decomposition) which is closely related to matrix factorization. Interestingly, tests show that accuracy metrics produces other winners compared to RMSE. A variant of the SVD method consistently scored the best on these test-sets using accuracy metrics. The authors conclude by urging the readers to not only use RMSE when evaluating algorithms and to consider SVD while deciding which model to use.

State-of-the-art recommendation models are, according to Li et al. 2018, based on matrix factorization. In the article, the authors use this model to solve the sparsity problem, one of the challenges of collaborative filtering. In addition, using matrix factorization to reduce dimensions has also been seen to beat K-nearest neighbor (Sarwar, Karypis, Konstan, and Riedl 2000a). This result was also found by Ji et al. 2016. In their article, they divide recommendation algorithms into memory-based and model-based collaborative filtering. The main difference between the two is that memory-based approaches look

at the similarity between users, while model-based approaches explores training data and base predictions on that. An example of a memory-based model is K-nearest neighbor and an example of a model-based is matrix factorization. Overall, model-based models are generally seen as better and the authors conclude that matrix factorization is becoming the dominant recommendation algorithm.

2.6.2 Evaluation

Evaluating recommendation algorithms is hard because of several reasons. Firstly, data-sets differ a lot and different algorithms can be suitable for different data-sets depending on the ratio between users and items. Secondly, many researchers argue about what to measure and what metric is important, often between error prediction and accuracy metrics. For instance, accuracy is more relevant for decision problems and error prediction can be more useful when ratings are involved. Furthermore, a combination of the two can be beneficial (Li et al. 2018).

Shani and Gunawardana 2011 specifically discusses evaluation methods in their research paper. They identify three main methods that are used in the field of recommendation systems:

- Offline tests, in which the evaluation is done offline with the use of standard metrics such as error rate or accuracy metrics
- User studies, in which the evaluation is done by asking the users of the system what they think.
- Live tests, in which the evaluation is done on the website directly by redirecting the users to different versions of it. This method is also called A/B testing.

The most common way of doing the offline test is to predict error rate and accuracy with different methods. There exist several ways of doing this and the authors list all of them in decreasing popularity. The most popular method is Root Mean Squared Error (RMSE) which is often used to measure accuracy when predicting ratings, then comes Precision, Recall and F-measure which is used to measure accuracy when predicting user behavior. In addition, there are several methods with special use cases.

User studies have the advantage of interacting with real users, which is the end

goal of all recommendation algorithms. However, the authors also mention several problems with user studies. Firstly, they are very expensive, both in resources and time, to conduct. Secondly, they require a good sampling so that the test users are similar to the real users. In general, user studies seem to attract the most interested users who produce bias.

The best way to make user tests is to deploy the algorithms live online as that is the most realistic setting. This is typically done by redirecting a small percentage of the users to versions of the site with different recommendation algorithms. However, this method can be expensive and time-consuming. There is also the risk of making the users angry as they are required to test functions that are not working properly without their knowing (ibid.). Live tests are also used by Davidson et al. 2010 in an article in which the authors analyze YouTube and how they recommend videos to their users. They do this by redirecting the users to different instances of their site. By doing that, the users can evaluate different recommendation algorithms without even knowing it.

Cremonesi, Koren, et al. 2010 argue that the performance should not only be measured by one metric when evaluating recommendation models. One of the most common ways of evaluating recommendation algorithms is the root mean square error (RMSE) which simply calculates the error between the actual and estimated recommendations. The authors argue that this is not a sufficient metric for evaluating recommendation algorithms, especially top-n recommendations (where the result is an "n" long list):

"While the majority of the literature is focused on convenient error metrics (RMSE, MAE), such classical error criteria do not measure top-N performance. At most, they can serve as proxies of the true top-N experience. Direct evaluation of top-N performance must be accomplished by means of alternative methodologies based on accuracy metrics (e.g., recall and precision)." (ibid.)

Herlocker et al. 2004 agrees with this and says that the most common way of evaluating recommendation algorithms offline tests with standard metrics such as RMSE or Precision and Recall. They provide a quick and economical way of testing and comparing different algorithms. There are two main weaknesses in offline tests. Firstly, data sparsity limits the set of items that can be evaluated and secondly, in the end, it is the user who decides if the recommendation is relevant or not, regardless of the score in standard metrics. This is especially true if the recommendations are of the type find good items, for example, other movies that the user would like. These two weaknesses point to real user-tests

as a complement to standard metrics in recommendation systems.

Furthermore, there is no established standard when evaluating models in recommendation systems which makes it hard to compare the performance between different papers and publications. As a result, the field, in general, has a large collection of different accuracy metrics and new ones are invented every year. The most common metrics are Predictive Accuracy Metrics such as RMSE and Classification Accuracy Metrics such as accuracy and recall. Predicted Accuracy Metrics are widely used and are one of the most popular ways of evaluating recommendation systems. Its widespread use is because of the two main advantages, firstly, it is easy to use and simple to understand, and secondly, it has well proven statistical properties. However, it may be less than ideal when the recommendation algorithm is of the kind find good items (Herlocker et al. 2004).

Li et al. 2018 evaluate matrix factorization models using RMSE while also introducing a stability measurement to complement the error measurement. The authors have found that good generalization is correlated with high stability in their models which makes the stability metric interesting to look at. One drawback is that it introduces yet another evaluation metric to an already crowded field

2.6.3 Challenges

The most cited challenge in recommendation systems seems to be the scalability problem. In an article, which analyses E-commerce recommendation systems for the biggest retailers in the United States, Schafer et al. 2001 argue that scalability together with performance is the most important parts of a recommendation system. That is because these systems tend to be very large with several thousand users and the users expect their recommendations to display within milliseconds on the web-page.

Sarwar, Karypis, Konstan, and Reidl 2001 also identifies the scalability problem as the major challenge but argues that the quality of the recommendations is also important. Firstly, as the user-base grows, scalability can be a big problem. For instance, hundreds of thousands of users mean that all neighborhoods must be searched, which takes computational power. Secondly, users vote with their feet and if they feel that the recommendations are bad or too slow, they will not use the service anymore. Furthermore, the authors argue that these two challenges conflict with each other as a fast algorithm often leads to a

compromise in the quality of the results. The challenges scalability and quality are also mentioned by Ma et al. 2008 as important. In addition, they also mention data sparsity as a challenge which according to Li et al. 2018 is one of the biggest challenges of recommendation systems.

In an a well cited paper, Degemmis et al. 2007 summarized what they see as the biggest challenges in the recommendation field as follows:

- Sparsity problem. Most users only rate or interact with a limited number of items leading to a very sparse user-item matrix.
- Scalability problem. Collaborative filtering systems require an increasing amount of computing power as the number of users and items grows.
- Lack of transparency problem. Collaborative systems today are black boxes which give recommendations but it is hard to understand why they were given.

Which is similar to what the other papers had identified with the addition of the transparency problem.

2.6.4 Data-sets

In the field of recommendation systems, almost all research use the same data-set, the MovieLens data-set (Herlocker et al. 2004). This is because that was one of the first widely available data-sets on the internet for a long time. As a result, many papers are basing their research on the same data (Cremonesi, Koren, et al. 2010; Portugal et al. 2018). The data-set is made out of users and their movie preferences.

Recently, Netflix decided to publish their data in a competition to create the best recommendation algorithm (R. Bell et al. 2009). This data-set is of the same type as MovieLens, i.e. users and their movie preferences. As a result, this has made the field of recommendation systems a field largely about movie recommendations.

2.7 Summary of theory

From the literature study, it is clear that K-nearest neighbor is the most popular and widely used recommendation system on the web today. The model is easy

to use and produces good results. This makes the model ideal for this project as well.

Recently, matrix factorization techniques have become popular. They have shown to be very capable in recent studies, especially in how to handle the the scalability problem. Furthermore, matrix factorization also represents the very state-of-the-art in regards to recommendation techniques today.

In regards to evaluation, RMSE is the most popular technique followed by accuracy metrics such as precision and recall. In the literature study, many articles argue that both should be used to get a better overall picture of how the models perform.

Many studies argue that offline test such as RMSE and accuracy metrics should be complemented with user tests and live tests. User test can be problematic for many reasons such as bias, correct sampling of users and costs (Shani and Gunawardana 2011). Live tests are a very good solution to this. Unfortunately, this is not possible in this project as the web-portal this recommendation system is made for is not up and running yet. Overall, offline tests are the most popular evaluation technique for recommendation systems and it will be used in this project as well.

The most cited challenge for recommendation systems seems to be the scalability problem. Thus, it will be interesting to see if this problem exists in our data-set as well and if the matrix factorization techniques will solve that problem.

Chapter 3

Method

3.1 Research design

For the purpose of this thesis, the recommendation models KNN, SVD, SVD++ and NMF have been evaluated. Two data-sets have been used to evaluate these models. Firstly, an open data-set called MovieLens was used to test the models. Secondly, a proprietary data-set from the principal of this thesis, referred to as the principal's data-set, have been used for most the testing. The results of the models have been evaluated using error metrics (RMSE and MAE) and accuracy metrics (precision and recall). A more thorough description of all the methods used in this thesis can be seen in the following sections.

3.2 Data

3.2.1 MovieLens data-set

The MovieLens database is the most popular data-set in the world of recommendation systems today and is used in many scientific papers (Cremonesi, Koren, et al. 2010; Portugal et al. 2018). Thus, to test the recommendation models built for the purpose of this thesis, an open and publicly available data-set was first used. The data-set can be downloaded for free on the internet (*MovieLens 100K Dataset* 2015).

The particular data-set used was the "MovieLens 100K" movie ratings. This data-set consists of 100,000 ratings from 1000 users on 1700 movies.

3.2.2 The principal's data-set

The customer data was exported from a database in a CSV-format. The database consisted of 134,234 rows, each row representing one download of an item by one user from the year 2013 to 2019. Furthermore, the number of unique users was 4,342 and the number of unique items available for download was 334. An overview of the data and the fields can be seen in Table 3.1.

The data was first anonymized and the names of the users and companies were replaced with unique text strings. Furthermore, rows with incomplete data were removed. In addition, items that had been downloaded more than one time by the same user were removed so that only one occurrence of that download remained. Instead, a new field that counts the number of downloads that user has made of that particular item was created. This was made to measure interest, further explained in section 3.3.

After all pre-processing of the data was complete, 53,512 rows remained.

Table 3.1: Customer data, description of how a row of the database looks like.

Field	Description	Example
Download	Iterator that assigns an int to the download.	34
USSR	The username of the login, usually the email.	erik@kth.se
User ID	Self explanatory, just an int.	22
CRM ID	ID to connect to customer relations database.	25
User domain	Company name.	Innovation AB
Type	What type of customer it is.	Vendor
File	What file was downloaded.	supersecretreport.pdf
File date	What date the file was created (2013-2019).	2013-07-14
Date	What date the file was downloaded (2013-2019).	2019-07-14

3.3 Determine interest

An important question is how do you know if a user that downloaded an item from the internet portal was interested in it, or rather, did the user like the item

he or she downloaded? It became clear that just a download of the item was not sufficient information. The downloaded item could have been a misclick or just not relevant upon further inspection. Furthermore, the download could have happened a long time ago and that does not mean that it is as relevant today.

As a result, to try to quantify the user's interest, an additional field was created for every download. If the user had downloaded the item one time a score of 3 was awarded to it, two times a score of 4, and lastly three and more times a score of 5. Furthermore, with the assumption that items downloaded by the user recently are more relevant, a score from 1 to 5 was given to the item if it was downloaded between 2013 and 2019 (the later date getting a higher score and the earlier a lower score). In addition, if the user had downloaded one item two times on separate dates, the newest date was the one that was counted. Lastly, the interest score was added with the recent download score and averaged.

For instance, if the user "Erik" has downloaded an item two times, 2013 and 2019, it means that this item gets an interest score of 4. Furthermore, the latest date of the two items is relevant here and it being 2019 means that it gets a recent download score of 5. Adding them together and getting the average means that "Erik" has a score of 4.5 on that particular item.

$$Score = \frac{Interest(3-5) + Recentdownload(1-5)}{2}$$

3.4 Models used

The models chosen for this thesis are KNN, SVD, SVD++, NMF and lastly random to act as a baseline. The models were chosen for their popularity in the recommendation field as seen in section 2. Furthermore, the random model was implemented to show the reader what a bad result is. When looking at performance numbers from machine learning models it is sometimes hard to get a sense of what a good result is, the random result can hopefully help with that.

All the models were implemented using the scikit-learn framework in Python. Firstly, the data was imported from the CSV-file into the pandas data-frame so that matrices could be created and be used by all the models. Secondly,

with the help of the scikit-learn framework and Numpy the models were then implemented according to the theory found in section 2 for each model.

The models have different parameters to optimize for. For the KNN model, the only real parameter to change is the value of K. This makes it a very simple model to implement.

For the SVD model and the theory in section 2.3.3 we can see that the parameters we can play with are the step size γ, the regularization term λ, and the number of epochs we run the model. It was recommended to have the standard deviation fixed at 0.1 so that was what we used.

SVD++, being an extension of the SVD model, has more parameters to play with compared to SVD as seen in the theory in section 2.3.3. The parameters are the step size γ, the first regularization term λ_1, the second regularization term λ_2 and the number of epochs we run the model. Once again, the standard deviation was fixed at 0.1.

As we can see in the theory section 2.3.3, the NMF model has three parameters we can change and optimize for. The values are the first regularization term λ_u, the second regularization term λ_i, and the number of epochs we run the model.

The random model is not really a model; it just assigns random recommendations to every user in the system from the total list of items with the same probability. For instance, with the total items in the principal data-set being 334, the probability for each recommended item is 1/334.

3.5 Cross validation

In the experiments, the method of m-fold cross-validation was used. How it works and how it compares to other methods of cross-validation can be seen in section 2.5. This was done because of its popularity in recommendation systems in general and also because it has been shown to be less biased than a simple divide between training and test data.

It was decided that the value of m should be 5 as the data-set used in this report is not that large. This was done to prevent high bias due to the folds being too small. Thus, the results will be shown in 5 different folds.

3.6 Searching for hyper-parameters

A big challenge when doing machine learning is to find what combinations of values for the models that work best for that specific data-set, so-called hyper-parameters. Every model has several unknowns, for instance: learning rate, regularization or number of epochs. To see the unknowns for every model, see section 2.

When searching for hyperparameters, the scikit-learn modules gridsearchCV, and randomizedsearchCV were used. A normal grid search exhaustively goes through all the combinations of the model parameters and presents the best ones. In contrast, a randomized grid search does not try all parameter values, but rather some of them sampled from the specified distributions.

The main benefit of using randomized grid search compared to normal grid search is speed, trying every combination can take quite some time. Studies have shown that randomized search is performing just as good, for a fraction of the computation time (Bergstra and Bengio 2012).

When testing the K-nearest neighbor, only one parameter was interesting, the K. So then a normal grid search was used. For the rest of the models, a randomized grid search was used.

3.7 Evaluation

For the tests in general, the tests were run ten times and the mean of all runs was calculated. This was done because the initial values of the algorithms and cross-validation fold are randomly generated. This helps to prevent any strange results that could happen by chance.

RMSE and MAE were calculated according to the theory in section 2.4.1. The process of doing that is rather straight-forward as it only involves comparing the predicted rating with the actual ratings.

Calculating precision and recall, on the other hand, is more complicated. As mentioned in section 2.4.2, there are many ways of doing this. One common way of doing it in recommendation systems is to look at the top-n recommendations (Cremonesi, Koren, et al. 2010; Karypis 2001). Top-n recommendations mean that the result is a list of n items in descending order with the highest rated item first. Thus, precision and recall are calculated based on these n

recommendations. Furthermore, this way of presenting the result is similar to how this system would be implemented later, as a list of recommendations for the user to look at, which makes this method suitable for this thesis.

Because of this, precision and recall were calculated using the top 10 items for every unique user. Also, to qualify as a relevant rating, a threshold was set at a rating of 3.

Using top-n recommendations means that precision-recall curves are not as relevant. Precision recall curves are supposed to show the trade-off between precision and recall for different thresholds. Although in this case, an increasing recall would mean that the number "n" in the top-n recommendations would also need to be increased, defeating the purpose of using the top-n method altogether.

Lastly, a short comment on how to interpret the results of the standard metrics used in this thesis. RMSE and MAE are both error metrics, ie. a lower error is better with 0 being the best result. For the theory behind these error-metrics see section 2.4.1.

In contrast, the accuracy metrics precision and recall are the other way around, which can be a bit confusing. Thus, a higher value of the accuracy is better with 1 being the best result. For the theory behind these accuracy-metrics see section 2.4.2.

3.8 Software and hardware used

The code was written in Python (version 3.7.3) using the libraries numpy (version 1.13.3) and scikit-learn (version 0.21).

The hardware used was a Macbook air 2013 with the following specifications:

- **CPU:** 1,3 GHz Intel Core i5
- **Memory:** 4 GB 1600 MHz DDR3
- **GPU:** Intel HD Graphics 5000 1536 MB

Chapter 4

Results

4.1 The MovieLens data-set

In this section, a short summary of the results using the open data-set MovieLens is presented. To read more about this data-set and why it was used, see section 3.2.1.

4.1.1 Error rates

In figure 4.1 we can see the error rates for all the models. The results show that SVD++ produces the best results, but not by a large margin. In addition, SVD also produces good result and becomes the second best model on this data-set.

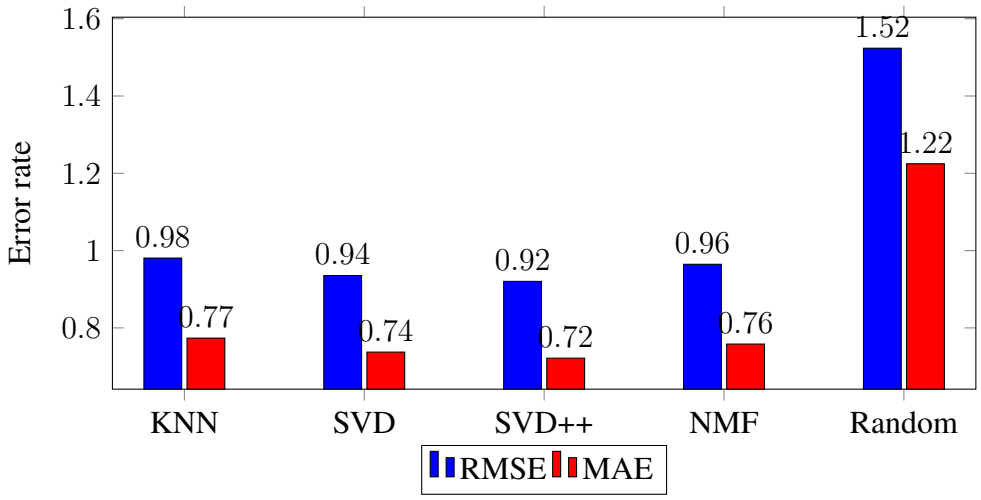

Figure 4.1: RMSE and MAE for MovieLens, a lower value is better.

4.1.2 Accuracy

In figure 4.2, we see a largely similar result. SVD++ once again produces the best results with SVD coming close second.

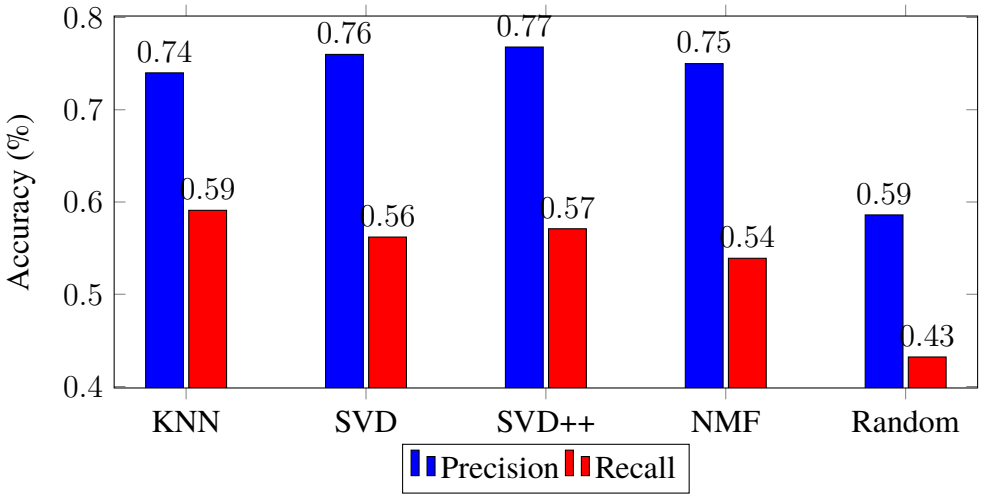

Figure 4.2: Precision and recall for MovieLens, a higher value is better.

4.1.3 Time

Figure 4.3 shows the computation time for all models. Here we can see some interesting results. The more advanced model, SVD++, has a significantly longer training time, almost 43 times longer, than the SVD model. Furthermore, we can also see that all the matrix factorization models have slightly longer training times while having short test times. In contrast, the KNN model is the other way around with a longer testing time compared to its training time.

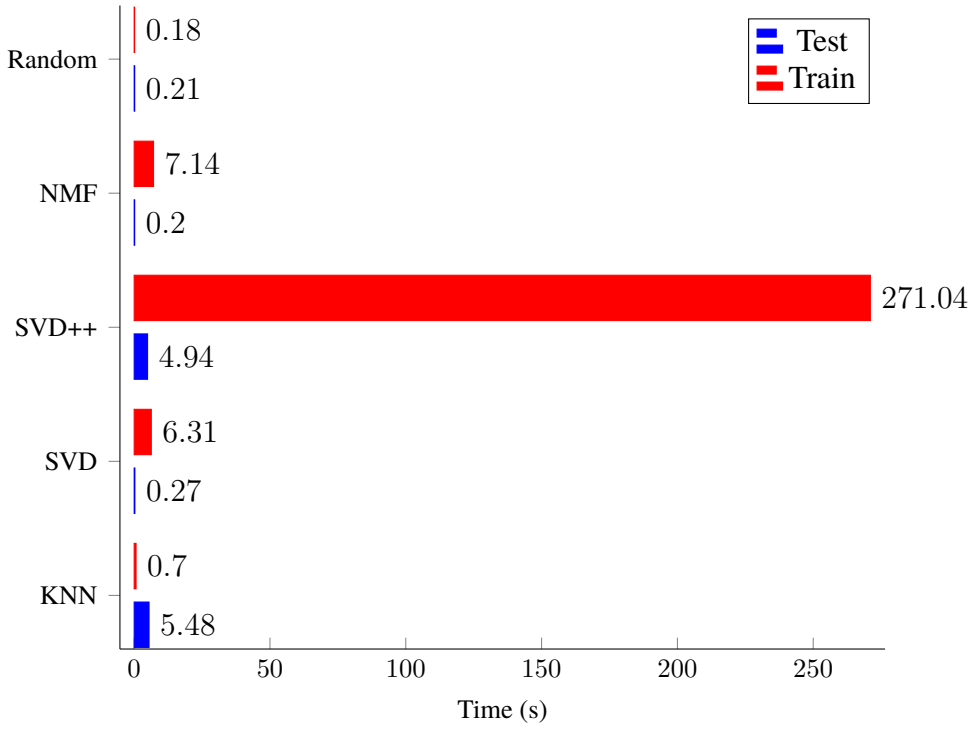

Figure 4.3: Mean training and test time for MovieLens in seconds, a lower value is better.

4.2 The principal's data-set

The following results are performed on the intended data-set of this thesis using the same models as before. For an overview of how the data-set is structured, see section 3.2.2. For a more in-depth look at the results, figures with all the result data is available in the appendix. Furthermore, these results have

more optimized hyper-parameters, as the MovieLens data-set only acted as a test before this data-set was used.

4.2.1 Searching for hyper-parameters

KNN

From the theory in section 2.2.1 we can see that K-nearest neighbor is a relatively simple model and the only parameter that really matters is the value of *K*. Searching manually for a suitable *K* showed the best results for a *K* around 40. After a grid search that searched in the range of 20 to 200, the best result for this data-set for the value of *K* was 52.

SVD

From the theory in section 2.3.3 we can see that the parameters we can play with are the step size γ, the regularization term λ, and the number of epochs we run the model. For all tests, the standard deviation for the initialization was set to 0.1.

Previous work (Koren and Robert Bell 2011) has shown that suitable values for these hyper-parameters are $\gamma = 0.005$ and $\lambda = 0.02$ so those values became the basis for the randomized grid search. Thus, the final values of the hyper-parameters were $\gamma = 0.0041$, $\lambda = 0.033$ and the number of epochs the model was run were 24.

SVD++

SVD++, being an extension of the SVD model, has more parameters to play with compared to SVD as seen in the theory in section 2.3.3. The parameters are the step size γ, the first regularization term λ_1, the second regularization term λ_2 and the number of epochs we run the model. For all tests, the standard deviation for the initialization was set to 0.1.

Looking at previous implementations (ibid.) and their hyper-parameter settings gave us the following values to start with: $\gamma = 0.007$, $\lambda_1 = 0.005$, and $\lambda_2 = 0.015$. After the randomized grid search around these numbers the best results of the model where found for the values $\gamma = 0.0021$, $\lambda_1 = 0.0081$, and

$\lambda_2 = 0.024$. Furthermore, the number of epochs the model was run for were 41.

NMF

As we can see in the theory section 2.3.3, the NMF model has three parameters we can change and optimize for. The values are the first regularization term λ_u, the second regularization term λ_i, and the number of epochs we run the model.

The model had the proposed starting values of $\lambda_u = 0.06$ and $\lambda_i = 0.06$, so that is what formed the basis for the randomized grid search. After the search was complete, we arrived at the values $\lambda_u = 0.041$ and $\lambda_i = 0.052$ and the number of epochs the model ran for was 53.

4.2.2 Error rates

The results for the error metric, as seen in figure 4.4, shows that SVD++ is the model that performs the best on this data-set. The SVD model is very close to SVD++ in both RMSE and MAE, only trailing it with 0.01 in both error metrics. Furthermore, the NMF model performs the worst on this data-set, not counting random, and is far behind all the other models.

Going into the individual results, the KNN model came in third in the error metrics. In addition, KNN had a relatively big difference between the test and train results in both RMSE and MAE with training having almost half the errors compared to testing. The difference between the error rate in the test-set compared to the training-set in every fold seems to indicate that some over-fitting has occurred.

Both SVD and its extension SVD++ had fewer errors than the KNN model. Both models performed better on the test-set than the training-set but not as much as the KNN model. Thus, overfitting seems to be much less for both these models.

Looking at the results in figure 4.4, we can immediately see that NMF performs the worst on this data-set in the error metrics. Similarly to the KNN model, there is a big difference between the performance on the training-set compared to the test-set, almost 50%. Thus, it is likely that overfitting has occurred.

Overall, it was hard to get good results for the NMF model for any value of the hyper-parameters.

The results of all five folds and the difference between training and test data can be found in the appendix.

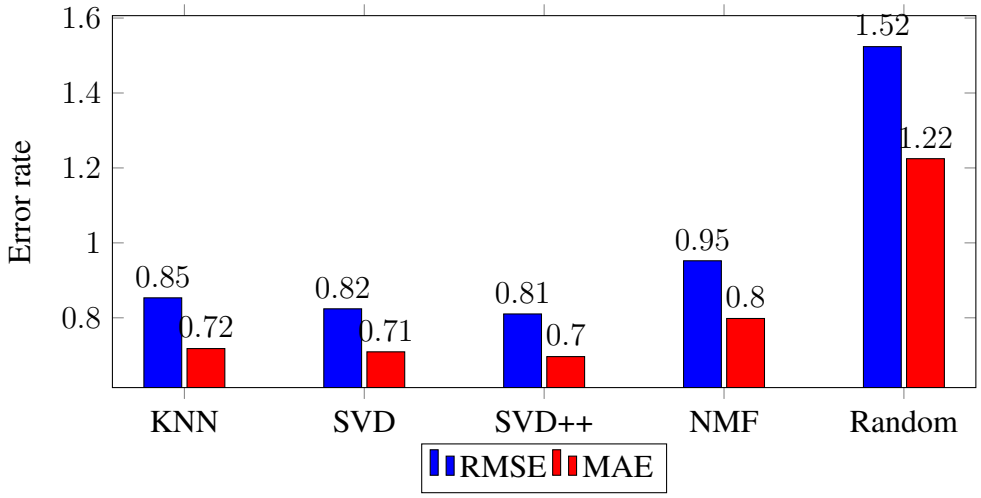

Figure 4.4: Mean RMSE and MAE on test-data, a lower value is better.

4.2.3 Accuracy

In figure 4.5, the accuracy metrics show a very similar story as in the error metrics. The SVD++ model once again shows the best results and once again, the SVD model comes close second. The NMF model, on the other hand, shows a little better results in this metric but is still far behind the other models. This in turn, clearly shows that the NMF model has problems with this particular data-set.

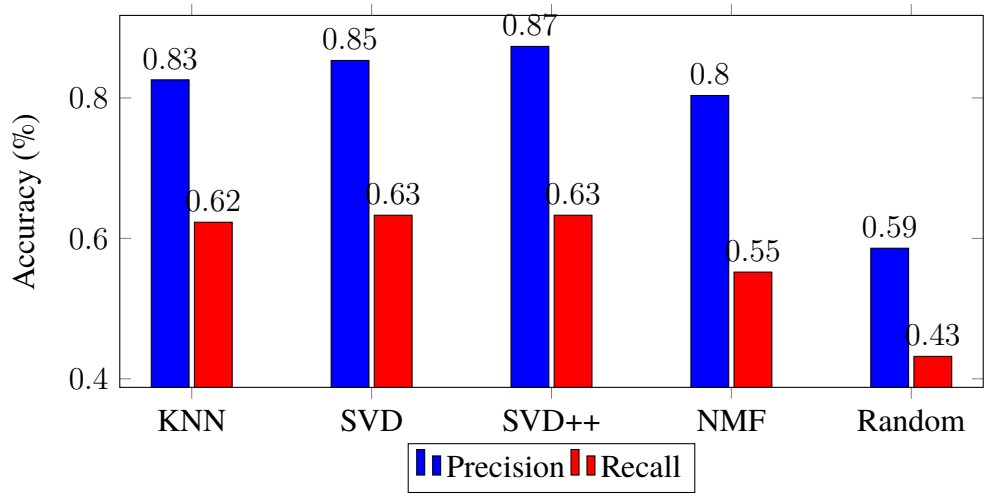

Figure 4.5: Mean precision and recall on test-data, a higher value is better.

4.2.4 Time

In the last plot, shown in figure 4.6, the computation time for all models is shown. The testing time is fast in the case of the models SVD and NMF, coming in at 0.07 seconds and 0.06 seconds. In contrast, training time for the SVD++ model is long relative to the other models at around 22 seconds. Lastly, the KNN model is unique in the sense that it has a longer testing time than training time.

Looking at the results, the model that stands out is the SVD++, especially the training time. Compared to the regular SVD model, training takes quite a bit longer with an average training time of 22.04 seconds. On the other hand, testing the model takes an average of 0.44 seconds which is still slower than the regular SVD. Overall, it seems that the better results of the model come at the price of longer computation time.

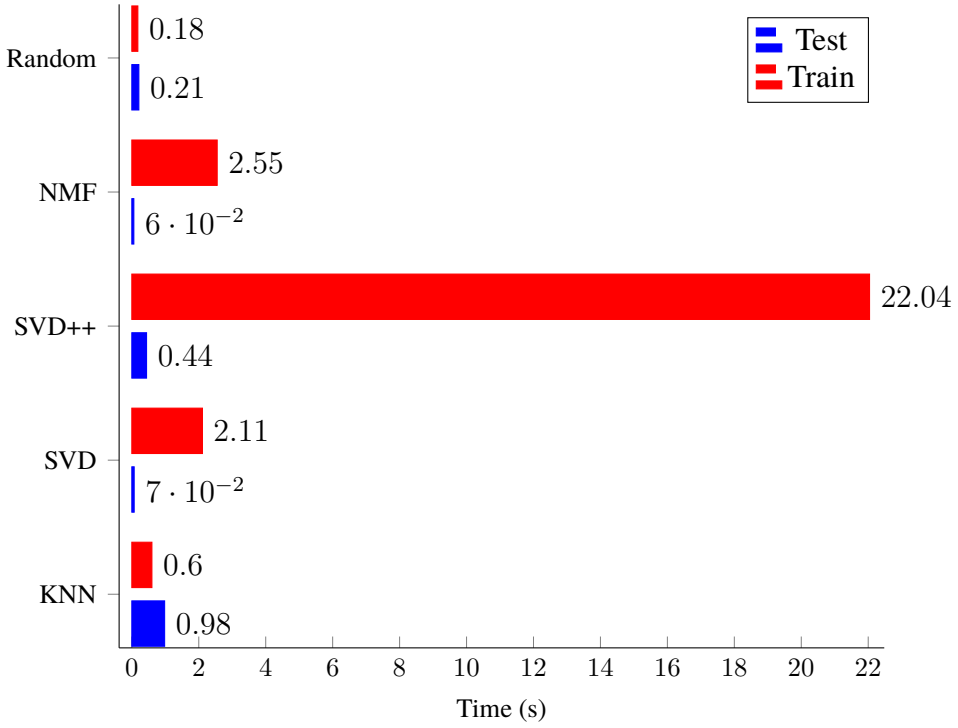

Figure 4.6: Mean training and test time in seconds, a lower value is better.

4.2.5 T-test

Because of the similar results of the SVD and SVD++ model, a statistical T-test was performed to determine if the difference between the two was significant. Using the results of the five different folds for the respective model and the hypothesis that SVD++ is performing better, the following results were produced:

$$P(RMSE) = 0.003$$

$$P(MAE) = 0.001$$

$$P(precision) = 0.0003$$

$$P(recall) = 0.4$$

The T-test was one-sided and computed with two samples (5 data points each) with different variance. Thus, with a confidence level of 95 %, we can say that the hypothesis that SVD++ is performing better than SVD is true for RMSE, MAE and precision, but not for recall.

Chapter 5

Discussion and future work

5.1 Discussion

The purpose of this thesis has been to evaluate different recommendation models based on machine learning. Traditional recommendation algorithms are based on neighborhood models such as K-nearest neighbor while new state-of-the-art models are based on matrix factorization. The principal of this thesis, Radar Ecosystem, wants to implement a recommendation algorithm for their new internet portal. Therefore, several models have been evaluated using both error and accuracy metrics on Radar Ecosystems proprietary customer data. Furthermore, to test the models used in this thesis, an open data-set called MovieLens, was also used. Overall, the goal and challenge of the thesis has been to find which recommendation model that performs the best on Radar Ecosystems unique data-set while also being suitable for their internet portal.

Thus, going back to the initial research question which was:

- How does matrix factorization compare to neighborhood models using standard metrics for generating customer-specific content recommendations on an internet portal from activity data?

From the experiments done it this thesis on both the MovieLens and principals data-set, it is clear that matrix factorization is the superior technique. The results of the error metrics RMSE and MAE show that the matrix factorization techniques SVD and SVD++ consistently produced results with fewer errors than K-nearest neighbors. Furthermore, the accuracy metrics precision and re-

call showed the same results which give credibility to the result. The result is also consistent with the recent trends in recommendation systems where matrix factorization techniques are replacing the once so dominant neighborhood techniques (R. Bell et al. 2009).

The best performing model with both data-sets were the SVD++ model with SVD being close second. On the principal's data-set, the difference was smaller, but still statistically significant on all standards metrics except for recall. The reason recall was not different is probably because of how precision and recall was measured, by using the top-n recommendations as described in section 3.7. This way of measuring means that precision is the most relevant value as we only measure performance on a fixed number of items, effectively restricting how high recall can be.

An interesting result is that almost all recommendation models had better results in both the error and accuracy metric in the principals data-set. In this case, the explanation may lie in how the data is structured. The inclusion of the interest scoring system described in section 3.3 means that certain predictability is added to the data-set. In particular, this is especially true because of the recent download score where older items are considered to be of less interest. Consequently, that field alone, even though it is averaged with the interest score, means that older items are grouped together and the data-set becomes more predictable.

Furthermore, the strange behavior of the non-negative matrix factorization model (NMF) can perhaps be explained by the same reason, a predictable data-set. According to the authors of the model, it is sensitive to overfitting (Luo et al. 2014). In this case, because the model only deals with positive values and discards the rest, the remaining data is possibly even more predictable, which leads to overfitting. The big disparity between the training results and test results seems to support this theory.

To try and determine the user interest in the items they downloaded from the internet portal, a scoring system was created, as described in section 3.3. The biggest question this scoring system is trying to answer is whether or not the user has liked the item he or she has downloaded. This is done with the assumption that more downloads equal a more desirable item. Furthermore, the scoring system is also assuming that an item is less desirable if the user downloaded it a long time ago.

In short, both these assumptions may be wrong. For instance, a user could think that the item he or she downloaded just one time was fantastic and the

system will never know. What is more, a user can accidentally download the same item several times and still think it is useless. In addition, just because an item was downloaded a long time ago does not necessarily mean it is outdated. Overall, the assumptions that had to be made in designing the scoring system were probably justified given the limited information available in the data-set. Although, perhaps a better scoring model could have been implemented that weighted the scores better and made the data-set less predictable.

An interesting aspect of the model testing is the computation time. From the results, we can see that the computation time when building the model is slower in matrix factorization whereas the testing is fast. In contrast, the neighborhood model is the other way around. This seems like a classic computer science trade-off and the question is where that time is best spent. On a webpage, the results are often required to be near-instantaneous, which speaks for the matrix factorization models, assuming that the model building can happen beforehand.

What is more, the computation time can also give us a clue about one of the challenges of recommendation systems, the scalability problem (touched upon in section 2.6.3). If we compare the results of the principals data-set and the bigger MovieLens data-set, we can, for example, see that the KNN model scales worse compared to SVD. In particular, the testing time for the KNN model is 0.98 seconds for the principal's dataset and 5.48 for the MovieLens data-set, which is 5.5 times longer. In contrast, the SVD model goes from 0.07 seconds to 0.27 seconds, which is 3.8 times longer. Furthermore, the best performing model, SVD++, is about 10 times slower in the testing time for the bigger data-set. Thus, the scalability problem seems to be the lowest for the SVD model in these quick tests. Overall, these are only primitive tests with two different data-sets, further testing would need to be done to understand how the models scale with bigger data.

In section 2.6.2, the importance of using both error and accuracy metrics was mentioned by the scientific literature. This was because error metrics, although convenient and easy, does not tell the whole picture. This is especially true if the problem involves decision problems like top-n recommendation, which is the case in this thesis. However, in this thesis, the results would not differ if we only used one of the standard metrics. Both the error and accuracy metrics had the same ordering of the performance of the models, with SVD++ first, SVD second, KNN third and NMF last. On the whole, the main benefit of using both error and accuracy metrics is that it gives the results robustness.

To sum up, the objective of this thesis has been twofold, the academic goal to evaluate recommendation models on standard metrics, and the goal of proposing a recommendation model for the principal that increases customer satisfaction. However, there is a conflict between these two objectives. Specifically, the model that performs the best in the standard metrics may not be suitable for the principal. The question becomes, on what metric are we evaluating these models? Performance? Speed? Easy of use?

In regards to performance, the SVD++ consistently performed the best out of all the models. In addition, the model incorporates implicit data which can be anything from user behavior to other class attributes which makes it scalable and futureproof. However, it is a very slow algorithm. If we look at the computation time results in figure 4.6 we see that it is much slower than the other algorithms for only a few percents better results. Moreover, the algorithm is the hardest to implement and if other implicit data should be used, it will be even harder.

So what does the principal need for their recommendation system? The answer may be a combination of performance, speed, and ease of use. They want to increase customer satisfaction by helping their customers find more useful material on their internet portal. For that purpose, the pure performance metric is not the only important factor. As the model will be implemented on an internet portal, computation time is also important, especially the testing time. Furthermore, even though the scalability problem is not a big issue with the current volume of the data-set, it can perhaps become a problem if the future. Likewise, a model that is relatively easy to implement also makes it future-proof.

So, after discussing the disparity between the principal's goal compared to the academic goal, we are ready to answer the last question:

- Which recommendation model is most suitable for Radar Ecosystems internet portal?

SVD++ had the best performance in the standard metrics and can use implicit data to further increase its performance. However, its complexity makes it slow. Therefore SVD seems like the best alternative. It has good enough performance on the standard metrics and in the computation time, the testing speed is among the fastest. Overall, SVD seems like the best all-round choice.

5.2 Future work

One of the biggest assumptions in this thesis has been whether a user who have downloaded an item on the internet portal likes the item or not. The assumptions that were made in this thesis regarding user interest are explained in section 3.3 and why it can be problematic is explained in section 5.1. Thus, to better understand the user and by that create a better scoring system, more data would need to be collected. For instance, data from the web portal itself, like search queries or time spent looking at items, could be incorporated.

In addition, more data could also be used to improve the models, especially the SVD++ model which combines explicit and implicit feedback. Examples of implicit data could be what type of user it is, in what sector is he or she active, what position in the company does he or she have and more. However, this would perhaps increase the computation time of the SVD++ model even more. If possible, a future project can look at ways of improving the computation time of the model.

One limitation of this thesis is that only offline tests such as error and accuracy metrics were used to test the models because of the internet portal not being ready in time. This means that it is hard to evaluate if the goals of the recommendation system, as described in section 2.1, are fulfilled. For instance, the main goal of the principal was to increase customer satisfaction. Thus, to evaluate these goals (and especially customer satisfaction), the use of live user tests or an evaluation by its users would be useful.

Chapter 6

Conclusion

From tests done on both an open data-set and the principal's data-set, it is clear that the matrix factorization models perform better on both error and accuracy metrics. Furthermore, among the matrix factorization models, SVD++ performed the best with SVD second. In addition, the NMF model exhibited strange behaviour on the principal's data-set which can perhaps be explained by how the data-set is structured.

However, to determine which of the models that is best suited for Radar Ecosystems and their new internet portal, other factors must be considered. For them, performance is not the only criteria. Thus, the recommendation of this thesis is to implement a recommendation system based on the SVD model as it provides a good balance between state-of-the-art performance, ease of use and speed.

Bibliography

Adomavicius, G. and A. Tuzhilin (2005). "Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions". In: *IEEE Transactions on Knowledge and Data Engineering* 17.6, pp. 734–749. ISSN: 1041-4347. DOI: 10.1109/TKDE.2005.99.

Anand, Sarabjot Singh and Bamshad Mobasher (2005). "Intelligent Techniques for Web Personalization". In: *Proceedings of the 2003 International Conference on Intelligent Techniques for Web Personalization*. ITWP'03. event-place: Acapulco, Mexico. Berlin, Heidelberg: Springer-Verlag, pp. 1–36. ISBN: 978-3-540-29846-5. DOI: 10.1007/11577935_1.

Bell, R., Y. Koren, and C. Volinsky (2009). "Matrix Factorization Techniques for Recommender Systems". In: *Computer* 42.8, pp. 30–37. ISSN: 0018-9162.

Bergstra, James and Yoshua Bengio (2012). "Random Search for Hyper-Parameter Optimization". In: *Journal of Machine Learning Research* 13 (Feb), pp. 281–305. ISSN: ISSN 1533-7928.

Cremonesi, Paolo, Yehuda Koren, and Roberto Turrin (2010). "Performance of Recommender Algorithms on Top-n Recommendation Tasks". In: *Proceedings of the Fourth ACM Conference on Recommender Systems*. RecSys '10. event-place: Barcelona, Spain. New York, NY, USA: ACM, pp. 39–46. ISBN: 978-1-60558-906-0. DOI: 10.1145/1864708.1864721.

Cremonesi, Paolo, Roberto Turrin, Eugenio Lentini, and Matteo Matteucci (2008). "An Evaluation Methodology for Collaborative Recommender Systems". In: *2008 International Conference on Automated Solutions for Cross Media Content and Multi-Channel Distribution*. 2008 International Conference on Automated Solutions for Cross Media Content and Multi-Channel Distribution (AXMEDIS). Florence, Italy: IEEE, pp. 224–231. ISBN: 978-0-7695-3406-0. DOI: 10.1109/AXMEDIS.2008.13.

Davidson, James et al. (2010). "The YouTube Video Recommendation System". In: *Proceedings of the Fourth ACM Conference on Recommender*

Systems. RecSys '10. event-place: Barcelona, Spain. New York, NY, USA: ACM, pp. 293–296. ISBN: 978-1-60558-906-0. DOI: 10.1145/1864708.1864770.

Degemmis, Marco, Pasquale Lops, and Giovanni Semeraro (2007). "A content-collaborative recommender that exploits WordNet-based user profiles for neighborhood formation". In: *User Modeling and User-Adapted Interaction* 17.3, pp. 217–255. ISSN: 1573-1391. DOI: 10.1007/s11257-006-9023-4.

Grimes, David Robert (2017). "Echo chambers are dangerous – we must try to break free of our online bubbles". In: *The Guardian*. ISSN: 0261-3077.

Henke, Nicolaus and Noshir Kaka (2019). *Analytics comes of age | McKinsey Analytics | McKinsey & Company*. URL: https://www.mckinsey.com/business-functions/mckinsey-analytics/our-insights/analytics-comes-of-age (visited on 02/09/2019).

Herlocker, Jonathan L., Joseph A. Konstan, Loren G. Terveen, and John T. Riedl (2004). "Evaluating collaborative filtering recommender systems". In: *ACM Transactions on Information Systems* 22.1, pp. 5–53. ISSN: 10468188. DOI: 10.1145/963770.963772.

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani (2013). *An Introduction to Statistical Learning*. Vol. 103. Springer Texts in Statistics. New York, NY: Springer New York. ISBN: 978-1-4614-7137-0 978-1-4614-7138-7. DOI: 10.1007/978-1-4614-7138-7.

Ji, Ke, Runyuan Sun, Xiang Li, and Wenhao Shu (2016). "Improving matrix approximation for recommendation via a clustering-based reconstructive method". In: *Neurocomputing* 173, pp. 912–920. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2015.08.046.

Karypis, George (2001). "Evaluation of Item-Based Top-N Recommendation Algorithms". In: *Proceedings of the Tenth International Conference on Information and Knowledge Management*. CIKM '01. event-place: Atlanta, Georgia, USA. New York, NY, USA: ACM, pp. 247–254. ISBN: 978-1-58113-436-0. DOI: 10.1145/502585.502627.

Key Changes with the General Data Protection Regulation – EUGDPR (2019). URL: https://eugdpr.org/the-regulation/ (visited on 02/12/2019).

Koren, Yehuda and Robert Bell (2011). "Advances in Collaborative Filtering". In: *Recommender Systems Handbook*. Ed. by Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. Boston, MA: Springer US, pp. 145–186. ISBN: 978-0-387-85820-3. DOI: 10.1007/978-0-387-85820-3_5.

Koren, Yehuda, Robert Bell, and Chris Volinsky (2009). "Matrix Factorization Techniques for Recommender Systems". In: *Computer* 42.8, pp. 30–37. ISSN: 0018-9162. DOI: 10.1109/MC.2009.263.

Li, Dongsheng, Chao Chen, Qin Lv, Junchi Yan, Li Shang, and Stephen M. Chu (2018). "Collaborative Filtering with Stability". In: *arXiv:1811.02198 [cs, stat]*. arXiv: 1811.02198.

Luo, X., M. Zhou, Y. Xia, and Q. Zhu (2014). "An Efficient Non-Negative Matrix-Factorization-Based Approach to Collaborative Filtering for Recommender Systems". In: *IEEE Transactions on Industrial Informatics* 10.2, pp. 1273–1284. ISSN: 1551-3203. DOI: 10.1109/TII.2014.2308433.

Ma, Hao, Haixuan Yang, Michael R. Lyu, and Irwin King (2008). "SoRec: Social Recommendation Using Probabilistic Matrix Factorization". In: *Proceedings of the 17th ACM Conference on Information and Knowledge Management*. CIKM '08. event-place: Napa Valley, California, USA. New York, NY, USA: ACM, pp. 931–940. ISBN: 978-1-59593-991-3. DOI: 10.1145/1458082.1458205.

Mitchell, T, B Buchanan, G DeJong, T Dietterich, P Rosenbloom, and A Waibel (1990). "Machine Learning". In: *Annual Review of Computer Science* 4.1, pp. 417–433. DOI: 10.1146/annurev.cs.04.060190.002221.

MovieLens 100K Dataset (2015). GroupLens. URL: https://grouplens.org/datasets/movielens/100k/ (visited on 02/21/2019).

Portugal, Ivens, Paulo Alencar, and Donald Cowan (2018). "The use of machine learning algorithms in recommender systems: A systematic review". In: *Expert Systems with Applications* 97, pp. 205–227. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2017.12.020.

Ricci, Francesco, ed. (2011). *Recommender systems handbook*. OCLC: ocn373479846. New York: Springer. 842 pp. ISBN: 978-0-387-85819-7 978-0-387-85820-3.

Ricci, Francesco, Lior Rokach, and Bracha Shapira (2011). "Introduction to Recommender Systems Handbook". In: *Recommender Systems Handbook*. Ed. by Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. Boston, MA: Springer US, pp. 1–35. ISBN: 978-0-387-85819-7 978-0-387-85820-3. DOI: 10.1007/978-0-387-85820-3_1.

Samuel, Arthur L. (1988). "Some Studies in Machine Learning Using the Game of Checkers. II—Recent Progress". In: *Computer Games I*. Ed. by David N. L. Levy. New York, NY: Springer New York, pp. 366–400. ISBN: 978-1-4613-8716-9. DOI: 10.1007/978-1-4613-8716-9_15.

Sarwar, Badrul, George Karypis, Joseph Konstan, and John Reidl (2001). "Item-based collaborative filtering recommendation algorithms". In: *Proceed-*

ings of the tenth international conference on World Wide Web - WWW '01. the tenth international conference. Hong Kong, Hong Kong: ACM Press, pp. 285–295. ISBN: 978-1-58113-348-6. DOI: 10.1145/371920.372071.

Sarwar, Badrul, George Karypis, Joseph Konstan, and John Riedl (2000a). "Analysis of Recommendation Algorithms for e-Commerce". In: *Proceedings of the 2Nd ACM Conference on Electronic Commerce*. EC '00. event-place: Minneapolis, Minnesota, USA. New York, NY, USA: ACM, pp. 158–167. ISBN: 978-1-58113-272-4. DOI: 10.1145/352871.352887.

— (2000b). *Application of Dimensionality Reduction in Recommender System - A Case Study:* Fort Belvoir, VA: Defense Technical Information Center. DOI: 10.21236/ADA439541.

— (2002). "Incremental singular value decomposition algorithms for highly scalable recommender systems". In: *Fifth international conference on computer and information science*. Vol. 27. Citeseer, p. 28.

Schafer, J. Ben, Joseph A. Konstan, and John Riedl (2001). "E-Commerce Recommendation Applications". In: *Data Mining and Knowledge Discovery* 5.1, pp. 115–153. ISSN: 1573-756X. DOI: 10.1023/A:1009804230409.

Seo, Eugene and Choi Ho-Jin (2010). "MOVIE RECOMMENDATION WITH K-MEANS CLUSTERING AND SELF-ORGANIZING MAP METHODS:" in: *Proceedings of the 2nd International Conference on Agents and Artificial Intelligence*. 2nd International Conference on Agents and Artificial Intelligence. Valencia, Spain: SciTePress - Science, pp. 385–390. ISBN: 978-989-674-021-4 978-989-674-022-1. DOI: 10.5220/0002737603850390.

Shani, Guy and Asela Gunawardana (2011). "Evaluating Recommendation Systems". In: *Recommender Systems Handbook*. Ed. by Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. Boston, MA: Springer US, pp. 257–297. ISBN: 978-0-387-85820-3. DOI: 10.1007/978-0-387-85820-3_8.

Appendix A

Results for every model

A.0.1 Random

Table A.1: Results of all five folds and the mean for the random model.

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean
RMSE (test)	1.5250	1.5245	1.5166	1.5245	1.5214	1.5224
MAE (test)	1.2260	1.2230	1.2168	1.2250	1.2218	1.2225
RMSE (train)	1.5182	1.5202	1.5227	1.5236	1.5195	1.5208
MAE (train)	1.2189	1.2197	1.2246	1.2234	1.2185	1.2210
Precision	0.581	0.583	0.583	0.597	0.594	0.585
Recall	0.43	0.438	0.447	0.428	0.43	0.427
Training time	0.16	0.18	0.20	0.19	0.19	0.18
Test time	0.21	0.21	0.21	0.21	0.21	0.21

A.0.2 K-nearest neighbor

Table A.2: Results of all five folds and the mean for the K-nearest neighbor model.

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean
RMSE (test)	0.8509	0.8590	0.8489	0.8543	0.8534	0.8533
MAE (test)	0.7158	0.7264	0.7097	0.7185	0.7190	0.7179
RMSE (train)	0.4398	0.4409	0.4394	0.4395	0.4385	0.4396
MAE (train)	0.3227	0.3234	0.3223	0.3220	0.3204	0.3222
Precision	0.835	0.823	0.829	0.816	0.826	0.8258
Recall	0.619	0.627	0.628	0.628	0.615	0.623
Training time	0.83	0.58	0.54	0.54	0.53	0.60
Test time	0.99	0.97	0.97	1.02	0.94	0.98

A.0.3 SVD

Table A.3: Results of all five folds and the mean for the SVD model.

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean
RMSE (test)	0.8332	0.8210	0.8261	0.8161	0.8237	0.8240
MAE (test)	0.7179	0.7082	0.7102	0.7017	0.7080	0.7092
RMSE (train)	0.6249	0.6247	0.6232	0.6258	0.6255	0.6248
MAE (train)	0.5509	0.5504	0.5499	0.5514	0.5513	0.5508
Precision	0.856	0.856	0.851	0.850	0.854	0.8534
Recall	0.627	0.637	0.650	0.628	0.620	0.633
Training time	2.10	2.14	2.12	2.10	2.11	2.11
Test time	0.08	0.07	0.07	0.07	0.07	0.07

A.0.4 SVD++

Table A.4: Results of all five folds and the mean for the SVD++ model.

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean
RMSE (test)	0.8067	0.8073	0.8078	0.8120	0.8178	0.8103
MAE (test)	0.6900	0.6925	0.6948	0.6980	0.6971	0.6965
RMSE (train)	0.7031	0.7030	0.7019	0.6989	0.6997	0.7013
MAE (train)	0.6009	0.6002	0.5991	0.5968	0.5966	0.5987
Precision	0.863	0.870	0.871	0.873	0.878	0.8734
Recall	0.628	0.628	0.635	0.631	0.634	0.633
Training time	21.75	22.38	22.01	21.67	22.39	22.04
Test time	0.45	0.45	0.44	0.43	0.46	0.44

A.0.5 NMF

Table A.5: Results of all five folds and the mean for the NMF model.

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean
RMSE (test)	0.9574	0.9491	0.9497	0.9588	0.9461	0.9522
MAE (test)	0.8063	0.7951	0.7964	0.8040	0.7894	0.7982
RMSE (train)	0.4836	0.4825	0.4853	0.4821	0.4837	0.4834
MAE (train)	0.3720	0.3720	0.3734	0.3716	0.3732	0.3724
Precision	0.792	0.801	0.804	0.805	0.802	0.8035
Recall	0.558	0.562	0.551	0.551	0.567	0.552
Training time	2.53	2.54	2.59	2.54	2.57	2.55
Test time	0.08	0.05	0.05	0.05	0.05	0.06

TRITA -EECS-EX-2019:733