MASTER'S THESIS 2022

Designing and implementing a recommender system for an E-learning platform

Astrid Ekman

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2022-02

Designing and implementing a recommender system for an E-learning platform

Design och implementering av ett rekommendationssystem för en E-learning plattform

Astrid Ekman

Designing and implementing a recommender system for an E-learning platform

Astrid Ekman
as4154ek-s@student.lu.se

February 2, 2022

Master's thesis work carried out at Grade AB.

Supervisor: Rasmus Ros, rasmus.ros@cs.lth.se
Examiner: Elin Anna Topp, elin_anna.topp@cs.lth.se

Abstract

Today many web-based companies rely on recommender systems, as these systems may enhance the user experience by presenting only relevant products. In this master thesis, a recommender system is designed and implemented based on data given by the E-learning company Grade AB. The system combines collaborative filtering with demographic filtering by first categorizing the users with k-means clustering and then running matrix factorization on each cluster. The data used for training is historical user-course interactions, meaning that there is a lack of negative feedback. This may affect the model, why methods that deal with this are presented in the report. Furthermore, since there is one unique model for each of Grade AB's clients, this master thesis also investigates how sensitive the client-specific models are for hyperparameters. Offline evaluation is performed on the models to show how well they perform, as well as to optimize hyperparameters for three of Grade's clients. The results of this master thesis show that matrix factorization on demographic clusters performs better than matrix factorization on a single cluster, consisting of all data available. The results also show that the models perform better the larger the training data sets are. However, the metrics recall and NDCG of the models just barely beat the baseline, always recommending the most popular courses. This is probably due to the selection of metrics and constraints of offline evaluation, which is also discussed in this master thesis.

Keywords: recommender system, E-learning, collaborative filtering, matrix factorization, demographic filtering, offline evaluation

Acknowledgements

I would like to thank Grade AB for providing the data set that made this master thesis possible. In particular, I would like to thank my supervisor Rickard Nygren, Vice VD at Grade AB, and Arvid Pilhall, developer at Grade AB, for giving me helpful comments along the way. Additionally, a big thank you to my supervisor Rasmus Ros, Doctoral student at Software Engineering, for valuable feedback guiding me in the right direction during this project. Finally, I would like to thank my examiner Elin Anna Topp for important final comments, resulting in my master thesis being approved.

Astrid Ekman, January 2022

Contents

Acronyms

- **CBF** - Content Based Filtering.
- **CF** - Collaborative Filtering.
- **DF** - Demographic Filtering.
- **DMF** - Demographic Matrix Factorization.
- **IID** - Independent and Identically distributed.
- **MF** - Matrix Factorization.
- **NDCG** - Normalized Discounted Cumulative Gain.
- **SVD** - Singular Value Decomposition.

Chapter 1
Introduction

The purpose of recommender systems is to help users filter among items on web pages so that relevant items do not get lost in the large mass. During the past years, the web has emerged explosively, and consequently also the range of choices that are presented to customers. The desire of companies to personalize the content to their customers, in order to improve the user experience, has driven the development of recommender systems. The main idea is to analyze data specific to customers and items, and through this make predictions on what customers might be interested in [24]. This project will focus on recommender systems in the E-learning context and will be carried out at Grade AB, which will also provide the data used for training. In the next sections related work will be presented, followed by more information about Grade AB and details on this master thesis, including research questions.

1.1 Related Work

Recommender systems has been a hot topic during the last decades, because of the rapid development of web services. Recommender systems are common both in e-commerce and in online advertisement, where buyers are recommended products that might interest them. The fact that recommender systems may generate an uplift in revenue, makes it critical for many companies to develop such a system. Furthermore, recommender systems may enhance the customers' experience, leading to the company's product standing out from competitors [33]. Netflix is an example of a company having a lot of available items (movies and series) and is therefore dependent on accurate recommendations for their users, to maintain their status on the market. In 2006, Netflix released a data set [27], containing 100 million movie ratings made by different customers, and announced the Netflix Prize Competition. The goal of the competition was to create a recommender system, reaching a certain accuracy, and the winning team would get $1,000,000. This resulted in advancement in the field of recommender systems, particularly in Collaborative filtering, which is one of the most common approaches [4, pp. 75]. Other articles, such as [39], suggest combining collaborative filtering with other

filtering techniques. There are different alternatives of how to combine different filtering techniques, but [1, 32, 39] suggest categorizing the users into demographic clusters and then performing collaborative filtering once on each cluster.

There are many studies on recommender systems, but not much research has been done on recommender systems in the E-learning context. The goal of E-learning companies is to enhance the competence of the customers, and recommender systems can indeed be used to make the customers choose courses that help them with this.

1.2 Grade AB

Grade AB is a company that provides a web-based full-scale Learning Management System. The platform was created in the middle of the 1990s when lecturers at Lund University were struggling with distributing course material to the growing number of students. The platform was at this time called LUVIT and the main product was to provide students with learning material. Gradually the product was improved and today Grade has around 40 employees and 100 companies and organizations as customers and offers the following:

- **Grade Engage**, which measures the engagement of the employees
- **Grade Talent**, which makes agile human resources processes possible
- **Grade Learning**, which increases the competencies of the employees through courses, mainly carried out on the web
- **Grade Analytics**, which visualizes data from Engage, Talent, and Learning

The main purpose of Grade's product is to, through these four parts, make employees in companies and organizations motivated to achieve better results. There is an E-learning production tool integrated into Grade's platform, which can be used by the client to create web courses. These courses may also be produced in close cooperation with Grade's own E-learning team, to enhance the course content further. Furthermore, Grade Talent makes it possible for each user to be assigned to different roles and competencies, which then can be used by Human Resources, but also by the user itself to increase the motivation. To make it possible for employees to discover relevant courses, it is important to adapt the content to each user and its needs. To achieve this, it is necessary to analyze the users' unique characteristics and find out what the recommendations should be based on. Possible characteristics to base the recommendations on is which unit the user belongs to, and which roles, competencies, and certificates the user has. Each of Grade's clients offers different courses and the units, roles, competencies, and certificates are more or less unique for each client.

1.2.1 Current recommendations

Grade already has a prototype of a recommender system where the recommendations are based on employees' roles and competencies. However, it is not developed enough as it is implemented today. The current recommendation system is based on the following:

- Competence gap. This means the gap between the user's current score on a certain competence and the required score. If there is a competence gap, the user will be recommended courses that are related to the competence.
- Courses related to roles. If a user has a certain role, it can be recommended courses that are tagged with this specific role.

This recommendation system is naïve and requires much administrative work since competencies and roles manually have to be related to courses. Grade wants to improve the existing functionality to avoid this and to give the users even more relevant recommendations.

1.3 Research questions and outline

The goal of this Master Thesis is to investigate how well traditional recommendation algorithms, such as matrix factorization and clustering of users, work in the E-learning context. Further research questions are summarized below:

- How do we deal with implicit feedback and the lack of negative feedback?
- How can a hybrid recommender system consisting of collaborative and demographic filtering be designed?
- How sensitive for hyperparameters are the client-specific recommender systems, looking at the difference between the evaluation scores and the baseline scores?
- What are the limitations of offline evaluation systems, and how can we overcome these?

In the next chapter, relevant theory about recommender systems will be presented. This includes different types of user feedback, filtering techniques and how to combine them, and finally how to evaluate the performance of a recommender system. In chapter 3, the method used to implement the recommender system will be explained. This will include how the data was split into train and test sets, how the users were grouped using k-means clustering, and how matrix factorization was then performed on each cluster. Chapter 3 will also include information on how the evaluation of the system was done. In chapter 4, the evaluation results of the different models will be presented and discussed, along with a section suggesting future work on the topic. Lastly, in chapter 5 the conclusions of this master thesis will be summarized.

Chapter 2
Theory

This chapter will cover theory about user feedback and popular approaches used in recommender systems, including memory and model-based collaborative filtering (CF), demographic filtering (DF), and content based filtering (CBF). Different methods of combining filtering techniques will also be discussed, followed by methods on how to evaluate a recommender system.

2.1 User feedback

Many recommender systems rely on data that connects users with items. These user-item interactions can contain both explicit and implicit information concerning the users' preferences. The explicit feedback used in recommender systems is usually ratings and is a quantification of the user's preferences. This kind of feedback is usually reliable since the user has expressed his or her opinion explicitly [17]. The feedback can be either scalar, meaning the rating is for example between 0 and 10, or binary, for instance like or dislike [17]. The disadvantage of explicit feedback is that it may be difficult to collect. It requires the user to give ratings, which may disturb the experience [2]. Implicit feedback, on the other hand, does not require the user to share opinions explicitly. Implicit feedback is only an approximation of the user's opinion, and assumptions have been made to some extent. This could for example be that if a customer clicks a certain item, we may assume that it is interested in it [17]. A negative aspect of implicit feedback, except that it usually contains redundancy and is difficult to summarize, is that it may lack negative feedback. If a customer does not click a specific item, it does not have to mean that the customer actively did not like the item [21]. A suggested method that deals with this problem will be presented in section 2.2.3. In the case of this master thesis, implicit feedback was used, since the rating of courses is only optional for Grade's customers.

		Items					
		1	*2*	...	*i*	...	*m*
Users	*1*	5	3		1	2	
	2		2				4
	:			5			
	u	3	4		2	1	
	:					4	
	n			3	2		
	a	3	5		?	1	

Figure 2.1: An example of a user rating matrix, where the number in each cell corresponds to the user's rating on the specific item. The empty cells correspond to a user not having rated the specific item, **a** denotes the active user, and the question mark denotes that user **a**'s rating on item **i** should be predicted [24].

2.2 Collaborative filtering

The term collaborative filtering was coined in the 90s, but it is based on a well-known concept, that is opinion sharing. People tend to base their decisions on input from others, especially from people that usually have similar opinions. Instead of asking several people about their opinions, this can be done by a computer taking thousands of people's opinions into account [35]. Two users are considered similar if they have given the same items similar ratings. This means that no information about the users or items themselves is needed, only information about the user-item interactions [22]. The user-item interactions can be represented by a so-called user rating matrix, which describes how users have rated different items, see figure 2.1. Usually, all unknown values in the matrix are predicted, and then the items with the highest predicted ratings will be recommended to the user [24]. In the next sections, two types of collaborative filtering techniques will be explained: memory-based (2.2.1) and model-based (2.2.2) collaborative filtering. In section 2.2.3, solutions to the problem with lack of negative feedback will be discussed.

2.2.1 Memory based CF

Memory-based, also known as Neighbourhood-based, collaborative filtering can be either user-based or item-based. Assume that user i's rating on item j, r_{ij}, is supposed to be predicted and that User based collaborative filtering is to be used. Then the similarities between the target user i and all other users will be calculated. Users are considered similar if they give the same items similar ratings. When the similarity scores are determined, the weighted average of the k most similar users will be calculated with the similarities as weights. The result is the prediction of user i's rating of item j, see equation 2.1. Furthermore, the similarity

scores can be calculated in different ways. The most common are Pearson Correlation and Cosine Similarity, see equations 2.2 and 2.3.

$$r_{ij} = \frac{\sum_k Sim(u_i, u_k) r_{kj}}{\#ratings} \tag{2.1}$$

$$Sim_{Pearson}(u_i, u_k) = \frac{\sum_j (r_{ij} - r_i)(r_{kj} - r_k)}{\sqrt{\sum_j (r_{ij} - r_i)^2 \sum_j (r_{kj} - r_k)^2}} \tag{2.2}$$

$$Sim_{Cosine}(u_i, u_k) = \frac{r_i \cdot r_k}{|r_i||r_k|} = \frac{\sum_{j=1}^{m} r_{ij} r_{kj}}{\sqrt{\sum_{j=1}^{m} r_{ij}^2 \sum_{j=1}^{m} r_{kj}^2}} \tag{2.3}$$

In the item-based collaborative, on the other hand, two items are considered similar if they get similar ratings from the same user. The predictions are calculated analogously as for the case with user-based collaborative filtering. The advantage of this method is that it tends to be more stable than user-based collaborative filtering, since ratings on items will probably not change significantly over time which the users' taste might do [22]. A memory-based recommender system usually becomes more accurate than a model-based (see section 2.2.2), since it calculates all similarities and bases the recommendation on the extremely similar entities. However, in memory-based collaborative filtering, there is a significant risk that the user-item matrix is sparse, resulting in inaccurate predictions. Additionally, this technique may be computationally heavy, leading to a long response time [36].

2.2.2 Model based CF: Matrix factorization

As mentioned in the previous section, memory based collaborative filtering tends to be very slow and may suffer from sparse user-item matrices. This is the reason why model based filtering was used in this master thesis. This kind of model is also based on user-item interactions and assumes that there is a latent model that can describe these interactions. These latent representations have a mathematical meaning, but they may be difficult for a human to interpret. With the use of model based collaborative filtering, all similarities do not have to be calculated every time [33]. One of the most common model based approaches uses matrix factorization.

Assume that a user has finished the courses "Global sea level change", "Solar energy" and "Renewable energy". These are probably not based on three different opinions, but it shows that the user is interested in the environmental field. Maybe there are more courses in this field that would be suitable for the user. This environment category is an example of a latent feature. Matrix factorization makes it possible to extract latent features, even though they are not usually as concrete as the environment category. Depending on how much a course fits into a category, combined with how aligned a user is with this set of latent features, recommendations can be made. Even though two users have finished different courses, the users can be considered similar if they have similar underlying preferences, latent factors. This is an advantage of model based CF, that memory based CF does not have [22].

The idea of matrix factorization is to decompose the user-item interaction matrix into two or more lower dimensional matrices. The goal is to create an item-related vector $q_i \in \mathbb{R}^k$

and a user-related vector $p_u \in \mathbb{R}^k$, so that the prediction of user u's rating of item i, $\hat{r}_{ui}$, can be described by equation 2.4. Note that k is the number of latent features.

$$\hat{r}_{ui} = q_i^T p_u \tag{2.4}$$

Determining a matrix factorization model may be challenging. Singular Value Decomposition (SVD) is an established method that decomposes a matrix into lower dimensional matrices, extracting latent features. The problem is that conventional SVD requires a matrix where all elements are known, which is not the case in collaborative filtering. The missing entities have to be computed so that the loss function is minimized. Stochastic gradient descent is a technique that has proven to be effective for this [23]. To avoid overfitting, only relying on the relatively few known values in the user-item interaction matrix, the loss function should be regularized. Using squared error as loss function, adding a regularization term with a regularization constant λ gives equation 2.5, which is the minimization problem that we want to solve. Note that the convexity of equation 2.5 guarantees convergence. Also note that κ is the set of all (u, i) pairs where r_{ui} is known.

$$\min_{p,q} \sum_{(u,i)\in\kappa} (r_{ui} - q_i^T p_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2) \tag{2.5}$$

The idea of the Stochastic Gradient Descent technique is to optimize equation 2.5, by iteratively updating p_u and q_i in the opposite direction of the gradient. To do this the derivatives of $f(p_u, q_i) = (r_{ui} - q_i^T p_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2)$ have to be calculated. The algorithm works like this:

1. Initialize p_u and q_i randomly
2. Until the loss is small enough, for each known user-item pair:
   (a) Calculate $\frac{\partial f}{\partial p_u}$
   (b) Calculate $\frac{\partial f}{\partial q_i}$
   (c) Update p_u: $p_u \leftarrow p_u - \alpha \frac{\partial f}{\partial p_u}$
   (d) Update q_i: $q_i \leftarrow q_i - \alpha \frac{\partial f}{\partial q_i}$
3. Make predictions according to $\hat{r}_{ui} = q_i^T p_u$

The prediction error associated to the prediction of item i given by user u can be defined as $e_{ui} := r_{ui} - q_i^T p_u$. Given this, the algorithm can be written more concisely. Note that γ is the learning rate [19].

1. Initialize p_u and q_i randomly
2. Until the loss is small enough, for each known user-item pair:
   (a) Update p_u: $p_u \leftarrow p_u + \gamma(e_{ui} q_i - \alpha p_u)$
   (b) Update q_i: $q_i \leftarrow q_i + \gamma(e_{ui} p_u - \alpha q_i)$
3. Make predictions according to $\hat{r}_{ui} = q_i^T p_u$

One of the weaknesses of collaborative filtering is the so-called cold-start problem. This occurs when there are no ratings yet, which may occur either when there is a new item introduced to the system, or there is a new user with no previous ratings. This results in the predictions, made by the collaborative filtering recommender system, being unusable. A simple way to deal with this is to wait until the new user has made some ratings, or the new item has been given some ratings. The problem is that if an item never gets recommended, or a user never gets any recommendations, there will not be any new ratings. This problem has led to the development of hybrid recommender systems, combining collaborative filtering and some other kind of filtering, such as content based filtering or demographic filtering [6], which will be described in the next sections.

2.2.3 One-class collaborative filtering

As earlier mentioned, a problem with implicit feedback is that the data set may lack negative feedback. If a user finishes a course, one may assume that it was positive to the course. However, if the user does not finish a course, it does not necessarily mean that the user did not like it. It can also mean that the user simply is not aware of the course's existence. A solution to this is to give weights to the error terms in equation 2.5, see equation 2.6. If r_{ui} is known (positive examples), w_{ui} should always be one. However, if r_{ui} is unknown ("negative" examples), w_{ui} should be smaller since those elements in the user-item matrix are less reliable. There are a few different ways to set the weights for the implicit negative examples. Either the missing elements can be assumed to be negative with the same possibility for all users or all items. This means that the weights for all missing elements will be $\delta \in (0, 1)$. Another alternative for the weights is to assume that if a user has many positive examples, the missing elements are more likely to be negative. The third alternative is to assume that if an item has few positive examples, it is more likely that the missing elements are negative. The three alternatives are summarized in table 2.1 [29]. In the case of this master thesis, w_{ui} was set to 0.15 for all negative examples, that is uniform weights. The reason for this was the simplicity of the method. However, other weights should be investigated in the future.

$$\min_{p,q} \sum_{(u,i)\in\kappa} w_{ui} \left((r_{ui} - q_i^T p_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2) \right) \tag{2.6}$$

Table 2.1: Three different approaches to select w_{ui}. m denotes the number of users [29].

	Pos examples	"Neg" examples
Uniform	$w_{ui} = 1$	$w_{ui} = \delta$
User-oriented	$w_{ui} = 1$	$w_{ui} \propto \sum_i R_{ui}$
Item-oriented	$w_{ui} = 1$	$w_{ui} \propto m - \sum_u R_{ui}$

2.3 Demographic filtering

Demographic filtering makes recommendations based on demographic similarities between users. It assumes that users with similar personal features are likely to have common preferences [37]. Demographic filtering is similar to collaborative filtering since they both form "people-to-people" correlations. However, ordinary collaborative filtering techniques suffer from the cold-start problem when there is a new user introduced to the system. This is not the case in demographic filtering, since a new user can be assigned a category and then be recommended courses that are popular among the users within that category [34]. Since Grade has many attributes describing the users and because those attributes are relevant to base the recommendations on, demographic filtering was part of the algorithm designed in this master thesis.

The first step in demographic filtering is to create user profiles. To be able to compare two users to each other and for the computer to be able to handle the input data, it has to be encoded. To avoid personal features being weighted differently, we want to work with binary encoding [12]. This is done by using one-hot-encoding. If user 1 has role B and C, user 2 has role A and user 3 has role B and C, this can be represented as user 1: (0, 1, 1), user 2: (1, 0, 0), and user 3: (0, 1, 1), as shown in figure 2.2. Note that a user can have more than one role.

Table 2.2: Personal features represented by one-hot-encoding.

User	Role A	Role B	Role C
1	0	1	1
2	1	0	0
3	0	1	1

When the user profiles are ready, the users are classified according to their demographic attributes. Demographic filtering creates a number of categories, and all users assigned to a specific category will have similar demographic characteristics. When giving recommendations to a user, the system will first determine which category the user falls in, and then the recommendations will be based on the preferences of the users within that category [34].

2.3.1 Clustering techniques

In demographic filtering, clustering techniques such as k-means clustering can be used to categorize the users according to their personal attributes [16]. The idea of clustering is to divide a set of data points into a number of groups, so that the data points in one group are similar, while they are dissimilar to the data points in other groups. Clustering is an unsupervised learning technique, which makes it possible to find patterns in a data set, without knowing the input data's labels [31]. One of the most common clustering techniques is k-means. K cluster centroids are initialized by taking k random points from the data set. For every data point in the full set, the squared distance to each cluster centroid is calculated and the data point is then assigned to the cluster with closest centroid [11]. New cluster centroids are calculated by taking the average of the data points in each cluster. After this, the

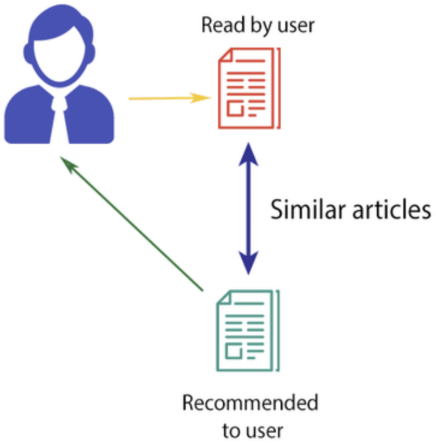

Figure 2.2: Content based filtering [28].

squared distances to each cluster are calculated once again. This procedure is repeated until the cluster centroids do not move significantly anymore [11].

2.4 Content based filtering

Content based filtering is another recommendation algorithm that takes similarities between items into account when recommending an item to a user. If the user has previously shown interest in a certain item, through a positive rating or action, an item similar to the previous item will be recommended, see figure 2.2. The first step is to create a feature matrix. The feature matrix should include all items and their related features such as category, length, creator, etc. The user should be represented in the same feature space. The user-related features can be both explicit and implicit. An explicit feature could be that the user has specified that he or she prefers a specific category in his or her profile, while an implicit feature could be that the user has only bought items of a certain color in the past. When the feature matrix is ready, a similarity metric has to be selected. The similarity metric simply describes how the similarity score should be measured, e.g. using the dot product. The similarity scores are then calculated based on the user, and each item. The item with the highest score will be recommended to the user. Content based filtering does not take other users into account, it only considers other items [13]. Content based filtering was not used in this master thesis, mainly because of the time limitations. This technique will however be suggested as future work.

2.5 Non transitive recommender systems

A problem with state-of-the-art recommender systems, such as collaborative filtering, is that they usually assume that ratings, users, and products are independent and identically distributed (IID). This may result in weak performance since low-level information is not taken into account. In other words, this kind of recommender system may simplify the funda-

mental driving force of users' ratings too much. Non-IID systems involve couplings between user pairs, item pairs, and user-item pairs, leading to heterogeneous recommendations. This will probably make up the next generation's recommender systems, but it comes with big challenges, especially when dealing with large data sets.

Collaborative filtering assumes that both users and items are IID and by combining it with content based filtering and/or demographic filtering, at least some more low-level information is taken into account. However, recommender systems that are not IID can be improved further, which you can read more about in [7]. Grade's users may have functional competencies, scored from one to five. One way to look at it is to consider users with the same functional competencies similar, ignoring the scores, resulting in an IID system. However, it might not be favorable to give users with high scores on a certain functional competence, recommendations based on users with low scores on that functional competence. In this case, it might be better to give recommendations in the opposite direction. This is why it might be a good idea to consider non-IID systems in the context of this master thesis.

2.6 Hybrids

The previously introduced methods have different advantages and choosing only one of the described filtering methods, may result in weak recommender systems. It is common to combine different techniques, to achieve better performance and avoid the cold-start problem. The idea is that the hybrid will benefit from the advantages of each separate method, while it will overcome the methods' weaknesses. There are different techniques to combine the methods, switching and weighted are two common approaches.

The switching approach selects the recommendation source that is most suitable for the specific situation. For example, if a new item is introduced to the system, content based filtering can be used, if there is a new user demographic filtering can be used, and in all other scenarios, collaborative filtering can be used. The weighted approach on the other hand, uses the formula shown in equation 2.7, where r_{CF} is the recommendation given by collaborative filtering, r_{DF} the recommendation given by demographic filtering, r_{CBF} the recommendation given by content based filtering and α, β and γ are the different weights [18]. Some [1, 32, 39] suggest that the filtering, such as collaborative filtering, can be performed once for each demographic category. This was the technique used in this master thesis.

$$r = \alpha r_{CF} + \beta r_{DF} + \gamma r_{CBF} \tag{2.7}$$

2.7 Evaluation of the recommender system

Determining whether a recommender system gives more accurate recommendations than others, might be challenging. The accuracy may differ depending on which evaluation technique is used and there are advantages and disadvantages with all of them. In this section, the three main types of evaluation techniques will be explained. Furthermore, different ways to measure the performance of a recommender system will be described.

2.7.1 Main Evaluations techniques

According to [3], evaluation methods for recommender systems can be divided into three groups: Online evaluation, offline evaluation, and user studies. In online evaluations, recommendations are presented to the users, and depending on the users' actions, the accuracy can be computed. For example, if a user gets a course recommendation and then finishes the course, the recommendation can be considered satisfactory. The so-called click-through rate (CTR) is a common measurement for this. For instance, if a user gets 100 recommendations, and accepts 20 of them, the CTR is 20%. To compare the performance of two algorithms, the CTRs of the two algorithms are compared (called A/B test). Note that this evaluation technique implicitly measures the satisfaction of the user. Also, note that it requires the system to have been launched. This was not possible in this master thesis, why offline evaluation was used instead. Offline evaluation is based on an offline data set, with some data being removed. The performance is then based on the system's ability to recommend the information that is missing. Furthermore, there are three types of offline data sets: true offline data sets, user offline data sets, and expert offline data sets. The difference between true offline data sets and user offline data sets, is that true offline data sets consist of explicit feedback (e.g. ratings), while user offline data sets consist of implicit feedback (e.g. clicks). In expert offline data sets, on the other hand, items have been classified by humans. In the third evaluation technique for recommender systems, user studies, users give explicit ratings on how they experienced the recommendations overall. This means that the participants of the study are asked to quantify their satisfaction with the recommendations generated. It is important to remember that this evaluation technique measures satisfaction at the time of the recommendation. At this time, there is no guarantee that the user itself knows if the recommendations are relevant or not [3].

2.7.2 Metrics

There are many different ways to evaluate a machine learning algorithm when using offline evaluation. One metric can give a high score while another kind of metric can give a very low score on the exact same machine learning model, using the same test data [25]. Some commonly used evaluation metrics are accuracy, precision, recall, and f1 score, where all metrics are based on the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN):

- **TP**: Positives, predicted as positives
- **TN**: Negatives, predicted as negatives
- **FP**: Negatives, predicted as positives
- **FN**: Positives, predicted as negatives

The definitions of the metrics based on TP, TN, FP and FN, are shown in equations 2.8, 2.9, 2.10 and 2.11 [15].

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.8}$$

$$Precision = \frac{TP}{TP + FP} \tag{2.9}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.10}$$

$$f1 = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{2.11}$$

Not all of these metrics are convenient in the recommender system context since the data sets are usually sparse. [10] suggests two different kinds of metrics when evaluating a recommender system based on one-class collaborative filtering: recall and NDCG. This is the reason why those metrics were used to evaluate the recommender system designed in this master thesis. NDCG stands for Normalized Discounted Cumulative Gain and is a measure of ranking quality. The definition of DCG is shown in equation 2.12. Low ranking, means a smaller contribution to DCG, why it is called *Discounted* Cumulative Gain. The definition of NDCG is shown in equation 2.13, where DCG is the DCG of the recommended order, while iDCG is the DCG of the ideal order, that is with the ratings in decreasing order [9].

$$DCG = \sum_{i=1}^{n} \frac{relevance_i}{log_2(i + 1)} \tag{2.12}$$

$$NDCG = \frac{DCG}{iDCG} \tag{2.13}$$

The metrics described above do not take into account how surprising the recommendations are. A good recommender system should not only recommend relevant items but also give recommendations that are surprising since users will find non-surprising recommendations on their own [8]. This is often referred to as serendipity. In the literature, there is no consensus on the definition of serendipity in recommender systems. However, [20] gives an overview of different definitions. The article states that serendipity is a concept that is based on other concepts such as novelty, relevance, and unexpectedness. [38] suggests novelty to measure how surprising recommendations are and defines it as how different the recommendations are compared to what has previously been seen, see equation 2.14. Novelty might be a good complement to metrics such as recall and NDCG, but it does not take into account how relevant the recommendations are. Definitions on serendipity, which also cover relevance, can be found in [38].

$$Novelty_i = 1 - \frac{\text{\# users recommended i}}{\text{\# users}} \tag{2.14}$$

2.7.3 Cross-validation

If the model is only trained and tested once, the evaluation scores might be misleading. This is due to the risk that the split introduced a skew into the data so that the evaluation scores get either very high or very low. To minimize this kind of validation error, cross-validation can be used. A popular method for this is k-fold cross-validation. The full data set is shuffled and then split into k folds. k-1 of the folds are used for training the model, while the last fold, not used in training, is used for evaluation. This is done over k iterations, where the train

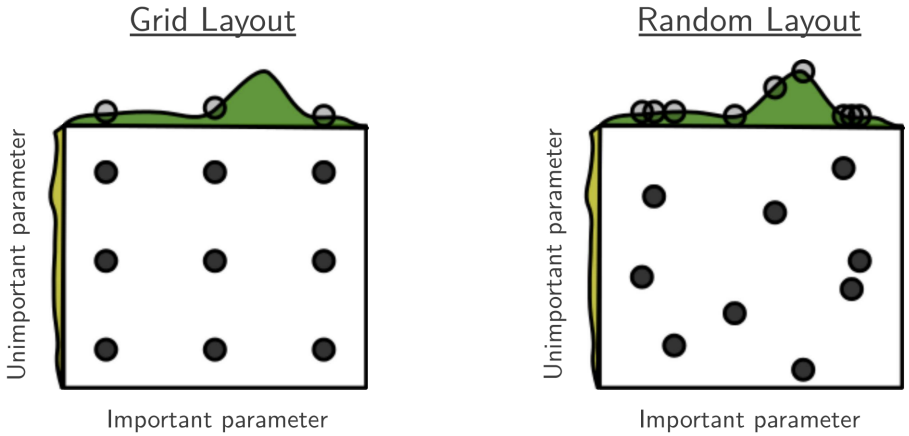

Figure 2.3: The green distribution (of the model's scores) above each square corresponds to g(x), while the yellow distribution on the left of each square corresponds to h(y). For grid search, nine combinations of hyperparameters only test g(x) on three different distinct values, while for random search nine combinations test g(x) on nine different distinct values [5].

and test sets are changed in each iteration. After all iterations, the average evaluation score is calculated. Common values of k are 5 and 10 and the folds are usually of equal size [26].

2.7.4 Hyperparameter optimization

In the machine learning field, hyperparameters are configurations that are external to the model, e.g. learning rate of the SGD algorithm and the regularization term of the function to be minimized. It is not possible to know exactly which hyperparameters are optimal in advance. However one can use rules of thumb, values found in similar machine learning problems, or search for the optimal values by trial and error. Nowadays there are many kinds of optimization algorithms, but two simple ones are grid search and random search. Let us start with grid search. Assume that a model has three different hyperparameters to be optimized: h_1, h_2 and h_3. The first step is to define a range of possible values of each hyperparameter. After this, a grid is constructed with all possible combinations of hyperparameters. The model is trained for each combination and the evaluation score is calculated. The combination of hyperparameters that gives the highest evaluation score, is considered optimal. Grid search is very computationally expensive since the model has to be trained for all different combinations of hyperparameters. To deal with this, another optimization technique is introduced: random search. Instead of providing a discrete set of possible values, a statistical distribution for each hyperparameter is provided, from where values can be randomly sampled. Hyperparameters are usually not equally important, which is something that random search may catch [30]. Figure 2.3, schematically shows the benefits of random search over grid search. Despite this, grid search was used in this master thesis. The reason was mainly the method's simplicity. However, other search algorithms are suggested as future work.

Chapter 3
Method

In this chapter, the method of this master thesis will be described. The method will be based on the previously presented theory, along with input from one of Grade's customers.

3.1 Data gathering

The first step of the method was to gather data, on which the model was to be trained on. In the two following sections, input from one of Grade's customers will be summarized, and statistics on the data found in the databases of three of Grade's customers will be presented.

3.1.1 Interview with customer

One of Grade's customers (customer A) was interviewed to determine what they expect from a recommender system, and which user data is more relevant to base the algorithm on. Unit belonging was something they thought would be suitable for recommendations. In their opinion, the existing units are narrow enough to assume that all members in a unit have similar interests when it comes to courses. When different roles were discussed, they mentioned that many roles are very general, and therefore there is a risk with basing recommendations on roles. However, competencies that are usually related to roles may give a good hint on which courses are relevant for a user. Furthermore, it might be a good idea to look at how a user is graded in each functional competence. Users with the same competence should be considered similar and furthermore, users with lower grades should receive recommendations from users with higher grades on that competence. During the interview, the cold-start problem was also discussed. How should we deal with new courses that no users have discovered yet? Administrators can make courses mandatory for specific groups, but this is not done to a very great extent as it is today. Customer A thought that it would be convenient if there was some tool in the platform presenting new and relevant courses. They also sug-

gested content based filtering as a solution to the cold-start problem, taking advantage of the keywords that each course has.

3.1.2 Data set

The data used to train the model was found in Grade's databases. Because of time limitations, only three customers were chosen. Some statistics on each data set (customer A, B, and C) can be seen in table 3.1. Note that the data used for training in this master thesis, was collected only once, but yet the system is integrated into Grade's platform, the data will be collected once per training, which will occur regularly.

Table 3.1: Statistics on data for customer A, B, and C.

Category	Customer A	Customer B	Customer C
users	49,123	22,476	108,135
courses	2,426	1,025	671
user-course interactions	226,945	175,089	78,590
units	1,812	1,066	645
roles	20	158	0
competences	189	312	0
certificates	172	21	4

3.1.3 Train and Test split

To get trustworthy evaluation scores, one has to evaluate the model using a separate data set than the one used during training. For this reason, the full data set was split into a train set (80 %) and a test set (20 %). Furthermore, when evaluating the system for different choices of hyperparameter settings, the train set (80 % of the full set) was divided into five folds and cross-validation was performed, as described in section 2.7.3. The final model was then evaluated using the test set (20 % of the full set). Note that the splitting was only performed on the data used for matrix factorization (user-course interactions), not on the data used for clustering. The reason for this was that the clustering was unsupervised, meaning that there were no known labels corresponding to the clusters. Evaluation could therefore not be performed only on the clustering part of the algorithm.

3.2 Model selection

The recommender system of this project was a hybrid between Collaborative filtering, using matrix factorization, and Demographic filtering, using k-means clustering. The demographic categories used in this master thesis were the users' unit, roles, competencies, and certificates. Note that roles were used, even though customer A suggested not to. The reason for this was that Grade thought that roles are actually important to use, at least for most of their

customers. Demographic filtering was chosen over Content based filtering since there were more data describing the users than data describing the courses. The fact that Content based filtering was not used in this project may result in the cold-start problem when new courses are introduced to the system. This will be discussed later, when considering the weaknesses of the final model, and adding content based filtering to the hybrid will be suggested as a possible improvement.

Microsoft's .NET framework is the framework used in Grade's existing platform, why the library ML.NET was used both for matrix factorization and for k-means clustering. Using .NET instead of for example TensorFlow will probably make it easier for Grade to integrate and maintain the recommender system, which is of utmost importance from their perspective.

As earlier mentioned, matrix factorization is a widely used and efficient method for Collaborative filtering, why this was used in this project. The clustering algorithm used was k-means, since it is a widely used technique, and since it is easy to implement using ML.NET. Each cluster had to be large enough to give each user in that cluster five recommendations. This means that the number of unique courses finished by each cluster had to be at least five more than the number of courses finished by the user who had finished the biggest amount of courses in that specific cluster. The number of unique courses finished by the cluster also had to be big enough even when courses that do not have any active sessions, were removed from the data set. In case one cluster was too small, each user in that cluster was assigned to the cluster with the second closest centroid. This means that if k (the number of clusters) was originally set to 25, the size of k might have become lower than 25 during training, due to too small clusters.

The number of latent factors (approximationRank) used in matrix factorization and the number of clusters (k) used in demographic filtering was changed over different experiments, while the remaining hyperparameters were fixed, see table 3.2. Note that LossFunction is the loss function to be minimized in the stochastic gradient descent algorithm, Alpha is the importance of unobserved elements, C is the value of the unobserved elements, lambda is the regularization parameter, and NumberOfIterations is the number of iterations where the loss function is minimized. Furthermore, OptimizationTolerance denotes when the clustering algorithm should stop, assuming that the number does not exceed MaximumNumberOfIterations. Also note that Alpha was chosen according to the suggestion in [14], see equation 3.1.

$$Alpha = \frac{\#\ of\ observed\ entries}{\#\ of\ unobserved\ entries} \tag{3.1}$$

3.2.1 Baseline algorithm

To get an idea of the performance of the recommender system, the evaluation scores of a baseline algorithm were calculated, in addition to the evaluation scores of the model. The baseline used in this project was to always recommend the five courses with the largest number of attendees. The evaluation scores recommending five random courses and the five courses with the smallest number of attendees were also calculated but will not be included in this report. An alternative baseline would be to use Grade's existing prototype of recommender system, but due to shortage of time, a simpler baseline was used instead.

Table 3.2: The default values of hyperparameters in matrix factorization and k-means clustering respectively.

	Parameter	Default value
Matrix factorization	approximationRank	*Varied*
	LossFunction	SquareLossOneClass
	Alpha	See equation 3.1
	C	0.15
	Lambda	0.1
	NumberOfIterations	100
k-means	K	*Varied*
	OptimizationTolerance	1e-30
	MaximumNumberOfIterations	1000

3.2.2 Evaluation

The metric used to evaluate the model was NDCG and recall, as suggested in [10]. During training, all user-course pairs got a score: close to zero if the user was unlikely to be interested in the course, close to one if the user was likely to be interested in the course. To retrieve the top five recommendations for a user, the courses with the highest scores were selected. However, for calculation of NDCG, the exact ranking was also of interest. Assume that the model predicted five courses c1, c2, c3, c4, and c5 for a user, with c1 having the highest score. To calculate NDCG, the actual scores of c1-c5 found in the test set were checked and those scores (in order c1-c5) were used to calculate DCG. After this, the list of actual scores was ordered with the highest score first, that is the ideal order, and DCG was then calculated (iDCG). NDCG was then calculated by dividing DCG with iDCG, according to equation 2.13. NDCG was used for hyperparameters optimization, but recall was also calculated and presented in Appendix A and B.

During the evaluation of recall, each user-course pair was labeled 1 if the course was included in that user's top five recommendations, otherwise labeled 0. These labels were stored in a list called predictions, which was then compared to a list called expected. This list only consisted of ones, since all user-course pairs in the test set were observed elements, meaning that the user had finished the course.

As a complement to automatic evaluation, a user study was performed (referred to as user studies in section 2.7.1). One of Grade's customers (customer A) chose eight of their users and the recommendations of those eight users, generated by the recommender system, were then presented to the customer. The customer gave feedback on the generated recommendations and the system was then revised according to their input. Note that in order to make complete user studies, more than eight users' recommendations should be analyzed. However, this was not possible because of the time limitations of this project.

Chapter 4

Results

In this chapter, the results of this master thesis will be presented. The chapter is divided into three parts: Matrix factorization, Demographic matrix factorization, and a summary of the results. Each of the sections 4.1 and 4.2 will be divided into three further sections: results from customer A, B and C. Note that the number of courses recommended was always five during training and evaluation.

4.1 Matrix Factorization

In this section figures showing how $NDCG$ varies for different values of $approximationRank$ (the number of latent features) when using MF, will be presented. This will be compared to a baseline where the users were always recommended the five most popular courses. The figures presented in this section only include the $NDCG$ scores, but values of $recall$ and $novelty$ can be found in appendix A. In general, the $NDCG$ scores were similar to the $recall$ scores, see table A.1, A.2, and A.3.

4.1.1 Customer A

Figure 4.1 shows how $NDCG$ varies over different values of $approximationRank$, when training a MF model on customer A's data. The lowest $NDCG$ retrieved from cross-validation was $NDCG = 0.1881$ ($approximationRank = 4$), while the highest $NDCG$ retrieved was $NDCG = 0.2909$ ($approximationRank = 38$). When training the MF model with $approximationRank = 38$ on the training set and evaluating it with the test set, the $NDCG$ retrieved was $NDCG = 0.3375$. All scores from the final evaluation, including $NDCG$, $recall$ and $novelty$ for the MF model and baseline, can be seen in table 4.4.

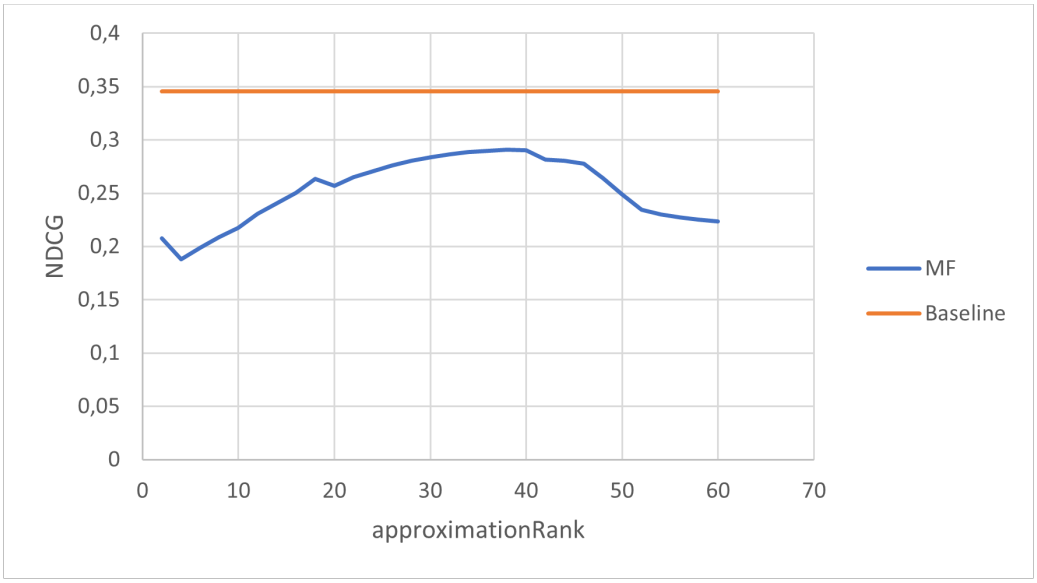

Figure 4.1: $NDCG$ for different values of approximationRank, model: MF, customer: A. The best result $NDCG = 0.2909$ was retrieved when using $approximationRank = 38$.

4.1.2 Customer B

Figure 4.2 shows how $NDCG$ varies over different values of $approximationRank$, when training a MF model on customer B's data. The lowest $NDCG$ retrieved from cross-validation was $NDCG = 0.1656$ ($approximationRank = 2$), while the highest $NDCG$ retrieved was $NDCG = 0.3254$ ($approximationRank = 32$). When training the MF model with $approximationRank = 32$ on the training set and evaluating it with the test set, the $NDCG$ retrieved was $NDCG = 0.3471$. All scores from the final evaluation, including $NDCG$, *recall* and *novelty* for the MF model and baseline, can be seen in table 4.4.

4.1.3 Customer C

Figure 4.3 shows how $NDCG$ varies over different values of $approximationRank$, when training a MF model on customer C's data. The lowest $NDCG$ retrieved from cross-validation was $NDCG = 0.4560$ ($approximationRank = 60$), while the highest $NDCG$ retrieved was $NDCG = 0.6150$ ($approximationRank = 14$). When training the MF model with $approximationRank = 14$ on the training set and evaluating it with the test set, the $NDCG$ retrieved was $NDCG = 0.6031$. All scores from the final evaluation, including $NDCG$, *recall* and *novelty* for the MF model and baseline, can be seen in table 4.4.

4.2 Demographic Matrix Factorization

In this section tables showing how $NDCG$ varies for different values of k and $approximationRank$ when using DMF, will be presented for each of customer A, B, and C. Additionally, the section with customer A's results will include a figure showing how $NDCG$

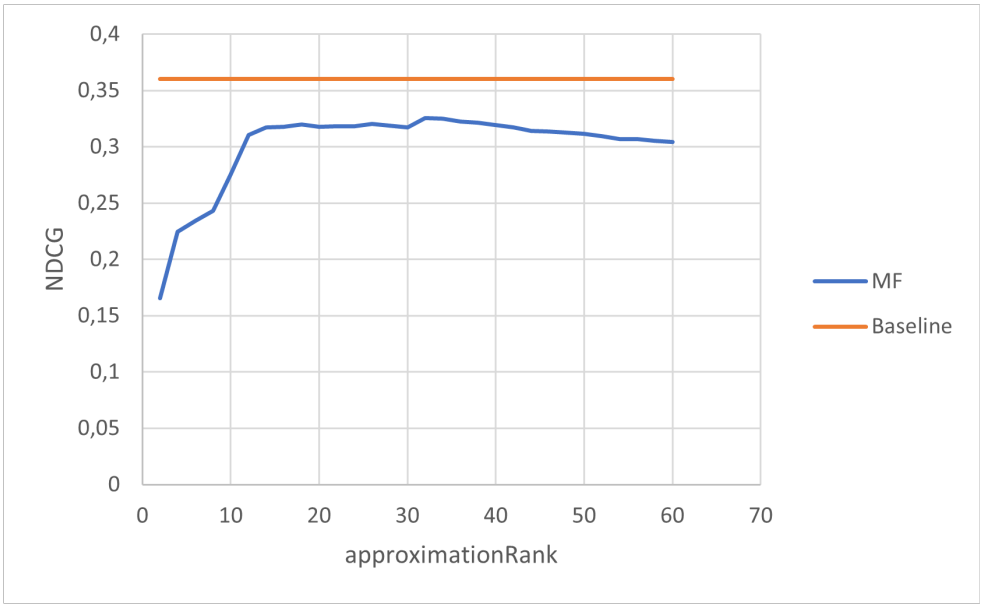

Figure 4.2: *NDCG* for different values of *approximationRank*, model: MF, customer: B. The best result $NDCG = 0.3254$ was retrieved when using $approximationRank = 32$.

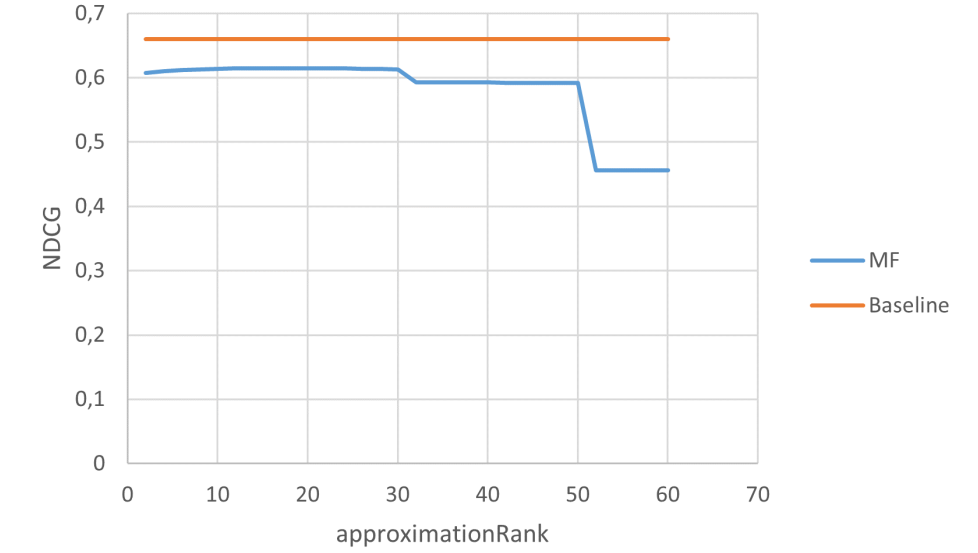

Figure 4.3: *NDCG* for different values of *approximationRank*, model: MF, customer: C. The best result $NDCG = 0.6031$ was retrieved when using $approximationRank = 14$.

varies for different values of *approximationRank* when keeping k constant, and a figure showing how *NDCG* varies for different values of k when keeping *approximationRank* constant. The results will be compared to a baseline where the users were always recommended the five most popular courses. The tables and figures presented in this section only include the *NDCG* scores, but values of *recall* and *novelty* can be found in appendix B. In general, the *NDCG* scores were similar to the *recall* scores, see table B.1, B.2, and B.3.

4.2.1 Customer A

Table 4.1 shows how *NDCG* varies over different values of k and *approximationRank*, when training a DMF model on customer A's data. The lowest *NDCG* retrieved from cross-validation was $NDCG = 0.2454$ ($k = 10$, $approximationRank = 30$), while the highest *NDCG* retrieved was $NDCG = 0.3215$ ($k = 30$, $approximationRank = 5$).

Table 4.1: *NDCG* for different combinations of *approximationRank* and k, model: DMF, customer: A. The highest *NDCG* written in bold. Remaining hyperparameters set to default (see table 3.2).

		approximationRank					
		5	**10**	**15**	**20**	**25**	**30**
k	**5**	0.2666	0.2865	0.2753	0.286	0.2872	0.2648
	10	0.2774	0.2459	0.3029	0.3103	0.3096	0.2454
	15	0.2951	0.2884	0.2876	0.298	0.299	0.2655
	20	0.2849	0.3021	0.3127	0.321	0.2778	0.2669
	25	0.3083	0.2922	0.2718	0.3056	0.2763	0.2782
	30	**0.3215**	0.2828	0.3053	0.3155	0.2977	0.2934

Figure 4.4 shows how *NDCG* varies for different values of *approximationRank*, when $k = 30$, using DMF on customer A's data. The highest *NDCG* retrieved from cross-validation was $NDCG = 0.3322$ ($k = 30$, $approximationRank = 2$).

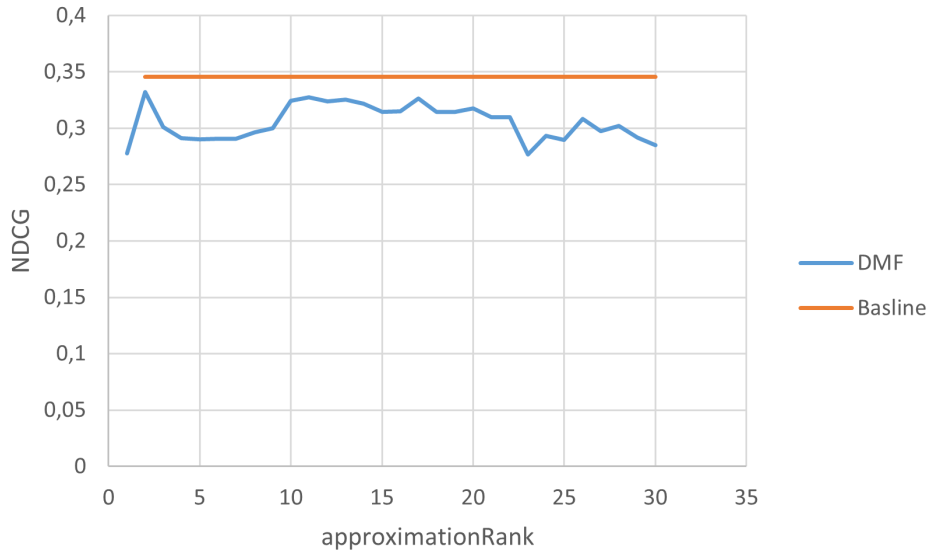

Figure 4.4: $NDCG$ for different values of $approximationRank$, model: DMF, customer: A, $k = 30$. Maximum in $approximationRank = 2$ ($NDCG = 0.3322$).

Figure 4.5 shows how $NDCG$ varies for different values of k, when $approximationRank = 5$, using DMF on customer A's data. The highest $NDCG$ retrieved from cross-validation was $NDCG = 0.3196$ ($k = 39$, $approximationRank = 5$).

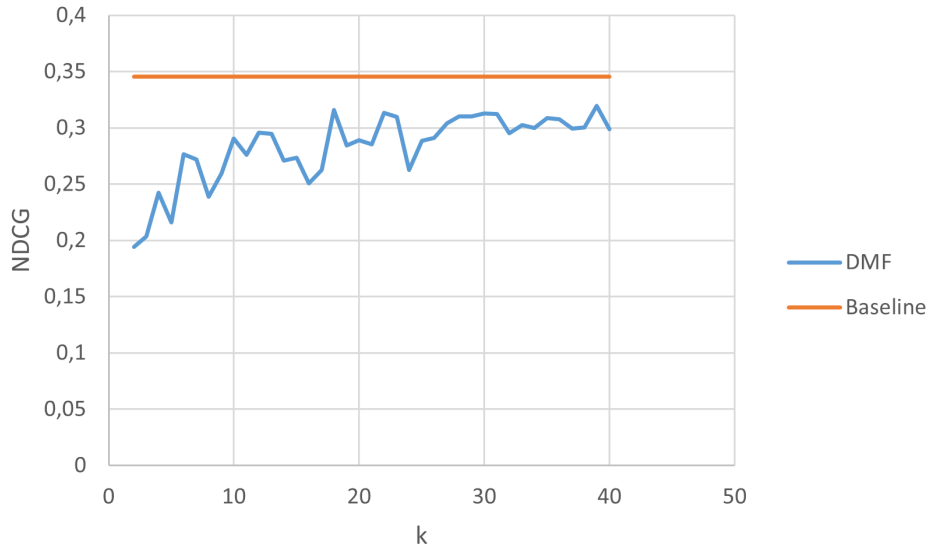

Figure 4.5: $NDCG$ for different values of k, model: DMF, customer: A, $approximationRank = 5$. Maximum in $k = 39$ ($NDCG = 0.3196$).

When training the DMF model using the optimal hyper parameters $k = 30$ and $approximationRank = 2$ (retreived from figure 4.4) on the training set and evaluating it

with the test set, the *NDCG* retrieved was $NDCG = 0.3721$. All scores, including *NDCG*, *recall* and *novelty* for the DMF model and baseline, can be seen in table 4.5.

4.2.2 Customer B

Table 4.2 shows how *NDCG* varies over different values of k and *approximationRank*, when training a DMF model on customer B's data. The lowest *NDCG* retrieved from cross-validation was $NDCG = 0.2788$ ($k = 25$, $approximationRank = 30$), while the highest *NDCG* retrieved was $NDCG = 0.3531$ ($k = 10$, $approximationRank = 20$). When training the DMF model with $k = 10$ and $approximationRank = 20$ on the training set and evaluating it with the test set, the *NDCG* retrieved was $NDCG = 0.3816$. All scores, including *NDCG*, *recall* and *novelty* for the DMF model and baseline, can be seen in table 4.5.

Table 4.2: *NDCG* for different combinations of *approximationRank* and k, model: DMF, customer: B. The highest *NDCG* written in bold. Remaining hyperparameters set to default (see table 3.2).

		approximationRank					
		5	**10**	**15**	**20**	**25**	**30**
k	**5**	0.3407	0.3295	0.3246	0.3159	0.3365	0.3294
	10	0.3521	0.3388	0.3159	**0.3531**	0.309	0.2919
	15	0.3339	0.347	0.3495	0.3229	0.312	0.317
	20	0.3476	0.3492	0.3349	0.3196	0.3062	0.2915
	25	0.3454	0.3176	0.3162	0.3198	0.312	0.2788
	30	0.3387	0.3428	0.3116	0.3155	0.3037	0.2831

4.2.3 Customer C

Table 4.3 shows how *NDCG* varies over different values of k and *approximationRank*, when training a DMF model on customer C's data. The lowest *NDCG* retrieved from cross-validation was $NDCG = 0.5117$ ($k = 20$, $approximationRank = 5$), while the highest *NDCG* retrieved was $NDCG = 0.6614$ ($k = 30$, $approximationRank = 20$). When training the DMF model with $k = 30$ and $approximationRank = 20$ on the training set and evaluating it with the test set, the *NDCG* retrieved was $NDCG = 0.6683$. All scores, including *NDCG*, *recall* and *novelty* for the DMF model and baseline, can be seen in table 4.5.

Table 4.3: *NDCG* for different combinations of *approximationRank* and *k*, model: DMF, customer: C. The highest *NDCG* written in bold. Remaining hyperparameters set to default (see table 3.2).

		approximationRank					
		5	**10**	**15**	**20**	**25**	**30**
k	**5**	0.5526	0.5551	0.5567	0.5555	0.5559	0.555
	10	0.5557	0.5554	0.5561	0.5559	0.556	0.5548
	15	0.5145	0.5153	0.5157	0.5153	0.5146	0.5154
	20	0.5117	0.5169	0.5157	0.5172	0.5142	0.5146
	25	0.6605	0.6614	0.6607	0.6614	0.661	0.6589
	30	0.658	0.6603	0.661	**0.6614**	0.6593	0.6586

4.2.4 Summary of results

Table 4.4 shows the results from the final evaluations, using the optimal values of *approximationRank* for each of customer A, B, and C. In addition to the results retrieved when evaluating the MF models, the baseline scores are presented. Furthermore, the difference between the results from MF and Baseline are presented.

Table 4.4: Summary of results, Matrix Factorization (MF).

		Customer A	Customer B	Customer C
MF	approximationRank	38	32	14
	NDCG	0.3375	0.3471	0.6031
	Recall	0.2973	0.2853	0.7176
	Novelty	0.9939	0.9863	0.9677
Baseline	NDCG	0.3724	0.3954	0.6603
	Recall	0.3177	0.3365	0.7836
	Novelty	0	0	0
MF-Baseline	NDCG	-0.0349	-0.0484	-0.0572
	Recall	-0.0203	-0.0512	-0.0659
	Novelty	0.9939	0.9863	0.9677

Table 4.5 shows the results from the final evaluations, using the optimal values of *approximationRank* and *k* for each of customer A, B, and C. In addition to the results retrieved when evaluating the DMF models, the baseline scores are presented. Furthermore, the difference between the results from MF and Baseline are presented, just like in table 4.4

Table 4.5: Summary of results, Demographic Matrix Factorization (DMF).

		Customer A	Customer B	Customer C
DMF	approximationRank	2	20	20
	k	30	10	30
	NDCG	0.3721	0.3816	0.6683
	Recall	0.3392	0.3104	0.787
	Novelty	0.9708	0.9867	0.9766
Baseline	NDCG	0.3454	0.3603	0.6604
	Recall	0.3166	0.3356	0.7836
	Novelty	0	0	0
DMF-Baseline	NDCG	0.0267	0.0213	0.0079
	Recall	0.0226	-0.0252	0.0034
	Novelty	0.9708	0.9867	0.9766

Chapter 5
Discussion

In this chapter, the results from Chapter 4 will be analyzed. Weaknesses of the model and the evaluation technique used, and future work will also be discussed.

5.1 Hyperparameters

Figure 4.1, 4.2 and 4.3 show that $approximationRank$ is indeed a hyperparameter that affects the $NDCG$ scores of the MF models. The optimal value of $approximationRank$ was 38, 32, and 14 for customer A, B, and C. The difference in optimal $approximationRank$ might depend on the amount of data that each of the models was trained on. The model for customer A was trained on $226,945$ user-course pairs, while customer B was trained on $175,089$ pairs, and customer C on $78,590$ pairs (see table 3.1). It seems like a larger training set requires a higher value of $approximationRank$ and vice versa. This is not surprising since the value of $approximationRank$ is the same as the number of latent factors. There are probably more latent factors to be found in a large data set than in a small data set and vice versa.

In figure 4.4, showing how $NDCG$ varies for different values of $approximationRank$ when $k = 30$, the trend is not as obvious as in figure 4.1, where MF was used. The reason for this might be that the value of $approximationRank$ used was the same for all different clusters, no matter the cluster sizes. When looking at figure 4.5, showing how $NDCG$ varies for different values of k when $approximationRank = 5$, there is no obvious trend either. The reason for this might be the same: the $approximationRank$ used was the same for all clusters. As earlier mentioned, the optimal value of $approximationRank$ seems to correlate with the size of the training set. Since the clusters may vary in size, $approximationRank$ should probably vary for different clusters.

When it comes to the hyperparameter k, it is important to compare k with $realK$, where k is the number of clusters that the model aimed to categorize the users into, while $realK$ is the final number of clusters.

In appendix B, in table B.1 and table B.2, one can see that the values of $realK$ for customer

A and B are close to the corresponding values of k. In table B.3 on the other hand, the differences between k and $realK$ are in almost all cases very large. What this means is that in the case of $approximationRank = 20$ and $k = 30$, 15 of the 30 initial clusters contained a too small amount of data, and the users in each of those 15 clusters had to be assigned to another cluster. The reason why $k - realK$ is so much larger for customer C than for customer A and B might be that customer C has many fewer personal features (units, roles, competencies, and certificates), see table 3.1. This makes it inconvenient to try to cluster the users into a large number of clusters since it will only result in reclustering. Focusing on customer A and B, it is hard to correlate the optimal value of k to the data presented in table 3.1. The optimal value of k is apparently not only correlated to the number of personal features, but also to how many users that have a specific personal feature, which is impossible to determine from table 3.1.

5.2 MF vs. DMF

When comparing table 4.4 with table 4.5, showing summaries of the final results of Matrix Factorization (MF) and Demographic Matrix Factorization (DMF), one can see that DMF performed better than MF for all three customers. This is not surprising since the MF models were trained only on user-course interactions, while the DMF models were trained on demographic information in addition to user-course interactions. It could be interesting to see how the $NDCG$ scores would change if some demographic categories were removed or added, which is left as future work.

5.3 Differences between customers

When comparing the results in table 4.5, one can see that the difference between $NDCG(DMF)$ and $NDCG(Baseline)$, was larger for A than for B, and larger for B than for C, meaning that the DMF on customer A performed better than DMF on customer B, etc. The reason for this is probably the amount of data used for training. A larger amount of training data results in a better model.

The results retrieved from DMF on customer C, shown in table 4.3, show that the $NDCG$ scores are very similar to each other. This is due to the big difference between k and $realK$ described in section 5.1. The tables showing the $NDCG$ scores retrieved from grid search are therefore a bit misleading, especially for customer C.

5.4 The evaluation technique

In the figures in the previous chapter, that is figure 4.1, 4.2, 4.3, 4.4, and 4.5, showing hyperparameter optimization using cross-validation, one can see that our models never beat baseline. However, the DMF models beat the baseline in the final evaluation, see table 4.5. The reason why the final evaluations give higher scores than the evaluation during hyperparameter optimization is probably that the models are trained on a larger data set in the final evaluation (80 % of the full data set), compared to when cross-validation is performed. In cross-validation,

the models are only trained on 4/5 of 80 % of the full data set. A larger training set usually results in a more accurate model. It is not surprising that the model *NDCG* and *recall* do not completely out-compete the corresponding baseline scores. If there are five courses that almost all users take (possibly mandatory courses) it will be impossible for our model's *NDCG* and *recall* to reach the corresponding scores of the baseline algorithm. The purpose of a recommender system is to make users aware of the existence of courses that they otherwise would not discover. Ideally, courses that not many users have finished should be recommended if relevant to the target user. This will not be taken into account by *NDCG* and *recall*. This is the reason why *novelty* was introduced, even though it should just be used as a complement to *NDCG* and *recall* since *novelty* itself does not take the relevance of the recommendations into account. The *novelty* scores are very close to 1 in all cases, while the baseline *novelty* is always 0. Another weakness of the evaluation technique is that for users that have finished $n < 5$ courses, $5 - n$ courses will inevitably get incorrect classifications. Furthermore, if a user-course pair is present in the test set, but the course is not present in the train set, this course will not be recommended to the user, because of the cold-start problem. This will result in a lower evaluation score.

To make the evaluation scores more accurate, it would be convenient to implement online evaluation. However, online evaluation was not possible in this master thesis, since that would require the recommender system to be integrated into Grade's platform. For this reason, the evaluation scores were mainly used for optimization of the hyperparameters. To get an idea of how well the system performed, customer A manually checked the recommendations (referred to as user study) of eight users of their choice and the feedback was positive. They thought that the recommendations seemed accurate, with the exception that some of the recommended courses did not have any active sessions. Courses with no active sessions were then removed from the recommendations. In addition to customer A's feedback, the recommendations of those eight users were analyzed more deeply. These users had similar user features and when listing the eight users' finished courses along with their recommended courses it was possible to see that many of the recommendations for one user could be found among the other users' finished courses.

5.5 Future work

The current model handles the cold-start problem when a new user is introduced to the system, but not when a new course is introduced. To handle this, content based filtering may be implemented, as described in section 2.4. In addition to category, length, etc., the descriptions of the courses could be used to find similarities between courses, using Natural Language Processing. To enhance the model further, the implicit feedback used in this recommender system might be supplemented by explicit feedback. This would however require Grade to make their users rate courses to a greater extent. It would also be interesting to investigate if the current use of implicit feedback could be enhanced, by experimenting with different values of C. Another suggestion on how to improve the model is to take users' scores on functional competencies into account. This would make the system non-IID as described in section 2.5, which comes with many challenges. Another suggestion of future work is to investigate if there are better ways to design the hybrid between collaborative and demographic filtering, resulting in higher evaluation scores.

As discussed in section 5.4, the evaluation techniques used in this master thesis have weaknesses. To overcome these, a more advanced definition of Serendipity should be used, involving the relevance of the recommendations. However, offline evaluation is not the best option in this case, and online evaluation should be introduced as soon as the system has been launched.

To improve the optimization of the recommender system, the hyperparameter *approximationRank* should be optimized for each cluster, since there seems to be a correlation between optimal *approximationRank* and cluster size. Furthermore, hyperparameter optimization should be automatized in the future, and there may be more optimal search algorithms than grid search, which was used in this master thesis.

Chapter 6
Conclusions

In this master thesis, a recommender system was developed for the E-learning company Grade AB. Based on user-course interactions, matrix factorization (MF) was used to generate recommendations to the users. In addition, the system was enhanced by also basing the recommendations on several user attributes, referred to as Demographic matrix factorization (DMF). As expected, DMF performed better than MF. Furthermore, the results of this master thesis showed that the larger the training data set, the better the evaluation scores. However, the difference of NDCG between the three clients was quite small, and the conclusion is that for customers with similar size to customer A, B, and C, it is possible to use the recommender system developed in this master thesis.

The biggest challenge of this project was to find a proper method to evaluate the recommender system. At first sight, it was surprising that the models developed in this master thesis only barely beat baseline, always recommending the most popular courses. However, this could be explained by the weaknesses of the evaluation technique. The goal of a recommender system is to give surprising, but relevant recommendations. Therefore, always recommending the most popular courses to users will not be completely satisfactory. The novelty score was introduced to highlight that NDCG and recall are not enough when evaluating a system. As a complement, user studies were used, to get a better idea of the performance of the recommender systems. The user studies gave surprisingly good results, but it is important to remember that only eight users were asked. To achieve more trustworthy evaluation scores in the future, online evaluation should be introduced, using for example CTR.

The goal of this project was to develop a recommender system that could generate relevant recommendations, avoiding all administrative work that was needed in Grade's initial system. This was achieved by this project, even though there are possibilities to enhance the performance further, as described in section 5.5.

References

[1] Web-based personalized hybrid book recommendation system. *2014 International Conference on Advances in Engineering Technology Research (ICAETR - 2014), Advances in Engineering and Technology Research (ICAETR), 2014 International Conference on*, pages 1 – 5, 2014.

[2] Zahra Ahmad. Recommender systems: Explicit feedback, implicit feedback and hybrid feedback, 2021.

[3] Joeran Beel, Marcel Genzmehr, Stefan Langer, Andreas Nürnberger, and Bela Gipp. A comparative analysis of offline and online evaluations and discussion of research paper recommender system evaluation. In *Proceedings of the international workshop on reproducibility and replication in recommender systems evaluation*, pages 7–14, 2013.

[4] Robert M Bell and Yehuda Koren. Lessons from the netflix prize challenge. *Acm Sigkdd Explorations Newsletter*, 9(2):75–79, 2007.

[5] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:284, 2012.

[6] Jesús Bobadilla, Fernando Ortega, Antonio Hernando, and Jesús Bernal. A collaborative filtering approach to mitigate the new user cold start problem. *Knowledge-Based Systems*, 26:225 – 238, 2012.

[7] Longbing Cao. Non-iid recommender systems: A review and framework of recommendation paradigm shifting. *Engineering*, 2(2):212–224, 2016.

[8] Òscar Celma and Perfecto Herrera. A new approach to evaluating novel recommendations. *Proceedings of the 2008 ACM Conference: Recommender Systems*, pages 179 – 186, 2008.

[9] Pranay Chandekar. Evaluate your recommendation engine using ndcg. https://towardsdatascience.com/evaluate-your-recommendation-engine-using-ndcg-759a851452d1, 2021. Visited on 2021-11-25.

[10] Jin Chen, Defu Lian, and Kai Zheng. Improving one-class collaborative filtering via ranking-based implicit regularizer. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:37–44, 2019.

[11] Imad Dabbura. K-means clustering: Algorithm, applications, evaluation methods, and drawbacks. https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a. Visited on 2021-11-05.

[12] Michael DelSole. What is one hot encoding and how to do it. https://medium.com/@michaeldelsole/what-is-one-hot-encoding-and-how-to-do-it-f0ae272f1179, 2021. Visited on 2021-10-28.

[13] Google Developers. Content-based filtering. https://developers.google.com/machine-learning/recommendation/content-based/basics, 2021. Visited on 2021-09-05.

[14] Microsoft Docs. matrixfactorizationtrainer.options.alpha field. https://docs.microsoft.com/en-us/dotnet/api/microsoft.ml.trainers.matrixfactorizationtrainer.options.alpha?view=ml-dotnet-preview. Visited on 2021-10-15.

[15] Shweta Goyal. Evaluation metrics for classification models. https://medium.com/analytics-vidhya/evaluation-metrics-for-classification-models-e2f0d8009d69, 2021. Visited on 2021-10-15.

[16] J. Gupta and J. Gadge. Performance analysis of recommendation system based on collaborative filtering and demographics. *2015 International Conference on Communication, Information Computing Technology (ICCICT), Communication, Information Computing Technology (ICCICT), 2015 International Conference on*, pages 1 – 6, 2015.

[17] Gawesh Jawaheer, Peter Weller, and Patty Kostkova. Modeling user preferences in recommender systems : A classification framework for explicit and implicit user feedback. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 4(2):1 – 26, 2014.

[18] Mohamed Elyes Ben Haj Kbaier, Hela Masri, and Saoussen Krichen. A personalized hybrid tourism recommender system. *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA), Computer Systems and Applications (AICCSA), 2017 IEEE/ACS 14th International Conference on, AICCSA*, pages 244 – 250, 2017.

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

[20] Denis Kotkov, Shuaiqiang Wang, and Jari Veijalainen. A survey of serendipity in recommender systems. *Knowledge-Based Systems*, 111, 08 2016.

[21] Siping Liu, Xiaohan Tu, and Renfa Li. Unifying explicit and implicit feedback for top-n recommendation. *2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)(, Big Data Analysis (ICBDA), 2017 IEEE 2nd International Conference on*, pages 35 – 39, 2017.

[22] Shuyu Luo. Intro to recommender system: Collaborative filtering. https://towardsdatascience.com/intro-to-recommender-system-collaborative-filtering-64a238194a26, 2021. Visited on 2021-09-20.

[23] Xin Luo, Dexian Wang, MengChu Zhou, and Huaqiang Yuan. Latent factor-based recommenders relying on extended stochastic gradient descent algorithms. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 51(2):916–926, 2021.

[24] Prem Melville and Vikas Sindhwani. Recommender systems. *Encyclopedia of machine learning*, 1:829–838, 2010.

[25] Aditya Mishra. Metrics to evaluate your machine learning algorithm. https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234, 2018. Visited on 2021-09-30.

[26] Caleb Neale. Cross validation: A beginner's guide. https://towardsdatascience.com/cross-validation-a-beginners-guide-5b8ca04962cd, 2021. Visited on 2021-10-20.

[27] Netflix. Netflix prize data. https://www.kaggle.com/netflix-inc/netflix-prize-data, 2021. Visited on 2021-12-15.

[28] Lionel Ngoupeyou Tondji. Web recommender system for job seeking and recruiting. https://www.researchgate.net/figure/Content-based-filtering-vs-Collaborative-filtering-Source_fig5_323726564, 02 2018.

[29] Rong Pan, Yunhong Zhou, Bin Cao, Nathan N. Liu, Rajan Lukose, Martin Scholz, and Qiang Yang. *One-Class Collaborative Filtering*. 2008.

[30] Sayak Paul. Hyperparameter optimization in machine learning models. https://www.datacamp.com/community/tutorials/parameter-optimization-machine-learning-models, 2018. Visited on 2021-11-19.

[31] Surya Priy. Clustering in machine learning - geeksforgeeks. https://www.geeksforgeeks.org/clustering-in-machine-learning/, 2021. Visited on 2021-11-05.

[32] Shini Renjith and C Anjali. A personalized mobile travel recommender system using hybrid algorithm. *2014 First International Conference on Computational Systems and Communications (ICCSC), Computational Systems and Communications (ICCSC), 2014 First International Conference on*, pages 12 – 17, 2014.

[33] Baptiste Rocca. Introduction to recommender systems. https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada, 2021. Visited on 2021-09-28.

[34] Iateilang Ryngksai and L Chameikho. Recommender systems: Types of filtering techniques. *International Journal of Engineering Research Technology*, 3(11), 2014.

[35] J Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. Collaborative filtering recommender systems. In *The adaptive web*, pages 291–324. Springer, 2007.

[36] S. Surekha. A personalized recommendation system using memory-based collaborative filtering algorithm. In *2018 International Conference on Current Trends towards Converging Technologies (ICCTCT)*, pages 1–6, 2018.

[37] Dang Thai Thinh, Duong Trang Hai, and Nguyen Hong Son. A hybrid framework for enhancing correlation to solve cold-start problem in recommender systems. *the 2014 Seventh IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA), Computational Intelligence for Security and Defense Applications (CISDA), 2014 Seventh IEEE Symposium on*, pages 1 – 5, 2014.

[38] Saúl Vargas and Pablo Castells. Rank and relevance in novelty and diversity metrics for recommender systems. *Proceedings of the Fifth ACM Conference: Recommender Systems*, pages 109 – 116, 2011.

[39] AFOUDI Yassine, LAZAAR Mohamed, and Mohammed Al Achhab. Intelligent recommender system based on unsupervised machine learning and demographic attributes. *Simulation Modelling Practice and Theory*, 107, 2021.

Appendix A
Additional results, MF

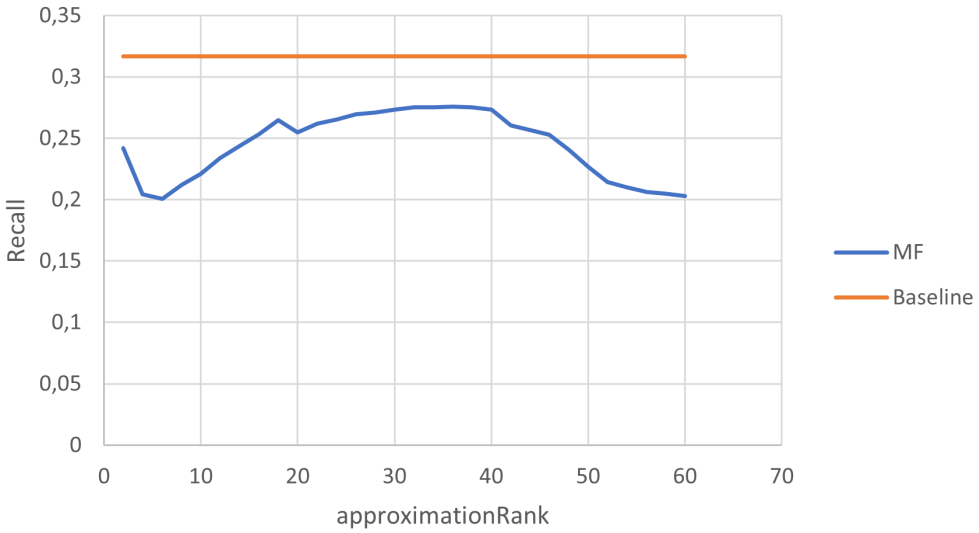

Figure A.1: Recall for different values of approximationRank, model: MF, customer: A.

Table A.1: NDCG, recall and novelty for varying values of *approximationRank*, model: MF, customer: A

approximationRank	k	NDCG	recall	novelty
2	1	0.2078	0.2418	0.8303
4	1	0.1881	0.2043	0.9579
6	1	0.1992	0.2007	0.9754
8	1	0.2088	0.2121	0.9828
10	1	0.2175	0.2209	0.9854
12	1	0.2307	0.234	0.9871
14	1	0.2407	0.2439	0.9881
16	1	0.2503	0.2532	0.989
18	1	0.2632	0.265	0.9899
20	1	0.2569	0.2549	0.9905
22	1	0.2651	0.2621	0.9912
24	1	0.2705	0.2654	0.9918
26	1	0.276	0.2697	0.9922
28	1	0.2804	0.2711	0.9927
30	1	0.2837	0.2732	0.993
32	1	0.2867	0.2751	0.9934
34	1	0.2884	0.2752	0.9936
36	1	0.2896	0.2758	0.9938
38	**1**	**0.2909**	**0.2753**	**0.9941**
40	1	0.29	0.2734	0.9942
42	1	0.2814	0.2606	0.9943
44	1	0.2802	0.2568	0.9944
46	1	0.2777	0.2527	0.9946
48	1	0.264	0.2406	0.9947
50	1	0.2485	0.2268	0.9949
52	1	0.2346	0.2143	0.995
54	1	0.2301	0.2099	0.9951
56	1	0.2273	0.2064	0.9952
58	1	0.2253	0.205	0.9953
60	1	0.2233	0.2029	0.9953

Table A.2: NDCG, recall and novelty for varying values of *approximationRank*, model: MF, customer: B

approximationRank	k	NDCG	recall	novelty
2	1	0.1656	0.1659	0.8007
4	1	0.2243	0.2259	0.9357
6	1	0.2343	0.2338	0.9622
8	1	0.2432	0.2407	0.9726
10	1	0.2752	0.2629	0.976
12	1	0.3105	0.2877	0.9787
14	1	0.317	0.2935	0.9805
16	1	0.3176	0.2939	0.9819
18	1	0.3196	0.2951	0.9825
20	1	0.3177	0.2943	0.9835
22	1	0.3184	0.2938	0.9842
24	1	0.3184	0.2947	0.9847
26	1	0.3202	0.2948	0.9851
28	1	0.3186	0.2942	0.9854
30	1	0.3172	0.2935	0.9857
32	**1**	**0.3254**	**0.3019**	**0.9857**
34	1	0.3249	0.3015	0.986
36	1	0.3223	0.2994	0.9863
38	1	0.3215	0.2978	0.9863
40	1	0.3193	0.2958	0.9864
42	1	0.317	0.2923	0.9866
44	1	0.3143	0.29	0.9867
46	1	0.3138	0.2914	0.9868
48	1	0.3123	0.2897	0.9869
50	1	0.3116	0.2894	0.987
52	1	0.3096	0.2875	0.9872
54	1	0.3069	0.2851	0.9872
56	1	0.3066	0.2866	0.9872
58	1	0.305	0.2843	0.9873
60	1	0.3044	0.2847	0.9874

Table A.3: NDCG, recall and novelty for varying values of *approximationRank*, model: MF, customer: C

approximationRank	k	NDCG	recall	novelty
2	1	0.6078	0.7201	0.6602
4	1	0.6103	0.7249	0.9037
6	1	0.6122	0.7285	0.9405
8	1	0.613	0.7297	0.9512
10	1	0.6139	0.7319	0.9586
12	1	0.6144	0.7327	0.9641
14	**1**	**0.615**	**0.7333**	**0.9666**
16	1	0.6148	0.7332	0.9689
18	1	0.6147	0.7333	0.9701
20	1	0.6149	0.7337	0.9716
22	1	0.6143	0.7328	0.9725
24	1	0.6142	0.7328	0.9728
26	1	0.6138	0.7322	0.9734
28	1	0.6136	0.732	0.9736
30	1	0.6132	0.7316	0.9738
32	1	0.5932	0.7133	0.9742
34	1	0.5933	0.7134	0.9744
36	1	0.5929	0.713	0.9745
38	1	0.5927	0.7128	0.9747
40	1	0.5925	0.7125	0.9747
42	1	0.5923	0.7123	0.9749
44	1	0.5923	0.7125	0.9749
46	1	0.5922	0.7123	0.9749
48	1	0.592	0.7119	0.9749
50	1	0.592	0.7119	0.9751
52	1	0.4562	0.7191	0.9753
54	1	0.456	0.7189	0.9755
56	1	0.456	0.7191	0.9753
58	1	0.456	0.7188	0.9754
60	1	0.456	0.7189	0.9754

Appendix B
Additional results, DMF

Table B.1: Grid search, model: DMF, customer: A.

approximationRank	k	realK	NDCG	recall	novelty
5	5	5	0.2666	0.2485	0.9803
5	10	10	0.2774	0.2669	0.9827
5	15	14	0.2951	0.2814	0.9841
5	20	18	0.2849	0.2746	0.9843
5	25	23	0.3083	0.2815	0.9855
5	**30**	**26**	**0.3215**	**0.2965**	**0.9869**
10	5	5	0.2865	0.2808	0.9886
10	10	10	0.2459	0.2449	0.9888
10	15	15	0.2884	0.2871	0.9893
10	20	19	0.3021	0.2979	0.9907
10	25	24	0.2922	0.2885	0.9907
10	30	28	0.2828	0.2827	0.9902
15	5	5	0.2753	0.2737	0.9912
15	10	10	0.3029	0.2972	0.9913
15	15	15	0.2876	0.2854	0.9919
15	20	18	0.3127	0.3052	0.9923
15	25	20	0.2718	0.2712	0.9922
15	30	25	0.3053	0.2991	0.9923
20	5	4	0.286	0.2769	0.9922
20	10	10	0.3103	0.2995	0.9932
20	15	14	0.298	0.2903	0.9933
20	20	18	0.321	0.3089	0.9936
20	25	19	0.3056	0.2953	0.9935
20	30	26	0.3155	0.3028	0.9939
25	5	4	0.2872	0.2722	0.9933
25	10	9	0.3096	0.2941	0.9937
25	15	13	0.299	0.2864	0.994
25	20	20	0.2778	0.27	0.9943
25	25	23	0.2763	0.2658	0.9947
25	30	28	0.2977	0.2821	0.9948
30	5	5	0.2648	0.2513	0.9944
30	10	9	0.2454	0.2347	0.9946
30	15	13	0.2655	0.2543	0.9946
30	20	19	0.2669	0.2518	0.9949
30	25	21	0.2782	0.2649	0.9951
30	30	23	0.2934	0.2763	0.995

Table B.2: Grid search, model: DMF, customer: B.

approximationRank	k	realK	NDCG	recall	novelty
5	5	4	0.3407	0.3176	0.9717
5	10	8	0.3521	0.3267	0.9742
5	15	13	0.3339	0.3097	0.9765
5	20	18	0.3476	0.3237	0.9787
5	25	24	0.3454	0.3196	0.9802
5	30	30	0.3387	0.3129	0.9806
10	5	4	0.3295	0.3065	0.9815
10	10	8	0.3388	0.3137	0.981
10	15	14	0.347	0.3187	0.9839
10	20	20	0.3492	0.3204	0.9846
10	25	24	0.3176	0.2949	0.9846
10	30	24	0.3428	0.3144	0.9844
15	5	5	0.3246	0.3003	0.9842
15	10	7	0.3159	0.2914	0.9848
15	15	14	0.3495	0.3213	0.9852
15	20	19	0.3349	0.3063	0.9866
15	25	25	0.3162	0.2898	0.9872
15	30	28	0.3116	0.2863	0.9872
20	5	4	0.3159	0.2942	0.9842
20	**10**	**9**	**0.3531**	**0.3238**	**0.9861**
20	15	14	0.3229	0.2979	0.9867
20	20	18	0.3196	0.2943	0.9867
20	25	23	0.3198	0.2929	0.9876
20	30	25	0.3155	0.2901	0.9874
25	5	5	0.3365	0.3105	0.9857
25	10	9	0.309	0.2866	0.9864
25	15	15	0.312	0.2869	0.9876
25	20	18	0.3062	0.2813	0.988
25	25	24	0.312	0.2874	0.9876
25	30	27	0.3037	0.2798	0.9874
30	5	5	0.3294	0.3043	0.9861
30	10	10	0.2919	0.2696	0.9872
30	15	14	0.317	0.2905	0.9873
30	20	19	0.2915	0.2725	0.9873
30	25	25	0.2788	0.2575	0.9881
30	30	27	0.2831	0.2632	0.9878

Table B.3: Grid search, model: DMF, customer: C.

approximationRank	k	realK	NDCG	recall	novelty
5	5	4	0.5526	0.7738	0.9535
5	10	10	0.5557	0.7791	0.9643
5	15	7	0.5145	0.7749	0.9642
5	20	7	0.5117	0.7699	0.9546
5	25	11	0.6605	0.7816	0.9645
5	30	16	0.658	0.7775	0.9609
10	5	3	0.5551	0.7783	0.9663
10	10	4	0.5554	0.7788	0.967
10	15	9	0.5153	0.7762	0.9667
10	20	13	0.5169	0.7785	0.9713
10	25	18	0.6614	0.7836	0.9727
10	30	17	0.6603	0.7821	0.9697
15	5	4	0.5567	0.7816	0.971
15	10	5	0.5561	0.7806	0.9714
15	15	3	0.5157	0.7782	0.9702
15	20	10	0.5157	0.7777	0.9721
15	25	9	0.6607	0.7833	0.9736
15	30	14	0.661	0.7843	0.9739
20	5	3	0.5555	0.7799	0.9721
20	10	3	0.5559	0.7805	0.9721
20	15	8	0.5153	0.7775	0.9725
20	20	13	0.5172	0.7811	0.9749
20	25	12	0.6614	0.7854	0.9732
20	**30**	**15**	**0.6614**	**0.7845**	**0.9731**
25	5	3	0.5559	0.7812	0.9728
25	10	7	0.556	0.7815	0.9731
25	15	8	0.5146	0.7771	0.9746
25	20	13	0.5142	0.7766	0.9748
25	25	12	0.661	0.7839	0.9732
25	30	11	0.6593	0.7805	0.9735
30	5	4	0.555	0.7801	0.9736
30	10	6	0.5548	0.7801	0.9746
30	15	11	0.5154	0.7782	0.9749
30	20	9	0.5146	0.7765	0.9746
30	25	14	0.6589	0.7799	0.9742
30	30	19	0.6586	0.7798	0.9751

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2022-01-17

EXAMENSARBETE Designing and implementing a recommender system for an E-learning platform
STUDENT Astrid Ekman
HANDLEDARE Rasmus Ros (LTH)
EXAMINATOR Elin Anna Topp (LTH)

Rekommendationssystem inom E-learning

POPULÄRVETENSKAPLIG SAMMANFATTNING **Astrid Ekman**

I takt med digitaliseringen blir det allt svårare för användare att själva orientera sig i det ökande informationsflödet. Automatiska rekommendationssystem kan vara ett sätt att underlätta för användarna att hitta relevanta produkter, men rekommendationssystem inom E-learning är hittills relativt outforskat.

Det har gjorts många studier på rekommendationssystem inom E-handel och underhållning, och idag finns sådana system på företag såsom Amazon, Netflix och Spotify. Studier om rekommendationssystem för E-learning plattformar är däremot inte särskilt förekommande, varför det här examensarbetet fokuserat på just detta.

Den enklaste formen av rekommendationssystem listar de mest populära produkterna för samtliga användare, medan andra rekommendationssystem är användarspecifika. De vanligaste formerna av rekommendationssystem är Collaborative, Demographic och Content based filtering. Collaborative filtering använder enbart information om vilka användare som konsumerat vilka produkter, kurser i det här fallet. Genom så kallad matrisfaktorisering går det att hitta underliggande preferenser hos användare, som sedan går att använda för att generera nya rekommendationer. Demographic filtering å andra sidan, använder personlig information för att hitta lika användare, och kan därigenom generera relevanta rekommendationer. Content based filtering hittar istället liknande produkter, till exempel genom att titta på produktbeskrivningar, för att skapa rekommendationer. Dessa tre delar kan kombineras för att generera relevanta rekommendationer. I detta projekt låg fokus på Collaborative filtering och Demographic filtering. Först delades användarna in i olika grupper baserat på enhetstillhörighet, rollbefattning, samt kompentenser. Därefter utfördes Collaborative filtering på respektive grupp.

Modellen evaluerades automatiskt och resultaten jämfördes sedan med resultaten från en modell där enbart Collaborative filtering användes. Kombinationen Collaborative och Demographic filtering gav alltid mer relevanta rekommendationer än när enbart Collaborative filtering användes. Vidare blev rekommendationerna bättre ju mer data (användare-kurs par) modellen tränades på, vilket generellt gäller alla maskininlärningsproblem. Den automatiska evalueringen kompletterades med att låta en kund manuellt titta på de rekommendationer som genererades för åtta liknande användare. Responsen var positiv, och det gick att se att många av rekommendationerna för en användare gick att återse i övriga användares redan avslutade kurser.

Sammanfattningsvis resulterade examensarbetet i ett rekommendationssystem som genererar relevanta rekommendationer, vilket förhoppningsvis leder till att fler användare läser fler relevanta kurser och därmed ökar sin kompetens.