Video Recommendation Systems

Finding a Suitable Recommendation Approach for an Application Without Sufficient Data

Marius Lørstad Solvang
Steffen Sand

Master's Thesis
Programming and Networks
60 credits

Department of Informatics
The Faculty of Mathematics and Natural Sciences

UNIVERSITY OF OSLO

August 2017

2017

Video Recommendation Systems

Marius Lørstad Solvang & Steffen Sand

http://www.duo.uio.no/

Print: Reprosentralen, University of Oslo

Abstract

In a world where more and more information is stored digitally rather than physically, the need to simplify the access to this information increases. This applies not only for literature, but also for most commerce businesses. While it is easy to take it for granted how easily we can find the information we want today, it lies massive research and effort behind many of the recommender systems that is constantly working in the shadows as we search and randomly browse for information to please our needs. We will in this thesis, look at the sport video application Forzify, and try to figure out what approach will give the best recommendations for users in this application.

We first look through research currently available regarding recommendation systems, and introduce the general approaches, before we discuss how they are used in practice by some of the largest and most popular systems on the web. We look for approaches in recommendation systems, that are best suited for the data in Forzify. We want to find approaches that gives accurate recommendation and scales well with large amounts of data in Forzify. Because we currently do not have sufficient user-data in Forzify, must we look at recommendations frameworks that are publicly available. We will evaluate different frameworks by running tests, and decide which framework fits our needs. Then, we have to look at what data we have available and which are similar to Forzify's data, so we can simulate a running system. At the end, we will evaluate the algorithms we have chosen with different datasets, and finally conclude which approach, or approaches, are best suited for the Forzify application.

Acknowledgements

We would first like to thank our supervisor at Simula Research Laboratory, Pål Halvorsen, for enabling us to work on this interesting topic in our thesis, and also for great counseling during the process. Without the feedback and meetings, this would not have been possible. We would also like to thank the team behind Forzify, as it is a great application in regard to our thesis, but also to enjoy from a football fan point of view.

Both of us who wrote this thesis would like to thank each other, for great teamwork, motivation and patience during more stressful times. We also want to thank Simen Røste Odden, who has been working on this topic as well. We gained loads by both meeting our supervisor together, and discussions afterwards.

Marius Lørstad Solvang & Steffen Sand

Oslo, July, 2017

Table of Contents

List of Figures

List of Tables

1 Introduction

Recommendation is something everyone is familiar with in one way or another. Whether a friend recommends a new book for you to read, or which workout-schedule you should follow, it all comes down to giving you good options and helping you make a choice. These recommendations are often given based on knowledge about what you like, or simply because someone who knows you believes that you will like it. *Recommendation systems* works in this exact way in the digital world, where the system tries to use data about users and items to predict what information you want to see. In this thesis, we will consider several approaches to create such systems, where the end goal is finding the best approach to use in the *Forzify* application.

1.1 Background

Over the entire world, there are millions of users of the internet, and the demand for more advanced and precise search engines increases all the time. Retrieving results based on only your basic search string is not good enough anymore, and to execute the more advanced searches, there is a wide array of algorithms which is getting more and more sophisticated every day. Such algorithms are often the search engine company's biggest secret. Even with these advanced search-algorithms, we are not quite satisfied. We also want to develop websites and applications that recommend content to users based on user-data gathered.

Opposite to physical stores, web-stores can have nearly unlimited numbers of products out for sales which demands an effective way for the system to pick out and display the most current products to its users so they are easy to see and access. How is this achievable and how does it work? For example, we have these applications: a news-site on the web want to recommend articles to users based on predicting what the user is interested in. What data can be gathered to achieve this? Web-stores wants to recommend products a user might be interested in based on earlier searches on the site or order history, or the products score and rank on the site. Media-streaming applications wants to be able to recommend additional content based on what the user has watched or heard before, based on data such as category, director or actors.

There are two different approaches of giving such recommendations on the web. We have personalized recommendations and non-personalized recommendations. Non-personalized recommendation systems are quite easy to implement and handle, as we do not need to gather and compare data from individual users. We could simply recommend the top-rated songs of all time on Spotify, or the most viewed videos on YouTube, to everyone. On the other hand, if we want to set up a personalized recommendation system, we need data from the users. This data needs to be gathered from every user, to be used for prediction of what content they like and would want to see next.

1.2 Problem definition

When designing a recommendation system, it is important to do research on the different usable techniques. There are a lot of options, and it can be hard to know exactly how the system should handle user interactions and data, and it is essential to consider other existing systems and how they work. In this thesis, we will work towards determining which recommendation approach is best suited for the Forzify application. However, given the current state of the application, we do not have enough user-interaction data to commit tests on this system.

How we can conclude which approach suits the needs of Forzify is therefore the main problem we are trying to solve in this thesis. To solve this problem, we will be thoroughly studying research related to recommendation systems, and compare them with each other. By looking at the data sources each approach depends on, we can find similarities with what Forzify has available and decide which approach to bring to further evaluation.

Without enough data from the application we want to create a recommendation system for, we need another way to run evaluation on the different approaches and algorithms, which is the second problem we will solve. We will look at available frameworks which contains pre-defined datasets and algorithms, then choose those containing similar data to what we have found in Forzify, and algorithms used within the approaches we decided to evaluate.

In a system like Forzify, which is expected to have a vast number of items, and probably users, it is important to consider the systems scalability. It is also important to conclude what kind of approach recommends with the best accuracy, both for recommendations within the

same domain, but also across domains. Therefore, the third problem we will solve in this thesis, is which approach provides the most accurate recommendations, and which approach is most scalable. To evaluate accuracy, we will find and look at different measuring techniques. To measure scalability, we will look at the different algorithms' training time and prediction time.

1.3 Scope & Limitations

In this thesis, we work towards determining which approach and implementation-method is best suited for a recommender system, without having the necessary data from the application at hand, namely Forzify. By doing extensive research on recommendation systems and approaches, we gather necessary information about algorithms and possible frameworks that we can use to evaluate and give us meaningful results, as we answer our research problems. We analyze and compare the different approaches, and find similarities between the data used in these applications and the potential data we can use from Forzify.

The deployment of Forzify has been delayed longer than what was planned when this thesis was started. Because of this, most experiments conducted in this thesis will be based on theoretical data and datasets like MovieLens and Jester, in an *offline environment*. Optimally, when designing a recommendation system, you would want to do your evaluation against real data on the application in consideration.

1.4 Contributions

The backbone of all our findings and conclusions in this thesis, is the thorough research we have done on the main recommendation approaches. With this knowledge as a foundation, we look further at several different approaches in recommendation systems, to find what is best for Forzify. We present different recommendation frameworks which can be used to evaluate the approaches, to help us determine which approach to use in Forzify.

The current data in Forzify is both user-data and content-data. The user-data is gathered both implicitly and explicitly, and the content-data is in the form of simple tags and descriptions of videos. We are going to determine what recommendation approaches fit considering this data. Because we have limited user-data in Forzify, we have to evaluate our approaches based on

datasets that are similar to Forzify's data. As there are several approaches that exist for recommendation systems, will we choose some approaches that we will evaluate within a chosen framework with some built in algorithms that provide recommendations. The chosen algorithms will be evaluated with measuring their accuracy and scalability, and we can then conclude which approach is the best fit for Forzify.

1.5 Research method

In this thesis, will we use the research method introduced by the ACM Task Force on the Core of Computer Science [9]. This is a report on how research should be conducted in computer science, and their *design paradigm* identifies these steps:

1) State requirements
2) State specifications
3) Design and implement the system
4) Test the system.

While working through these steps, we will be looking at what data is currently available on Forzify, and what kind of recommender system we want to create. By looking through research already available on recommender systems, we will try to find the best suited approach to use for Forzify. With the information we have gathered, we will choose a small set of candidate algorithms which we will implement and run on some datasets with the help of a chosen framework.

1.6 Outline

- In Chapter 2, we will introduce the general concept of recommender systems and why we want to use them. Existing approaches and applications will be discussed, and compared by their strengths and weaknesses. We also start looking at how we can evaluate recommendation systems.

- Chapter 3 introduces the Forzify application. We discuss the current state of the application and its general features, as well as what data we have available and which recommender system is currently running. Based on this information, we start discussing what features we want in a new recommender system and which approach might be best suited to achieve this.

- In Chapter 4, we will introduce several recommendation frameworks that provide algorithms, which we discuss in detail, and different datasets. The datasets most similar to the data in Forzify will be used in further evaluation.

- In Chapter 5, we use the information we have gathered about suitable approaches, datasets, algorithms and data in Forzify to run evaluations on the datasets with our candidate algorithms. We gather data about the two main measures of recommendation systems, namely accuracy and scalability.

2 Recommendation systems

In general, we can split the recommender system approaches into two broad groups, namely *content-based filtering* systems and *collaborative filtering* systems**.** In addition to these two, we have a few other approaches that will be introduced in the next few sections, as well as some possible *hybrid* versions. Before we dive into these approaches, we will introduce a brief example of how user-data can be stored in form of the *utility matrix model,* Table 1. This will give some context to the term user-data, which is crucial in several of the methods we will introduce, as they need proper user-data to predict items.

	Item1	Item2	Item3	Item4
User1	4	5		2
User2		3	1	
User3	2	3	4	1

Table 1: Illustration of user-item connection in form of a utility matrix [3]

In search recommendation systems, it is quite typical to have two classes of entities, namely *user* and *item.* This is typically used on a Netflix-type of site, where users can browse and watch different content, or other stores where you can rate the items you have bought. The point of this is to create a connection between users and items by linking their ratings with different items. This can be presented in a matrix like Table 1, where we can see three users and four items. The numbers inside the cells represents each user's rating for the different items, on a scale from 1-5. The blank cells are situations where the users have not given a rating for the item. By having a recommendation system implemented, the system can be used to *predict* rating for the items a user has not yet rated, represented by the blank cells in the matrix. It will also give the possibility to predict ratings for other users, for whom the system has determined have the same interest as another user. A system does not always recommend

based on a rating-scale, however, but can also be used to give *top-n recommendations,* which identifies a set of *N* items that will be of interest to a user [10].

Now that we have discussed the general notion of how to consider recommendation systems, we will further in this chapter give examples of how they are used in practice in some of the biggest and well-known systems in the world, to get an even better insight and understanding of this topic. Even further, we will investigate each recommendation approach and their ability to deal with some of the issues that we come across, when comparing them.

2.1 Why use recommendation systems

In the previous section, we introduced the general idea behind recommendation systems and talked about the relationship between users and items in these systems. We might think that these systems are only in place to satisfy the users/customers, but this is only one side of it. The companies behind the systems that, for example, sell products on their websites have great benefits both directly economically and by all the sales-information gathered over time for further business planning.

Here is a list, from Ricci et al., [33] of just a few of the benefits such *service providers* get from having a well-made recommender system in place, which ultimately also gives users the best possible experience and user satisfaction.

- *Information gathering:* a very important part of business management is information gathering. In the case of a recommender system on a grand-scale webstore for example, the company behind it can make various decisions regarding restocking items and notice trends to determine what needs increased production, or maybe which items needs to be advertised more to a specific user-group.

- *General increase in sales:* for companies behind commercial recommender systems, the obvious goal is to keep their business blooming and sell as much as possible. This is partly achieved by selling more products, because users have gotten the best possible recommendations and advertisement. This also applies to content-based web-sites that have no direct payment from their users, but rather base their success on views.

- *Display more diverse items:* as we said in the section 1.1, there are way more items stored on a website than in physical stores, and being in control of which items are on the frontline and not forgotten in the abyss is important for large websites. Some users may also be interested in getting all items from a given category, whether they are popular or not. These items are not generally advertised because of the lack out popularity, but they need to have a way to be accessed either way.

- *Gain user loyalty:* with a good recommender system, customers will feel that the more they use the site, the more tailored it will become for his/her specific needs and interests. This will greatly increase user satisfaction and make it more likely customers will return and keep using the site instead of others.

With all this primarily working to the service provider's advantage, we also have several benefits for the users of the service. One of the characteristics of a well-made recommender system, is that there is a finely tuned balance between the profits gained for the business and the customers. While the tasks of the recommender system might differ between these two actors, there is, obviously, a link between how well the users' recommendations are, and how the business can benefit from this regarding the listing above.

Some of the tasks [33] a recommender system has that concerns the *users* are:

- *Find Some Good Items:* The most typical use of a recommender system is to recommend "just some good items", where the system predicts and lists some good items based on how much the user would appreciate the items.

- *Find all good items:* A user sometimes has the need to see *all* possible good items, preferably also in a ranked list, to gather information from all reasonable sources on some topics. This is especially relevant if the item-base is small.

- *Recommend a sequence:* While exploring either academic articles or movies, an recommender system should be able to give a user items not necessarily pleasing to just one recommendation, but also a group of items pleasing as a whole. If this paper is recommended for example, why not also recommend papers about machine learning or something similar.

- *Improving your own user experience:* Gaining user-loyalty is a great benefit for any business, and on the way to achieve this we want to have our recommender system constantly improving for every user. This is a two-way relationship, and by staying loyal to a system, the recommendation system should improve any user's user-profile and provide them fresh and up to date recommendations.

- *Help and/or influence others.* While some users only care about their own well-being in a system, there are some who are happy to provide and share information with others. By giving their opinion and ratings on different items, they feel they contribute and help others. This is a two-way street however, as some users might also take advantage of this in a malicious way. This is mostly relevant on a bigger scale than just single users, but on a commercial web service where competitors try to sell their products alongside each other, there surely is a possibility to down vote others' products, or up vote your own, and influence the recommender system. To achieve a grand effect of this, you would need to control a huge amount of accounts, but botnets and related topics are not something we will go deeper into in this paper.

2.2 Recommendation system approaches

Recommendation systems provides us with several ways of finding items of interest, whether it is which video to watch next, or what book to buy [16]. These recommendations are based on data and metadata about users and/or items, and the different approaches we will introduce in the following sections takes advantage of this data in different degrees and ways. We start by looking at the two main approaches, namely content-based filtering systems and collaborative filtering, before looking at a few approaches that mostly work as supplements to those, or used when the system at hand does not gather enough data to make these approaches effective.

2.2.1 Content-based systems

Content-based recommender systems recommend items based on what the system knows about which features and items a user is interested in. The actual content the system learns from, refers to an item's description in form of attributes and metadata, which is illustrated in

Table 2. With this content-data, the system can use this as training data to create a user-specific classification, and make further predictions based on this [1].

The properties/characteristics for an item are gathered in what we call an *item profile.* An example of a site that uses this approach to some extent is Netflix. In Netflix's case, some of the item properties are actors, director, rating or genre for each video. It is very important to have a good and clean representation of items in a content-based system, as their properties defines how the system can find similarities between them. In Table 2, we see a basic example of how movies can be represented in a database, like Netflix's.

ID	Name	Genre	Language	Director	Actors
0001	From Russia with Love	Action	English	……	……
0002	Harry Potter	Adventure	English	……	……
0003	Casino Royale	Action	English	……	……

Table 2: Typical representation of content in a movie database

When the system has learned the user's previous ratings of items within the same genre or with the same director, it will weight its recommendation higher for those items in the future. To do this properly, however, the system will need some sort of user profiles as well, and the relationship between users and items is like we have illustrated in Table 1, the *utility matrix model*. A new user profile will contain the user's preferences, which are generally what the user is interested in. It will also contain the history of the user's previously watched movies, along with the item's description and a search history. On a movie streaming platform, we can use this information to either let the user go back to a movie that was recently watched, but not finished, or filter out the movies that has already been watched. With all this information stored about both the user and which items the users are interested in, we can start making recommendations.

One of the strengths of content-based systems, is that it deals with the *cold start problem* well, which is a common issue for new users in other approaches, where the system does not have enough data to recommend new items. In this approach however, the system can recommend new items as soon as the user has some sort of interaction history, at the cost of lower quality recommendations. However, because of how the system simply finds similarities between items, this approach is *self-biased,* which means that it has a problem recommending items of *diversity*. Because of advantages and disadvantages such as these, approaches are often merged into hybrid versions, to accommodate each other, and deal with the issues related to them when used alone. Hybrid systems will be discussed further in Section 2.2.5, and more strengths and weaknesses will be discussed in Section 2.4.

2.2.2 Collaborative filtering systems

Collaborative filtering systems focus on the relationship between users and items, which is illustrated in Table 1. The way this method recommends an item for a user, is by checking the columns in the utility matrix and comparing it to other users. If a user's ratings of items are similar to another user's rating, we can conclude that they have similar interests, and recommend items that the second user likes to the first user, which fills in the blank cells in the matrix. To calculate such similarity between users or items, we can use *cosine similarity,* which is exemplified and discussed more in detail in Section 4.2.3.

There are two main approaches to do collaborative-filtering; *model-based* and *memory-based filtering*, where the latter is also commonly known as *neighborhood-based filtering.* Memory-based filtering was one of the earliest ways of generating recommendations, and is used where the user-item ratings combination is predicted based on their neighborhoods, and can be defined by *user-based* collaborative filtering or *item-based* collaborative filtering [6].

Memory-based filtering

In user-based collaborative filtering, the recommendations are created by gathering the ratings by similar users to a selected user (active user), so that we then can give recommendations to the active user. To do this, we must compute a weighted average of ratings from similar users, for items that the active user has not yet visited. For example, if user A and user B has shown

positive interest in several of the same movies by giving them equal ratings, we can predict that user A has an interest in a movie he has not seen, but where user B has rated it highly.

When using item-based collaborative filtering, we create predictions based on the similarity of items. In this approach, if we want to predict the rating of a target item for any user, we determine a set of items similar to the target item [1], and by looking at the ratings of the items in the set, we can predict whether the user will also like the targeted item.

Model-based filtering

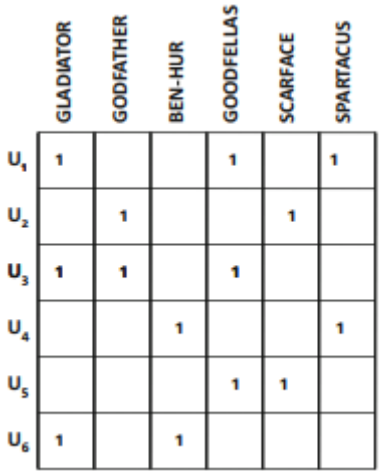
The model-based filtering technique revolves around the creation of predictive models. Machine learning and data mining plays a central part to create said models, where methods like decision tress, Bayesian methods, clustering techniques, and rule-based models are used to gather vast amounts of data [1]. The Bayesian network model is built of probabilistic model for collaborative filtering problem. The clustering model looks at collaborative filtering as a classification problem, and works by clustering similar users in a class and calculating that a particular user has the same interests as other users in a class, and finally calculates the conditional probability of ratings. The rule-based approach creates associations between purchased items and new items, which it then generates recommendations from.

Representations of ratings

For several of the recommendation systems to work, and collaborative filtering techniques in particular, the system requires a way of tracking ratings for items. The way the rating of items is set up, differs from system to system, but in general we have two approaches, represented in Table 1 and Figure 1. Table 1 shows us what is known as *interval*-based ratings, or *ordered* ratings, and is commonly used by video-streaming websites that allows us to rate the videos we have watched on a scale from 1-5 stars. The range of the rating scale can vary as well, like the Jester recommendation engine which uses a scale from -10 to 10 [1], which is not as common.

Figure 1 shows us a representation of *unary* ratings, and such a matrix is known as a *positive preference utility matrix* [1], as there is no possibility of negative ratings, only positive. This is the case for systems where you can only press a like-button, which we will later see is the

only way Forzify collects user-rating currently. The values gathered are not necessarily 0, 1 or nothing, as they can also be a value to count page-views or video-views.



	GLADIATOR	GODFATHER	BEN-HUR	GOODFELLAS	SCARFACE	SPARTACUS
U_1	1			1		1
U_2		1			1	
U_3	1	1		1		
U_4			1			1
U_5				1	1	
U_6	1		1			

Figure 1: Unary ratings example [1]

While unary and ordered ratings are the two main types of ratings, how the system collects the rating data is split into two groups; *explicit* ratings and *implicit* ratings. Explicit ratings can be both unary and ordered ratings, but what they have in common is that it is information the system actively gathers to create recommendations. Explicit rating is therefore the best indicator or a user's preferences, as this information gives concrete data of a user's interests. On the other hand, implicit ratings are values gathered rather effortlessly by the system, through for example browse history. This data is not necessarily a good indicator of a user's interest, as we cannot conclude that a user really likes an item just by visiting it, or watching a video once. However, in the case where the unary data is page views or amount of times a user has watched a video, it can be viewed as a good indicator.

2.2.3 Knowledge-based systems

The *knowledge-based* recommendation approach is solely based on item assortment, user preferences and recommendation criteria. This approach is often used for items that are not often visited, like for example luxury goods or expensive cars. By default, such items will more often than not, lack sufficient user ratings, and a collaborative filtering approach will not

be beneficial to use, as it will get bottlenecked by the cold start problem, described in Section 2.4.3. To counter this, domain knowledge and user preferences provides necessary information to the system, and is both used to calculate similarities and base recommendations on sets of explicit rules and constraints [1].

Knowledge-based recommender systems are split into two groups, defined by how they achieve their goal of creating recommendations: *constraint-based* systems, and *case-based systems* [1]. A constraint-based system creates recommendations based on a set of explicitly defined rules between a user's preferences and the features of items, where the system retrieves a set of items which fulfills the constraints defined by a user's preferences. The case-based approach on the other hand, retrieves items by using similarity measures [24].

A knowledge-based recommender system can take form as a conversational system, which means that the system will get user requirements and preferences from a feedback loop. Search-based recommendation is also an approach, which gets its user feedback from user's answers to questions. Navigation-based recommendation is based on use feedback provided from critiques, where it is typical that the user can alter his request for an item that has already been recommended, to narrow down and navigate towards a satisfactory result.

While this approach handles problems like cold start well, it is limited and totally dependent on expert domain knowledge. In addition, as this kind of system is based on current knowledge and does not learn more and more about users, its ability to adapt is relatively poor. This is not necessarily an issue, however, as this approach is preferably used either in combination with other approaches in a hybrid recommendation system, to deal with the cold start problem, or where the consideration of user's ratings of items are not relevant in regard to what kind of items are in the system.

2.2.4 Demographic-based systems

A *demographic-based* system takes advantage of demographic data from users to create recommendation groups. Such systems do not need domain knowledge, but instead it requires users to specify their demographic attributes. Demographic attributes can vary from gender, occupation, education or simply age, and is essential information for the system to be able to group a user with users who has similar features. Many systems use some sort of demographic

recommendation, but this is in most cases not a standalone technique in practice, and is therefore rather used as a supplement in knowledge-based or hybrid systems [1].

As with knowledge-based systems, this approach does not require a learning period, only that the user specifies its own demographic data, and is therefore relatively static. However, as with other systems that does not base its recommendation on what the system learns over time, it requires user interaction and the user's willingness to share its personal information. With privacy as a hot topic in today's society, this introduces a disadvantage with this approach, which is its inability to recommend items without a user's personal data [7]. A demographic-based system will also have issues with recommendations of new items, as they cannot be recommended properly before they are interacted with by several users who indirectly determines to who the item shall be recommended for.

2.2.5 Hybrid recommender systems

As there are clear drawbacks in each of the different recommender approaches, there has been done research [7] concluding that combining several approaches together would create a better system in many cases. The biggest standalone techniques we mix to create hybrid systems are the four we have introduced in the past subchapters; content-based, collaborative filtering, knowledge-based, and demographic. The main purpose of creating a hybrid system is to deal with issues like the cold start problem and sparsity, which we talk about in Section 2.4.3 and 2.4.4. In the list below, we introduce the general ways of how such systems can be created.

- Making content-based and collaborative-based predictions separately and then combining them.
- Adding content-based capabilities to a collaborative-based approach.
- Adding collaborative-based capabilities to a content-based approach.
- Combining approaches into one model.

To get into more detail, we can reference some of Burke's [7] list of hybrid categories; *weighted*, *switching*, *mixed*, *feature combination* and *cascade*.

- Weighted hybrids: combining the results of different recommendation techniques by using the score from each one with different degree of importance, to compute a final recommendation. Relies on the fact that the different scores are linearly combinable.
- Switching hybrids: this is a hybrid that will switch from one technique to another, depending on the situation. In practice, this could be a system using a content-collaborative hybrid, where the content-based recommendation is the first step. If the recommendation results generated here is not satisfying, the collaborative approach is used to try to achieve a better result. This approach's biggest challenge is to find a good switching condition.
- Mixed hybrids: in this hybrid approach, we merge several ranked lists from different recommender approaches into one final ranked list. How to compute the scores for the final list from the ones created by the other techniques can be challenging, but the simplest example is just adding their scores together.
- Feature combination hybrids: features from different recommendation sources provide input to the "final" technique.
- Cascade hybrids: this hybrid uses the output from one technique as input to another that refines the recommendation result. The techniques involved have a pre-defined priority and order as to when they are executed. If the first technique gives good enough results that are clear and distinct, the need for the second technique is not there and will not be deployed. This is an effective way to save resources.

A good and simple example of the usage of a hybrid system is Netflix. Netflix uses collaborative filtering by comparing the watching- and searching-habits of similar users, as well as showing the users movies that share similarities with other movies, which is in the content-based filtering domain.

2.3 Applications of recommendation systems

We have now discussed several ways of using recommendation systems, and will in this section take a look at some of the biggest and most known websites to see how they work in practice. We will discover that functionality we take for granted, have underlying technology that most people have not even heard of. However, a lot of these technologies are big secrets

for the involved companies, and is not published in great detail except the general idea behind them.

2.3.1 Netflix

The movie streaming service Netflix is a platform which is important to consider when we want to develop a recommender system for the application Forzify. There are several similarities between the two, with the biggest similarity being that they both gives users the possibility of watching videos online, and rate them to create further recommendations. Netflix uses different tags like action, comedy and so on to identify the content of the videos, and this is used in combination with implicit user data like user history, which will contain information of what genres, actors and titles the user has already watched. The first page on Netflix contains movies that are popular now, what you have watched before, and then movies under genres that may contain actors you have watched before.

The system also gets information from what your friends have watched, and will recommend videos to you based on this. Netflix uses a lot of different algorithms that are optimized for different situations, and some of them are listed below [21].

The personalized video ranker is an algorithm which sole purpose is to find the best personalized recommendations for each user, where it orders an entire catalog of videos from genres or different groupings and personalizing them for each user profile. Then a resulting ordering is used to select the order of the videos in genre and other rows.

Another algorithm Netflix uses is called Top-N video ranker, which generates the recommendations in the top picks row. This algorithm has the job of finding the best few personalized recommendation of the users, where it focuses on only the head of the ranking.

The trending ranker algorithms focuses on giving recommendations that are short-term and temporal, and are ranging from a few minutes to a few days. This ranker looks at two types of trends, (1) those that repeat every several months, but also have a short-term effect when they occur, and (2) one-off, short-term events, which is when something in media is trending and drives users to watch similar movies or documentaries. In Figure 2 we can see some items that have been recommended to the user, using this algorithm.

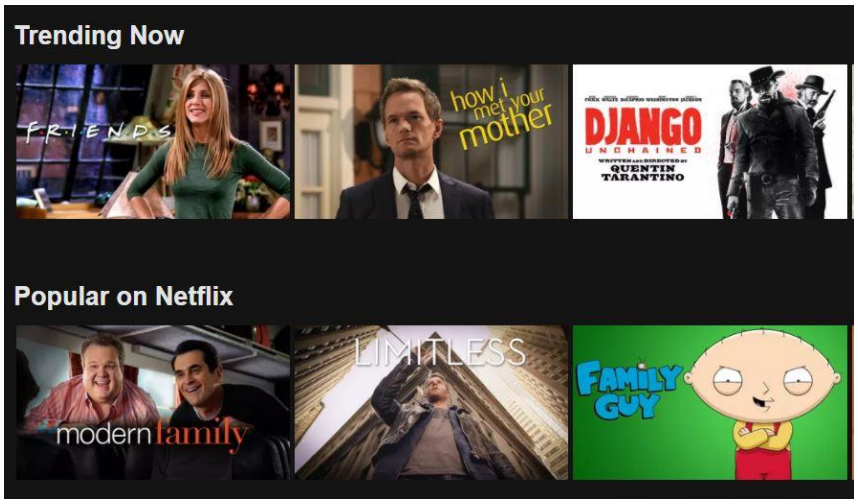


Figure 2: Netflix showing items that are trending and popular now

Netflix also allows the user to watch movies in small bits, and the *continue watching ranker* is an important algorithm that sorts the subset of recently viewed videos based on the best estimate of whether the user will continue watching the title. This ranker uses the time elapsed since viewing, the point of abandonment, whether different titles have been viewed since, and the devices used.

In Netflix, the user is shown a "Because You Watched"-row of videos, and is driven by the *video-video similarity* algorithm. This algorithm is non-personalized and computes a ranked list of similar videos. Even though this algorithm is non-personalized, it is still personalization that decides if a video makes it into the Because You Watched row. Figure 3 shows the items that is recommended when a user has watched the title "The Keepers", and uses the Because You Watched algorithm to recommend these titles.

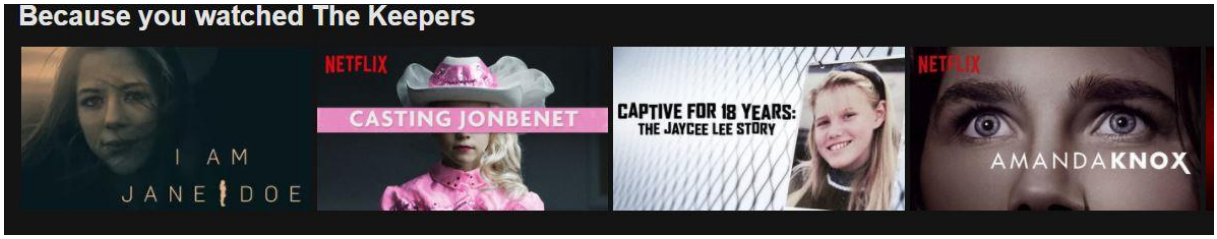


Figure 3: Netflix showing items that are similar to an item you have watched before

Page generation: row selection and ranking**.** This algorithm uses every algorithm already described to generate every single recommendation, where it looks at the relevance of a row of videos to a user as well as the diversity of the page.

Netflix introduced a new way of matching titles to user called “The thumbs up, thumbs down rating system” [2], which is a unary rating system. This was first well received, but ended up with users not liking it. The reason behind this, is that there no middle ground between liking or disliking items. Users were hesitant to rate titles when they only mildly enjoyed a title, because they did not want to mess with their ratings. In Figure 4, we can see how the series Narcos has been given a 97% match based on previous liking and disliking of items.

Figure 4: Netflix showing a percentage of how well this item matches your profile

2.3.2 YouTube

YouTube is also a video streaming website, but on this platform users share their own videos by uploading them for other users to view, rate, share and comment on. The recommendation shows the users what videos are popular right now, videos that are similar to what the user have already watched, and videos you may not have watched earlier created by one of the user’s subscriptions.

The recommendation system for YouTube gets its data from the user's activity and content data, which is the tags, titles, description of the video. The user activity data is collected from the user's video ratings, favorites, views and how long the user has watched one video.

Because YouTube lets users upload their own videos with a personalized thumbnail, title and description, it cannot create a recommendation based on video clicks. This user can end the video just after realizing that the video is not of interest, and because of this there must be created a recommendation based on several factors which are discussed below.

Applications like YouTube must also handle searches where a user does not have a history of earlier searches. This is called a cold start [37]. There are different ways to handle a cold start, but YouTube generally handles this by recommending the highest ranked videos and videos that are trending right now [37].

What kind of algorithm YouTube is using is hard to know, because the details are kept somewhat a secret and have not been published to the public. However, there is an article on someone that reverse engineered the YouTube algorithms [20]. The different algorithms that are doing work for YouTube is recommended, suggested, related, search and metascore. Which are all optimized for "WatchTime", this is a combination of Views, View duration, Session Starts, Upload Frequency, Session Duration and Session Ends. For a video on YouTube to become popular you need to get a lot of views in the beginning of the video release, and is calculated in something called *View Velocity. View Velocity* is the number of subscribers a user has, that is watching a video within the first 48 hours, if the video is not clicked on by a large number of subscribers, it will impact negatively on the next video you publish. View duration is a calculation of how long a video must be watched, until it can be decided that, that video is of interest to a user. Session Starts is how many of a user's subscribers start their session on YouTube with watching the users video. Session Duration is how long a user's videos are keeping other users on YouTube while watching the users content, and how long they stay after they are finished watching. Session Ends is a negative metric that calculates how often someone leaves YouTube while or after watching a user's videos. Upload Frequency is also a metric used, which basically is how often a user uploads content and how fast the subscribers watch that content. From this article [20], which attempts to reverse engineer the algorithms used by YouTube, they have come up with an algorithm theory. They claim that YouTube's algorithm is designed to promote channels, not individual videos, but, it uses videos to promote individual channels.

2.3.3 Spotify

Spotify is a music streaming service, that lets a user share and stream music, which makes it comparable to our case of Forzify in some ways, and it is therefore important to consider Spotify when we are talking about recommendation ranking. Spotify's "Discover Weekly" service is a recommendation system, and is creating a playlist for a user with undiscovered music based on the user's history. This service has become very popular since its release, and has been streamed 1.7 billion times and the Discover Weekly model can be seen in Figure 5.

From the user's song history, Spotify creates a "taste profile" and then finds playlists with the same features. With this profile, the system creates a new playlist which contains undiscovered songs from the playlists found from the last search as shown in Figure 5.

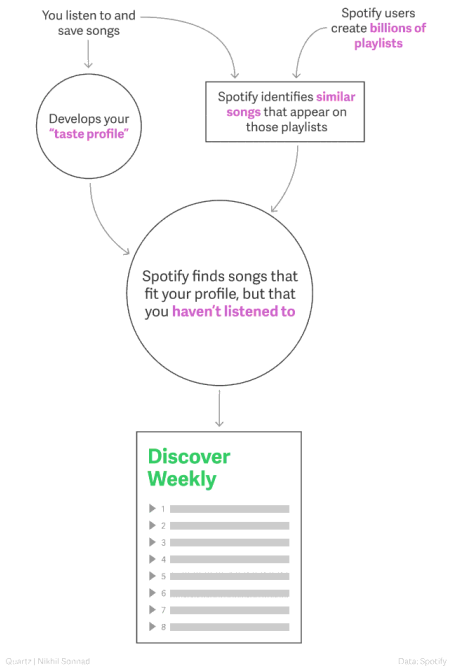


Figure 5: Spotify discover weekly model [32]

2.3.4 Amazon

Amazon is an electronic commerce and cloud computing company, and is the largest web-based retailer in the word, by total sales and market capitalization. Amazon uses recommendations to predict what items a user may want to buy, and have different

approaches to how it does that. There is a “Your Recommendations” button, that if clicked returns you to a page with a list of items that is specified to what you may like and have browsed earlier. In Figure 6, there is an item that is clicked on and different items that is frequently bought with that item shows up and is recommended to you. This is an example of item-based collaboration filtering being used, where the item clicked is similar to the items shown in Figure 6.

Figure 6: Amazon, frequently bought together

When a user views an item of interest, Amazon also gives recommendations of other items that have been previously viewed by users who has also shown interest in this item. Such recommendations are shown in Figure 7.

Figure 7: Amazon: "customers who viewed this item also viewed"

One of Amazon's also great marketing tactics is to recommend items to a user via email. Kwasi Studios [27], has written an article on how Amazon is able to give a user recommendation, based on a 3 minutes viewing of items.

2.4 Comparison of approaches and recommender techniques

By now, we know that there are a lot of different approaches to creating a recommender system. This makes it important to look at the advantages and disadvantages of the approaches, so that we can tailor our recommender system to the best of our ability to fill the needs of the application at hand.

In Table 3, we have listed some of the potential problems in recommendation systems and the different recommendation approaches. We will determine where these problems occur and whether the approaches handle them well or not. We have decided to keep hybrid recommendation systems out of the matrix, because this approach can be a combination of all approaches.

	Content-based	Collaborative	Knowledge	Demographic
Popularity bias		-		-
New items		-	+	-
New users		-	+	
Self-biased	-	+		+
Over time learning	+	+		+
Identify cross-genre niches		+		+

Table 3: How different approaches deal with different issues related to recommendations

2.4.1 Popularity bias

In collaborative filtering especially, there will always be users who have a unique taste that does not match most other users. This can lead to a problem when the system wants to compare users to each other and then give recommendations. To handle this type of users, the recommender system must adapt so that it can catch these users and then give recommendations with a content-based approach. Then it can give recommendations of items that are similar to what the user has already liked or rated.

2.4.2 Scalability issue

The scalability of recommender systems can be a problem in most approaches [28], but especially in systems where machine-learning is essential, which is often the case in collaborative-filtering techniques. In collaborative filtering, we will have an exponential growth since there will be new users who will give new ratings all the time, on both old and newer items. An approach to distinguish the scalability issue, would be to use a hybrid recommender system which can switch between small and large calculations, and between different approaches.

2.4.3 Cold-start problem

A great challenge in recommender systems is the cold start problem [28], which is when the system has a tough time recommending items to new users or recommending new items into the application. In Section 2.2, we learned that content-based systems are based on item descriptions and user preferences, and collaborative filtering creates recommendation on similarities between user's information. Especially collaborative filtering techniques suffers greatly by the cold start problem, while knowledge-based filtering does not. Often the cold start problem is talked about when there are new users or new items entering a system, but this is not always the case. A system does not know if a user wants the same items as he/she did when they last visited, this user can have bought the item that was looked for earlier and have no longer interest in this item anymore. The problem will always exist, but these are circumstances where a user buys items that are not collectibles, such as desks, sofas, et cetera.

We can minimize the cold start problem with different approaches, and one of the ways to do this, is to use the "what is popular now" strategy. This can be determined from what is popular recently or demographically, which can be determined by for example GPS coordinates, which site they came from or knowing the device/operation system they are using. There is also the solution of when interface agents can share their information of the specific user.

2.4.4 Sparsity problem

In systems where users either purchase or rate items, there is usually a vast number of items that will not be rated or purchased, like in the empty cells of Table 1. This can influence recommendations negatively, as the system cannot collect similarities between items that are not rated or purchased and will therefore not be able to recommend them [28]. The sparsity problem can be handled by reducing the number of items and users an algorithm runs on. This has been proven efficient from the Netflix Prize competition. In content-based systems, this is a frequent problem, since this type of filtering does a lot of recommendation based on items.

2.4.5 Self-biased

When a user is recommended items based only on which items this user has rated or bought previously, it can become a problem that this user will only get recommendations from the same category. For instance, if a user in the Netflix universe only watches movies or series which is considered crime, the user might get overloaded with recommendations from the crime genre.

2.5 Evaluating recommender systems

Deciding which approach to use when developing and implementing a recommender system varies on several levels. Recommendation systems have several metrics we need to take into consideration, like accuracy, scalability, robustness, user experience and more.

The most obvious way of evaluating a recommender system might be to look only at its prediction accuracy. This means retrieving only items that are extremely close to matching the meta-data of earlier items retrieved for each user. Having such an accurate recommender system is all well and good, but in many applications, users might want to discover a bit more than their exact anticipation. This is where it gets even trickier, and we will need to evaluate the recommender systems appropriately as to what kind of application it is being implemented for. To finally test these recommenders, we can either do offline experiments on existing datasets, a limited use-case study where users are asked to perform tasks with the current system, or online experiments on an up and running system with loads of data and users.

2.5.1 Offline testing

The least demanding way of testing out a recommender system is to do offline testing with simulated data and users [36]. This is a cheap way of first-time testing new systems, as it requires no online functionality, and we deal only with pre-arranged datasets. The goal of such testing is to simulate as real as possible user behavior scenarios, but it is limited in its ways when compared to real-time online testing because of the small amount of cases that gets tested. Because of this, what we really want to do here is filter out the most inappropriate recommendation-algorithms before going further to real user testing or even online deployment with way more user interactions and behaviors.

Datasets

When we want to create a recommender system, we must test different algorithms on some kind of data to know how efficient and accurate an algorithm can be. This process often takes place in an offline environment, which is the way we will go about evaluating algorithms later in this thesis. The data we can test our algorithms on must be represented in an orderly fashion, and are often called datasets. A dataset is a table containing related sets of information, and in the context of this thesis that information would be user and item profiles, where every user has rated different movies and a preferred genre, and the items will contain movies with which genre they belong and what overall rating they have. There are several different datasets that can be used, such as Lab41, MovieLens, Jester and more. In the next chapter, we will look into popular algorithms used for recommending and test these algorithms with different frameworks on datasets. There are a lot of different datasets when we want to evaluate recommendation algorithms, some popular ones are MovieLens, Jester, Book-crossing, Last.fm. They have ratings from different categories and have different number of users. MovieLens have four different sized datasets, one with 100 thousand ratings, one with 1 million ratings, one with 10 million ratings, and the biggest one with 20 million ratings. The ratings from MovieLens are ratings on movies by users, and are rated orderly from 1 to 5. Jester come with three different datasets, dataset 1 with over 4.1 million ratings, dataset 2 with over 1.7 million ratings and dataset 2+ which contain the dataset 2 with another 500 000 new ratings. Jester dataset is rating jokes with a rating ranging from -10 to 10. The Book-crossing dataset contains 1.1 million ratings on books.

	Users	Items	Ratings
MovieLens 100K	1,000	1,700	100,000
MovieLens 1M	6,000	4,000	1,000,000
MovieLens 10M	72,000	10,000	10,000,000
MovieLens 20M	138,000	27,000	20,000,000
Jester 1	73,421	100	4,100,000
Jester 2	59,132	150	1,700,00
Jester 2+	79,681	150	2,200,00
Book-crossing	287,858	271,379	1,149,780
Last.fm	21,000,000	600,000,000	

Table 4: Number of users, items and ratings in different datasets

2.5.2 Testing with real users

One of the most used techniques to test new developments in the programming world is by using test subjects who are given certain tasks to perform on the new system. While the tasks are being performed, the developers can observe and take notes of their behavior and see what scenarios the system struggles with, as well as what the users are having a hard time doing. We can see what tasks were particularly hard by measuring how much time it took for the users to complete them, and measure how much time the system spent on executing the tasks the users executed.

An experiment like this can be done by providing several articles on the web to a user and have them read the ones they find most interesting. After a few iterations, we can observe how well the system recommends other articles, based on the data gathered from the previous reads.

2.5.3 Online testing

The testing method that will give us the most realistic and reliable info is online testing. This is where we implement the recommender into the system and deploy it online for real users to test. The scope of data gathered here will be a lot wider than the testing introduced in the two previous chapters, and as we all know: more test-data equals more precise results. One typical way of doing this in the game-industry especially is to have beta-testing. This is a testing-phase where the users are aware of it not being the final iteration of the software, but they participate and are encouraged to report bugs and similar problems that needs to be fixed to the developers. In beta-testing, it is typical to have different builds that goes live after new fixes are implemented, and this gives the developers the possibility to test different functionality and observe how they interact with each other.

Online testing can also be considered a bit risky if it is initialized without proper testing earlier. Users who test a very faulty system online might be discouraged to try the real thing later. Therefore, we suggest online testing to be the last step in a longer testing period, with offline- and user-testing first.

2.5.4 Evaluation metrics and measures

Whatever the subject of evaluation, we need to know which characteristics to measure to be able to determine its quality. The quality of recommendation systems can be measured by looking at several metrics, and in this section, we will present the most important ones, in accordance to Aggarwal's [1] listed goals of recommendation systems. It is important to keep in mind that different systems have unique needs and goals. Depending on these goals, the different metrics plays a big part in some systems, while they are not as important in other.

- The most important measure of a recommendation systems quality is its *accuracy*. Accuracy can be measured in the case of estimating ratings in a system with ordered ratings gathered from explicit feedback, or in a system consisting of unary ratings with implicit user-feedback. Algorithms to compute accuracy are discussed in Section 2.5.5.

- While high accuracy is the main goal of most recommendation systems, it does not always give good *coverage* of items. A systems coverage measures how many items are accessible and recommended, and can be grouped by *user-space coverage* and *item-space coverage*. Respectively, user-space coverage determines how many items can be predicted for a user, and item-spaced coverage how many users an item can be predicted for.

- Measuring *confidence and trust* can be crucial in critical systems. The basis of these measures is whether the system is confident in its recommendations, and if the users trust the systems recommendations.

- *Novelty* and *diversity* are two measures of the same notion, but differs in a few ways. An items novelty refers to its difference from other items viewed by a specific user, while item diversity refers to a systems ability to recommend different items within the same set of recommended items [8].

- *Serendipity* measures the level of surprise in recommendations, which can tell us to what degree a user is able to discover unexpected material from successful recommendations.

- A possibility in many recommendation systems is fake information or ratings. This may be caused by profit-driven motivations, or can happen by accident. To measure how a system deals with this, we evaluate the system's *robustness*. As systems can also evolve over time, we evaluate its *stability,* and see if the recommendations stay consistent.

- With ever-growing data-collections and users of the internet, *scalability* has come one of the two most important measures to keep in mind alongside accuracy. Scalability revolves around both time consumption and space, and is measured by training time, prediction time and memory requirements.

2.5.5 Accuracy measuring algorithms

Depending on what kind of rating data the system is using, the preferred evaluation algorithms differ. While some are best if the system uses implicit data and unary ratings, others are better suited to evaluate explicit ordered ratings.

Precision and Recall

Precision and recall are two important measures in recommendation systems. Precision is the measure of what rate the retrieved items that is relevant to the user, while recall measures the rate of the relevant items that are retrieved [29].

$$Precision = \frac{Relevant\ items\ retrieved}{Retrieved\ items} \tag{1}$$

$$Recall = \frac{Relevant\ items\ retrieved}{Relevant\ items} \tag{2}$$

When we want to use precision and recall, we need to classify each item. These items can be classified as true positive, false negative, false positive and true negative. We get the classification based of if an item is used or and if the items is bought or not. This is shown in Table 5.

	Relevant	Nonrelevant
Retrieved	True positive (TP)	False positive (FP)
Not retrieved	False negative (FN)	True negative (TN)

Table 5: Classification of items [29]

Then after the items have been classified we can measure precision and recall by:

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

$$Recall = \frac{TP}{TP + FN} \tag{4}$$

RMSE & MAE

In Chapter 5, we will run tests on datasets of movie ratings explicit ordered ratings. To compare different algorithms, we use the root-mean-square derivation (RMSE) and mean absolute error (MAE). The MAE is a measure of absolute values of errors to obtain the 'total error', and then dividing the total error by *n* [39], and from this paper, MAE is described with this formula:

$$MAE = \left[n^{-1} \sum_{i=1}^{n} |e_i| \right] \tag{5}$$

RMSE is calculated with getting the 'total square error', as the sum of the individual squared errors. This means that each error influences the total errors in proportion to its square, rather than its magnitude [39]. As a result, large errors have a greater influence on the total square error than the smaller errors. Then total square error is then divided by *n*, and we then get the mean-square error [39]. Finally, we then take the square root of the mean-square error and we get the RMSE. And within the paper [39] we have the formula:

$$RMSE = \left[n^{-1} \sum_{i=1}^{n} |e_i|^2 \right]^{1/2} \quad (6)$$

Mean Average Precision

Mean average precision (MAP) is an evaluation metric commonly used to evaluate the precision in recommendation systems based on unary ratings, and focuses on getting the top *n* recommendations for the user. In "The Million Song Data Challenge" [30], there is a formula of the truncated ranking, described as: "for any $k \leq \tau$, the precision-at-k (P_k) is the proportion of correct recommendations within the top-k of the predicted ranking:"

$$P_k(u, y) = \frac{1}{k} \sum_{j=1}^{k} M_{u,y(j)} \quad (7)$$

"For each user, we now take the average precision at each recall point:"

$$AP(u, y) = \frac{1}{n_u} \sum_{k=1}^{\tau} P_k(u, y) * M_{u,y(j)} \quad (8)$$

"Where n_u is the smaller of τ and the number of positively associated songs for user u. Finally, averaging over all m users, we have the mean average precision:"

$$mAP = \frac{1}{m}\sum_{u} AP(u, y_u) \tag{9}$$

"where y_u is the ranking predicted for user u" (The Million Song Dataset Challenge) [30].

2.6 Summary

In this chapter, the general ideas behind recommendation systems are introduced. By having a well-tailored recommendation system active, benefits for both businesses looking for profit arise, as well as enhancing user experience. While you as a user sometimes know specifically what you are looking for, there are also times you are just browsing. With a system in place that guides you, and which seemingly knows your interests, it can help you discover new items and possibly influence others by giving your own opinion on visited items by leaving a rating.

Loads of research has been done on recommendation systems the last decades, and four of the most common recommendation approaches has been discussed: content-based, collaborative filtering, knowledge-based and demographic-based. What they have in common, is that they all use data from either the users or items, or both, to guide users around in systems with often overwhelming amounts of data. While some of the systems compare either unary or ordered ratings between users to create personalized recommendations, others use domain knowledge and user feedback to achieve the same goal. Each of these approaches has advantages and disadvantages. To counter these, a common approach is to merge them to use the strengths of one to disarm the weaknesses of another. Such merged systems are called hybrid systems, and have several different possibilities for merging, depending on which issues is probable to arise in the related system. Some of the systems that uses these approaches, and most people can relate to, are Netflix, Spotify, Amazon and YouTube.

When evaluation recommendation systems, there are several metrics to take into consideration. Some of them being accuracy, coverage, confidence, trust, novelty, serendipity, diversity, robustness, stability and scalability, with accuracy and scalability often being most important. Depending on the type of rating data represented in the system, either RMSE or MAE is used, or MAP. While MAP is a good measure when evaluating implicit data and a top-*n* recommendation problem, RMSE and MAE is the best measure when dealing with explicit data, often in the form of ordered ratings.

3 Forzify

As we have now covered several recommendation system approaches, we will here try to make use of the gathered information and discuss which approach is best suited for the application Forzify. We will talk about how the application works and what users can expect from the service, as well as considering its current recommender system and what we want to achieve. These conclusions will be made by using what we have learnt about seemingly similar systems, including what data we will have available in Forzify and what features we want included.

3.1 About Forzify

Forzify is a system that provides content in form of football clips and videos on the web. The system we are involved with in this thesis is a lightweight consumer version of the Forzify desktop version [17]. Social media is already a big part of most of our lives, and Forzify aims to build upon this and allow for quick access to football clips through sharing and interacting with friends, and rapid updates during game days. Eventually, this will hopefully generate more interest and social interaction on game days, but also act as a platform where fans can be a part of a football-loving community outside these specific days.

The Forzify application is meant to give users, in form of football supporters, a social experience for sports. To accomplish this, Forzify is built up similarly to Spotify; a system for all your favorite sport events [18]. Some of the features included are the possibility of watching highlights and game summaries, sharing and discussing sport events with your friends, and creating your own events. Currently, Forzify is used by a few Norwegian clubs, namely Vålerenga IF, Viking FK and Tromsø IL. As of now, these versions are stand-alone versions, which means that they are completely separated and you will only be able to watch videos from the respective club depending on which site you have entered. This is about to change, though, as there will soon be one version for the Norwegian top division Eliteserien, alongside the Swedish Superettan and Allsvenskan. These versions will include a lot more clips than the current club based versions, so a good system to keep track of each users' preferences will be needed, and how this can be done to make good recommendations is what we will be discussing in the next sections and chapters.

While we have already made a comparison of Forzify and Spotify, the applications' layout is also similar to that of YouTube, which we can see in Figure 8. The screenshot shows us the front page of the Vålerenga IF version of Forzify, where videos from the latest Vålerenga game is shown in the upper container, and trending videos are shown below. A user can log in, search for videos, create a new playlist, add videos to a playlist or simply browse videos. You can also view your history of videos played, which is important for the system to keep track of in regard to personalized recommending.

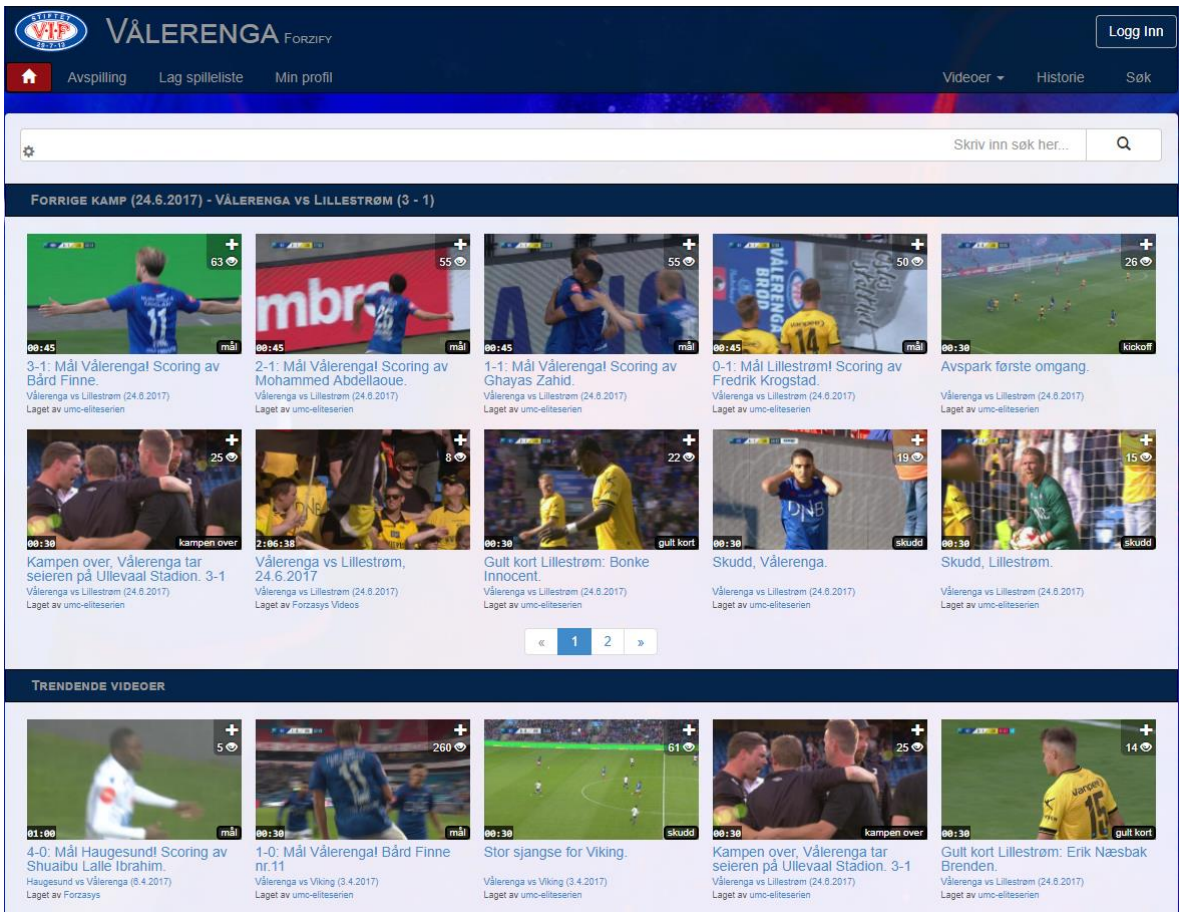


Figure 8: Front page of VIF Forzify

3.2 Data

Before we start discussing what recommender system is best suited for the application, it is crucial to know what data we can work with from the different data sources available in

Forzify. Both the videos and users have properties we can take advantage of, and these are the ones we will look at now.

Videos on Forzify are uploaded and given a title and different tags to describe its content. Unlike text documents on the web, videos do not have the same default properties we can use to differentiate them from one another, which makes such properties invaluable. A few of the tags used as content descriptors in Forzify, are “shot”, “yellow card”, “goal” and “penalty”. While these are the main descriptors of a clip, they also contain meta-data. The meta-data is in place because it is not only relevant for the system to know what happens in the clip, but also what players were involved and which clubs. Why? Because when Forzify contains clips from hundreds of players and several clubs, a user will more than likely want his top recommendations to be goals and other content related to his favorite players and clubs.

To create personalized recommendations, we need user data. As we introduced in the previous section, Forzify keeps track of a user’s interaction history. Even though it might be drastic to conclude that you like a video only because you have watched it, we can at least conclude that you are interested in its content, especially if the user has watched it more than once. However, there is also stored explicit data in form of the possibility to press the like-button on the videos. This, as well as adding videos to a playlist, is a much stronger indicator of a users’ liking of an item, and should be weighted higher than the previous example. What is currently lacking is the possibility to dislike items, and maybe more importantly, rate items. Without such grading of preference, it is not as achievable to predict ratings, which means we will have to predict user preference based on only positive indications. This can also greatly affect how algorithms perform on the system, and will be taken into account when running the evaluations in Chapter 5.

3.3 Current recommendations in Forzify

Currently, the first recommendations users will receive are the ones displayed on the front page as shown in Figure 9. If a user is not logged in however, the same part of the page will be occupied by trending videos instead, as we can see in Figure 8. These recommendations can also be viewed by entering your own recommendation page, which is achieved by clicking the “Recommended”-tab. The second way Forzify gives us recommendations is by showing related videos on the right-hand side of the page when watching a video, which is

very similar to the way YouTube does it. It is important to keep in mind that the current iteration of Forzify is split into versions for each team, and therefore recommendations will always be related to one club only. We will however consider Forzify's recommendation system as if these versions were merged, in regard to further discussions of a suitable approach.

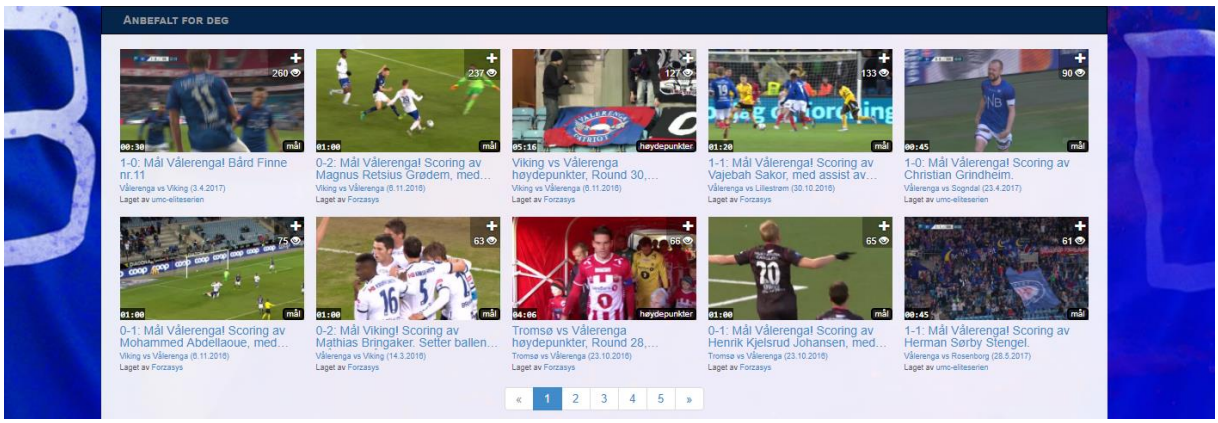


Figure 9: Forzify "Recommended for you"

Forzify currently uses Elasticsearch as a tool to manage information retrieval related to the site's search engine, as well as for creating recommendations. Elasticsearch is a distributed, scalable, real-time search and analytics engine. This engine supports full-text searches, structured searches, analytics and a combination of all three [14]. Elasticsearch is based on Lucene which is an open-source information retrieval software library written in Java, and so is Elasticsearch. Lucene is used for its indexing- and searching-techniques, and Elasticsearch uses Lucene to make full-text searches easy by running a RESTful API, so that the complexity of Lucene is hidden. This makes Elasticsearch easy to use and can be more advanced as the user learns more [14]. Elasticsearch is used by many big co-operations such as: Facebook, GitHub and Netflix, but Elasticsearch is also used on prototypes to make the into scalable solutions. Elasticsearch can run on a laptop or scale over a vast number of servers.

As Forzify currently bases its recommendations on the content of items, the approach in use is content-based, which is only logical given that this the only available data source in the system. As already discussed, such a system is good for handling the cold-start problem, as

well as learning over time to give users better recommendations. However, recommendations can become self-biased, and have a hard time with serendipity. In general, content-based systems also have lower accuracy than collaborating filtering.

3.4 Improving the recommendation system

In a system where content in form of videos will be uploaded often and on a regular basis, users will most likely pay frequent visits to the site to check out new content. In such situations, the system benefits from learning user behavior and their preferences from the respective user profiles, and improve recommendations over time. We also need to take the constant flow of new items introduced to the system into account, and find a way for the system to handle the cold-start problem, especially for new items. Helping users explore and discover new videos should also be one of the main concerns for a system like this, in terms of giving the users a pleasant experience when visiting the application. With a lot of new items, and possibly new users added to the system, we also have to take the systems scalability into consideration.

Considering the data at hand, we can exclude the use of a demographic-based approach to improve the recommendation system. While it can be a good addition in a hybrid setting, the current data source only allows for either content-based or collaborative filtering. For a demographic approach to be feasible, the system needs to get explicit feedback from users about their demographic data, as described in Section 2.2.4. Some users are not willing to share their personal data, which results in below par recommendations. The knowledge-based approach could be a possibility, as it deals with the cold-start problem well, but has no improvement over time by learning preferences. And as stated in Section 2.2.3, it is heavily dependent on expert domain knowledge and engineering, which can be both costly and time consuming.

The main disadvantages to consider if we want to implement collaborative-filtering based recommendations in Forzify, will be cold-start, sparsity and scalability. The latter two can especially become a problem due to the number of items being introduced to the system. With a vast number of users and items, there will be a lot of items that are not rated or interacted with by users, and the rating-matrix will become sparse.

When using a content-based approach, a problem might occur related to how well new items are described for the system. New videos need to be machine-recognizable, as the recommendations are based on the tags and description of the clips. When using a pure content-based approach, words that have the same meaning but are spelled differently can cause problems because the system might recognize these words as independent words, and will not find similarities between them.

At the moment, Forzify collects implicit data in form of a user's interaction history. In addition to this, it also gathers explicit feedback from users, who have the possibility to show their affection for items by clicking the like-button. While this is sufficient to create recommendations, the use of ordered ratings is also a possibility we will take into consideration, when evaluating the different approaches' respective algorithms in Chapter 5. By looking at research done by using MAP to evaluate algorithms on datasets with unary ratings [31], we will evaluate by using RMSE & MAE on some of the same datasets with ordered ratings, and compare the results.

3.5 Summary

In this chapter, we have learned that Forzify is an application created to gather and share content in the form of football-videos to eager football fans. To allow those users to explore this content in an effective and user-friendly way, it is of most importance to have a recommendation system in place. Currently, the system is split into versions for each participating club, but will soon be a platform for all clubs in the leagues represented in the system. We see that there are currently both personalized and non-personalized recommendations implemented, which depends partly on whether the user is logged in or not.

The usable user-data sources in Forzify are currently gathered both implicitly and explicitly. The system keeps track a user's interaction- and search-history, as well as whether the user has added videos to a playlist. The explicit data is gathered by giving users the possibility of liking items, which is a unary form of rating. The main source of data needed to run the current content-based system, is the data about the videos. All videos on the site have descriptors in the form of tags, that allows the system to find similarities between them.

When discussing a suitable approach to use for Forzify, we have taken the current data-sources into account and considered how they can be of further use, while at the same time minimizing the impact of the greatest challenges and problems concerning recommendation systems. We see that a content-based or collaborative filtering approach is the only option given this data, as a demographic-based or knowledge-based approach either demands additional data-sources. We also introduce the idea of implementing an ordered rating system, that allows for rating-prediction, and not only finding the top-n recommendations for the users. We will bring this information further to the next two chapter, where we look for suitable algorithms to use within the content-based and collaborative filtering approaches.

4 Implementation

When we are going to choose an approach to recommend something to a set of users, it is important that we look at what options we have to choose from. When we have an idea of what we want to recommend and what user specifications which are important to us, we can start looking at the algorithms and frameworks that work great for us. There are several different algorithms that we can choose to use, but they have their own strengths and weaknesses. Some algorithms may give fast recommendations, but in return, they are less accurate than their counterparts. There are also the questions of how much money it is worth to use on computer parts for the recommender system.

If we get a close look at the different options we have before we decide what approach we would choose, we can save a lot of money and get a recommender system that works great with our customers. In this chapter, we will look into different algorithms that are used in recommender systems, and also look at some frameworks for using these algorithms. We use the MovieLens dataset to test runtime of some popular algorithms on different frameworks.

4.1 Frameworks

There are several frameworks we can use when we want to evaluate how algorithms perform on different data. We will now present some frameworks that we may want to compare to decide which framework we want to use to evaluate algorithms for Forzify.

4.1.1 Surprise (RecSys)

Surprise is an open source recommender system, which is a Python recommender package [23]. This system has made it easy for the user to create their own implementation of prediction algorithms, while also having a lot of known recommenders already built into the package. Surprise is a package created for students and researchers.

4.1.2 Apache PredictionIO

PredictionIO [15] is a machine learning server, which can be used to create a recommender system. This system is used with Apache Spark, MLlib, HBase, Spray and Elasticsearch. Spark is a large-scale data processing engine that works with the data preparation and input to the algorithm, training and the serving process. HBase is used as the data store, which stores imported events. PredictionIO supports user-based, item-based, item-based cross-action, defined ranking, item-set-based and limited content-based recommendations. The recommender uses the Correlated Cross-Occurrence algorithm to automatically correlate data such as items clicked, terms searched, categories viewed, items shared, people followed, items disliked, gender, age, location and device to make better recommendations.

4.1.3 LensKit

LensKit [12] is a recommender toolkit based on Java. It comes with recommendation algorithms in the category item-based collaborative filtering, user-based collaborative filtering, matrix factorization and slope-one. LensKit has three primary goals, modularity, clarity and efficiency. The modularity in LensKit is designed so that every algorithm and component, can be reconstructed and modified to serve the best needs. Efficiency is important, and LensKit is optimized for clear code over unnecessary optimization, but still have a reasonable efficiency through data structures.

4.2 Algorithms

In this section, we are going to present some popular prediction algorithms and show the equations for some of these algorithms. We are going to run these algorithms on a framework to determine what algorithms perform best on some datasets, and which one is the best option for recommending items on Forzify.

4.2.1 Baseline algorithm

When we want to create a recommender system, that involves users and items, it is important that we can handle new users and the cold start problem as we have discussed in section 2.4.3. Baseline algorithms are non-personalized and do not depend on a user's rating of an item,

which helps when it comes to new users. At the same time, it can be useful for our personalized algorithms so that they can compare data. Baseline algorithms first job is to predict an average rating over all the ratings we have. [13] We set the baseline prediction to $b_{u,i}$, where u is the user's and i is the item, and the most basic understanding we have is $b_{u,i} = \mu$, here μ is the overall average rating. We can also predict the average rating by a user or for a specific item: $b_{u,i} = \bar{r}_u$ or $b_{u,i} = \bar{r}_i$. From this article [13] we can further enhance the baseline predictor by combining the user mean with the average deviation from user mean rating for a particular item:

$$b_{u,i} = \mu + b_u + b_i \tag{10}$$

b_u and b_i are user and item baseline predictors. This baseline can be further regularized as it has been in [13]. What is great for a baseline predictor is that we can assume that when a new user enters the system, he/she is an average user and will get predictions respectively.

4.2.2 Matrix factorization

Matrix factorization [25] is a collaborative filtering technique, which have become popular over the recent years. While collaborative filtering uses user feedback to give recommendations, matrix factorization has the possibility to gather a lot more data than what a user explicit give away, it has the possibility to get data from user purchases, behavior, internet history, searches and mouse movements. So that when a user does not give ratings or feedback, the system can give recommendations from other sources. This is a great strength in matrix factorization, since there will always be users who do not give feedback to all the items [26].

When we use Matrix factorization we start with a set of U users, and a set of I items. The matrix of size |U| x |I| we call R, and contains the ratings the users have given to the items. The latent feature would be discovered now. We then find two metrices, P(|U| x K) and Q(|I| x K) such that their product approximately equals to R is given by:

$$R \approx P \times Q^T = \hat{R} \tag{11}$$

Now the Matrix factorization models map both users and items to a joint latent factor space of dimensionality f, user-item interactions are modeled as inner products in that space. Accordingly, every item *i* is associated with a vector $q_i \in R^f$, and the user *u* is associated with a vector $p_u \in R^f$. The elements of q_i measure the extent of which the item possesses those factors positive or negative for a given item *i*. The resulting dot product $q_i^T p_u$ gets the interaction between user *u* and item *i*, which is denoted by r_{ui} leading to the estimate [5]:

$$\hat{r}_{ui} = q_i{}^T \, p_u \tag{12}$$

The system the regularized squared error on the set of known ratings to learn the factor vectors (P_u and Q_i) as [5]

$$\min_{q^*,p^*} \sum_{(u,i) \in K} (r_{ui} - q_i^T p_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2) \tag{13}$$

Now, K is the set of the (*u,i*) pairs of which r_{ui} is known the training set. The constant λ controls the extent of regularization and is usually determined by cross-validation [5].

Singular value decomposition (SVD) is a popular matrix factorization model, which handle a lot of the problems of collaborative filtering such as scalability, handling of large datasets and the empty matrix fields. How we use Singular value decomposition is complex, but the simple version is that we have a very spare matrix that we want to get the recommendation rankings out of. First, we take that matrix and decompose it into two low-rank matrices which include the user factors and item factors. We can do this by finding the minima or maxima by iteration, which is called the Stochastic Gradient Descent. After this is done we can then try to predict unknown values in our original matrix. SVD decomposes a matrix R into the best lower rank approximation of the original matrix R. SVD decomposes R into two unitary matrices and a diagonal matrix: $R=U\sum V^T$. [4] where R is user ratings matrix, U is the user

"features" matrix, ∑ is the diagonal matrix of singular values, and V^T is the movie "features" matrix[x1] Then to get the lower rank approximation, we keep only the top *k* features from the matrices, which we think of as the *k* most important underlying taste and preference vectors.

4.2.3 K-nearest neighbors(KNN)

KNN is an algorithm which take user information and compares the information on other users, and creates recommendations based on what the nearest neighbors liked or bought. The user information compared might be age, demographic, salary or gender. KNN uses the cosine similarity to find the nearest neighbors; cosine similarity is a similarity computation technique [34] to get the similarities between two items or users. The two items we want to compare is looked at as two vectors, the similarity is measured by computing the cosine of the angle between the two vectors. We can refer to the items as *A* and *B* and denoted by sim(A,B) is given by:

$$sim(A,B) = \cos(A,B) = \frac{A \times B}{||A||_2 ||B||_2} \quad (14)$$

In Table 6, users are compared with each other to determine a cosine similarity. We can see that six items have been rated by 5 users from a range 1-7. We want to give recommendations to user 3, by first finding out which user he/she is most similar to. In the table, and in the formula (14), we can see that the cosine similarity between user 3 and 2 is 0,981. As this is the highest similarity, user 3 will be recommended items which user 2 is also interested in.

$$Cosine(2,3) = \frac{7\times3 + 4\times1 + 3\times1}{\sqrt{7^2 + 4^2 + 3^2} \times \sqrt{3^2 + 1^2 + 1^2}} = 0.981 (15)$$

Item-Id / User-Id	1	2	3	4	5	6	Mean Rating	Cosine(i, 3) (user-user)
1	7	6	7	4	5	4	5.5	0.956
2	6	7	?	4	3	4	4.8	0.981
3	?	3	3	1	1	?	2	1.0
4	1	2	2	3	3	4	2.5	0.789
5	1	?	1	2	3	3	2	0.645

Table 6: User-user similarity computation between user 3 and other users [1]

We will now show you a simple example taken from [38], on how KNN actually work. We start with a picture with yellow circles (YC) and blue squares (BS), we want to find the class of the green star (GS).

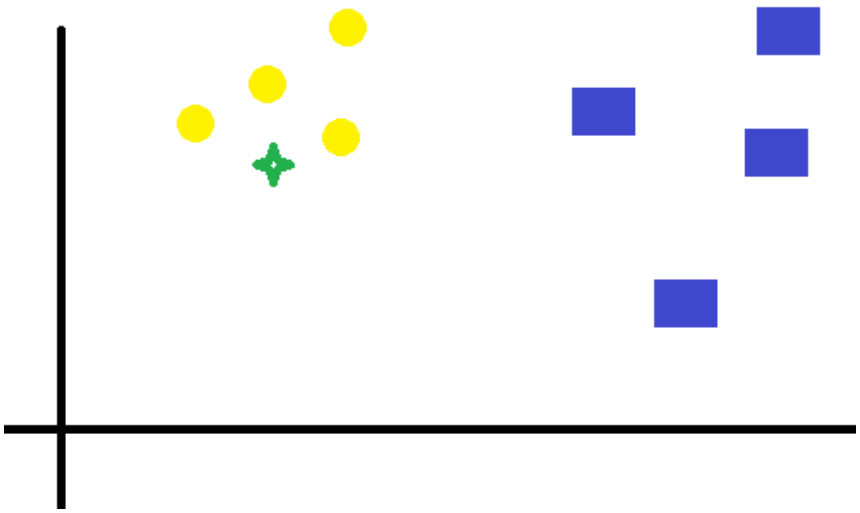


Figure 10: Explanation of KNN with different items

GS can either be YC or BS and nothing else. Let us assume that K = 4, and from that we want to find the 4 nearest neighbors to GS. Now we create a circle with GS at the center and only 4 of the closest points.

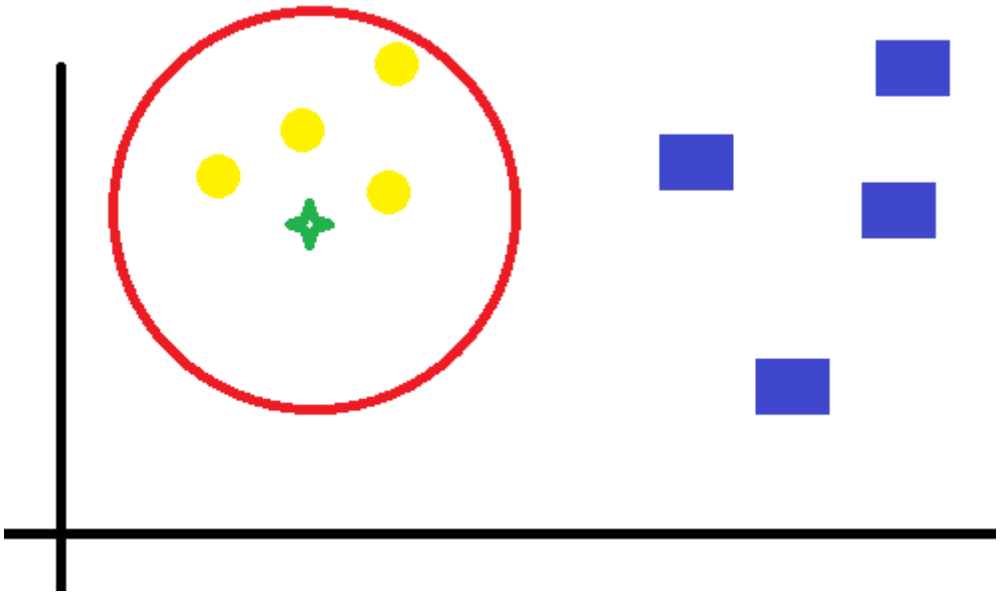


Figure 11: Explanation of the nearest neighbors to GS

We can now see that the four closest points are all YC, and with this we can then assume that GS is in the class YC. This example was very simplified to explain it easy.

4.2.4 Clustering

Clustering is a technique for dimensionality reduction. In a collaborative filtering approach, we can easily end up with vast amounts of data to deal with, and by using something simple as grouping, clustering can help deal with issues related to this. Users are placed in clusters based on related item-information, and recommendations are computed for other users within these clusters.

If we look at Table 7, we can see that customer B, C and D have similar items in interest, especially book 2 and 3, and therefore they are in the same cluster. If a new customer, F, has similar interest in book 3, he will become a “member” of this cluster. This leads to a recommendation of book 2 for this customer.

	Book1	Book2	Book3	Book4	Book5	Book6
Customer A	X			X		
Customer B		X	X		X	
Customer C		X	X			
Customer D		X				X
Customer E	X				X	
Customer F			X		X	

Table 7: Customer interests in books [40]

Co-clustering is an algorithm based on clustering, and now we will give an example of that algorithm which is based of [19]. Users and items are assigned clusters C_u, C_i and some co-clusters C_{ui}. Then the prediction $\hat{r}_{ui}$[22] is set as:

$$\hat{r}_{ui} = \bar{c}_{ui} + (\mu_u - \bar{c}_u) + (\mu_i - \bar{c}_i) \qquad (16)$$

Where $\bar{C}_{ui}$ is the average rating of co-cluster C_{ui}, $\bar{C}_u$ is the average rating of u's cluster, and $\bar{C}_i$ is the average ratings of i's cluster[x4]. If the user is unknown, the prediction is $\hat{r}_{ui} = \mu_i$ If the item is unknown, the prediction is $\hat{r}_{ui} = \mu_u$. If both are unknown, the prediction is $\hat{r}_{ui} = \mu$ [22]. Clusters are assigned using a straightforward optimization method, much like K-means.

4.3 Testing algorithms with frameworks

Below we will run some built-in prediction algorithms on the Surprise and Lenskit frameworks, and get results with the average RMSE and MAE. We have chosen to use RMSE and MAE as a measuring over MAP, as Simen Roste Odden has written a thesis [31] on the same topic and used the MAP for measuring accuracy. This was used because the data that is collected in Forzify is unary and implicit feedback, and since this has already been done, we wanted to do see what algorithm is predicting good if the Forzify data was with user ratings. We evaluate with a 3-folds cross-validation procedure. The dataset consists of 100,000 ratings (1-5) from 943 users on 1682 movies, where every user has rated at least 20 movies. The execution time given is for all three folds, and the MovieLens 100k dataset is used.

K-fold cross validation [35] is used for model evaluation. When it is used it divides the data set into *k* subsets and uses the holdout method *k* times. Every time a *k* subset is used as the test set and the other *k-1* subsets are used to generate a training set. Then the average error

from all the *k* trials is calculated. The holdout method is sometimes referred as the simplest of model evaluations, since this method alone only involves a single run of predicting values in the testing set and then generating the mean absolute test set error.

All the tests are run on a laptop with Intel Core i7-4712MQ, 2.30GHz, 8GB RAM.

4.3.1 Surprise

We will now look at how we can use our algorithms on the Surprise framework and then get a RMSE and MAE measure of the recommendations. We got the walkthrough from Surprise's website [23]. Surprise has a lot of algorithms built into their system, which makes it easy for us to choose what algorithms we use by just editing a couple of code lines. In Code snippet 1 we import the SVD algorithm from Surprise, but we can also choose NormalPredictor, BaselineOnly, KNNBasic, KNWithMeans, KNNBaseline, SVD++, NMF, SlopeOne and CoClustering. It is also possible to create your own algorithms. Surprise also lets us choose between three datasets, Jester, MovieLens 100k and MovieLens 1M. If we want we can also use different datasets, but then we have to create a Reader and define the datasets presets. In Code snippet 1 we use the built in MovieLens 1M dataset, which will automatically download when we run the code. We defined to use a 5-folds cross-validation and set the algorithm to SVD and asks for the measures RMSE and MAE. The results of the different algorithms can we see in Tables 8-17.

```
from surprise import SVD
from surprise import Dataset
from surprise import evaluate, print_perf
import time, os

start = time.time()
#data = Dataset.load_builtin('jester')
data = Dataset.load_builtin('ml-1m')
#data = Dataset.load_builtin('ml-100k')


data.split(n_folds=5)

algo = SVD()

perf = evaluate(algo,data,measures=['RMSE', 'MAE'])

print_perf(perf)
end = time.time()
print(end-start)
```

Code snippet 1: Running the SVD algorithm on the Surprise framework

- **NormalPredictor**
  Algorithm predicting a random rating based on the distribution of the training set, which is assumed to be normal

	Fold 1	Fold 2	Fold 3	Mean
MAE	1.2222	1.2190	1.2165	1.2192
RMSE	1.5189	1.5179	1.5125	1.5164

Table 8: NormalPredictor RMSE and MAE

Execution time: 1.404 seconds

- **BaselineOnly**
  Predicting the baseline estimate for given user and item

	Fold 1	Fold 2	Fold 3	Mean
MAE	0.7515	0.7505	0.7534	0.7518
RMSE	0.9467	0.9480	0.9479	0.9476

Table 9: BaselineOnly RMSE and MAE

Execution time: 1.689

- **KNNBasic**
  A basic collaborative filtering algorithm

	Fold 1	Fold 2	Fold 3	Mean
MAE	0.7795	0.7780	0.7880	0.7818
RMSE	0.9867	0.9861	0.9947	0.9892

Table 10: KNNBasic RMSE and MAE

11.123 seconds

- **KNWithMeans**
  A basic collaborative filtering algorithm, taking into account the mean ratings of each user

	Fold 1	Fold 2	Fold 3	Mean
MAE	0.7552	0.7502	0.7545	0.7533
RMSE	0.9575	0.9526	0.9574	0.9558

Table 11: KNWithMeans RMSE and MAE

Execution time: 11.880 seconds

- **KNNBaseline**
  A basic collaborative filtering algorithm taking into account a baseline rating

	Fold 1	Fold 2	Fold 3	Mean
MAE	0.7393	0.7385	0.7344	0.7374
RMSE	0.9383	0.9379	0.9311	0.9358

Table 12: KNNBaseline RMSE and MAE

Execution time: 13.460 seconds

- **SVD**
  The algorithm that was popularized by Simon Funk during the Netflix Prize

	Fold 1	Fold 2	Fold 3	Mean
MAE	0.7436	0.7424	0.7454	0.7438
RMSE	0.9439	0.9419	0.9447	0.9435

Table 13: SVD RMSE and MAE

Execution time: 10.753 seconds

- **SVD++**
  The SVD algorithm while taking into account implicit ratings

	Fold 1	Fold 2	Fold 3	Mean
MAE	0.7340	0.7309	0.7300	0.7317
RMSE	0.9275	0.9254	0.9263	0.9264

Table 14: SVD++ RMSE and MAE

Execution time: 283.822 seconds (4.73 minutes)

- **NMF**
  A collaborative filtering algorithm based on Non-negative Matrix Factorization

	Fold 1	Fold 2	Fold 3	Mean
MAE	0.7686	0.7641	0.7661	0.7663
RMSE	0.9783	0.9729	0.9744	0.9752

Table 15: NMF RMSE and MAE

Execution time: 10.097 seconds

- **SlopeOne**
  A collaborative filtering algorithm

	Fold 1	Fold 2	Fold 3	Mean
MAE	0.7458	0.7483	0.7471	0.7471
RMSE	0.9483	0.9515	0.9523	0.9507

Table 16: SlopeOne RMSE and MAE

Execution time: 9.995 seconds

- **CoClustering**
  A collaborative filtering algorithm based on co-clustering

	Fold 1	Fold 2	Fold 3	Mean
MAE	0.7705	0.7620	0.7663	0.7663
RMSE	0.9837	0.9723	0.9794	0.9784

Table 17: CoClustering RMSE and MAE

Execution time: 4.301 seconds

4.3.2 LensKit

Now we are going to run some algorithms on Lenskit evaluator. The walkthrough can be found at [11]. We are going to test two algorithms, the personalized mean (PersMean) and the item-item collaboration filtering (ItemItem) algorithm. The PersMean algorithm works by

calculating the user and item average offsets from the global rating. The prediction rule that is implemented is p(u,i) = q + bi + bu, where q is the global mean rating, bi is the difference between the item's mean rating and the global mean, and bu is the mean of the differences between the user's rating for each item and that items mean. We do this by using UserMeanItemScorer, that scores items using a user average, as the ItemScorer, and telling it to use the item mean rating as the offset from which to compute user means which is the UserMeanBaseline.

```
import org.lenskit.baseline.ItemMeanRatingItemScorer
import org.lenskit.baseline.UserMeanBaseline
import org.lenskit.baseline.UserMeanItemScorer

bind ItemScorer to UserMeanItemScorer
bind (UserMeanBaseline, ItemScorer) to ItemMeanRatingItemScorer
```

Code snippet 2: Running the PersMean algorithm on the Lenskit framework

The ItemItem algorithm uses standard item-item collaborative filtering. This is done by using ItemItemScorer as the item scorer implementation. It then normalizes the ratings by subtracting item means prior to computing similarities and scores. This is done by the UserVectorNormalizer, which is configured to subtract a baseline. The baseline then is set to the item mean rating.

```
import org.lenskit.baseline.BaselineScorer
import org.lenskit.baseline.ItemMeanRatingItemScorer
import org.lenskit.knn.item.ItemItemScorer
import org.lenskit.transform.normalize.BaselineSubtractingUserVectorNormalizer
import org.lenskit.transform.normalize.UserVectorNormalizer
bind ItemScorer to ItemItemScorer
bind UserVectorNormalizer to BaselineSubtractingUserVectorNormalizer
within (UserVectorNormalizer) {
bind (BaselineScorer, ItemScorer) to ItemMeanRatingItemScorer
}
```

Code snippet 3: Running the ItemItem algorithm on the Lenskit framework

We then run the evaluator which we got from [11] which ran each algorithm 5 times, since we are using 5-fold cross-validation and we compute the average value of each metric for each data set.

From Table 18, we can see the average RMSE by user and rating, and also get the normalized discounted cumulative gain for both algorithms.

	BuildTime	TestTime	RMSE.ByUser	RMSE.ByRating	Predict.nDCG	MRR
Algorithm						
ItemItem	6.2442	13.4406	0.906771	0.909157	0.953564	0.085192
PersMean	0.2114	1.6990	0.939325	0.952185	0.946713	0.001456

Table 18: ItemItem and PersMean algorithms run on Lenskit

4.3.3 Comparison of frameworks

Frameworks / Property	LensKit	Surprise
Built in algorithms	-	++
Support for implementation of different algorithms	+	+
Support for evaluation	++	+
Built in datasets	+	+
Documentation	+	+
Installation	-	+

Table 19: Comparison of frameworks

From Table 19 we can see that the frameworks do not differ in so many ways. When implementing the algorithms in Chapter 3.4 we noticed that the Surprise had more algorithms already built in than LensKit. Surprise's way of changing algorithms where also easier done than on LensKit. The other thing that where different on these frameworks where that LensKit where more difficult to install and do evaluation on than Surprise. LensKit where better on the evaluation that it came with more metrics which we could compare the algorithms with.

4.4 Summary

In this chapter, we presented three recommendation frameworks, Surprise, PredictionIO and Lenskit, these frameworks are potential algorithm evaluators we could use. Then we presented some popular prediction algorithms and how some of their equations work, such as the baseline algorithm, which is non-personalized and does not need user's rating to recommend items. Matrix factorization is a very popular collaboration filtering technique and the SVD algorithm is based on this. We also introduced K-nearest neighbors, which is a method that takes an items closest neighbors and pairs the up, so that the item gets the same recommendations as the closest neighbors. K-nearest neighbors uses the cosine similarity algorithm to get the similarities from items. Clustering is grouping users in clusters with related item-information, and recommendations is then computed for other users within those clusters, and CoClustering is a prediction algorithm that is within the Surprise framework. Then we looked deeper into the Surprise framework to see how we could evaluate some algorithms on a dataset. We tested algorithms on the Surprise framework with the MovieLens-1M dataset, and we used 5-fold cross-validation when predicting, the results are in Tables 8-17. Then we tested the ItemItem and PersMean algorithms from the Lenskit framework and the results are in Table 18. From these tests, we could determine what framework we wanted to use for our final tests, Surprise has more built in algorithms than Lenskit has, and we felt it was easier to do predictions on Surprise

5 Evaluation

We will in this chapter run five algorithms we have chosen on three datasets, to compare their accuracy and scalability. We will also discuss what algorithms we would prefer for a system as Forzify based on the data we gather. We have chosen to use the RMSE and MAE as accuracy measures, because we want to see what algorithms perform good on a system with user ratingsn while we still know that Forzify now uses unary data. We discussed in section X, that Netflix introduced the thumbs up, thumbs down system, which was received badly by the users and may be badly received by Forzify users. Simen Røste Odden have written a similar thesis [31], but used MAP values and unary data for measures in the evaluation, instead of RMSE, MAE and ordered ratings. We try to find out if the same type of algorithms performs best on datasets consisting of both unary ratings and ordered ratings.

5.1 Results

We will now run a few chosen algorithms on different datasets on the Surprise framework to see how well each algorithm performs with different data. We will get results with the average RMSE and MAE on a 5-folds cross-validation procedure. The algorithms that are run are pre-built from the Surprise framework, such as Baseline, KNNBasic, CoClustering, the SVD algorithm and we will use KNNBasic with a content-based rework. We will use the MovieLens 100k, MovieLens 1m and Jester datasets to compare values on different types of data. The algorithms prediction and training times will be compared with each other, to see what type of scalability they have. All the tests are run on a laptop with Intel Core i7-4712MQ, 2.30GHz, 8GB RAM with Ubuntu as the operating system.

5.1.1 Accuracy

We start by looking at the average RMSE and MAE from running our algorithms on the different datasets. In Figure 15 we have the results from these tests by executing the algorithms on the MovieLens 1M dataset. This is a relatively large dataset and we want the algorithms average RMSE and MAE values to be as low as possible. The ratings from MovieLens are from 1 to 5 and jester is -10 to 10 which is why the average MAE is higher on jester. On the MovieLens 1M dataset, we notice that the algorithms perform somewhat

equally, but there are some differences and we can see that SVD has the lowest RMSE and MAE and KNNBasic Content-based with the highest values. When we look at the RMSE we see that Baseline performs better than CoClustering. This is because large errors are weighted more on RMSE than MAE and that means that CoClustering has more large errors than Baseline has.

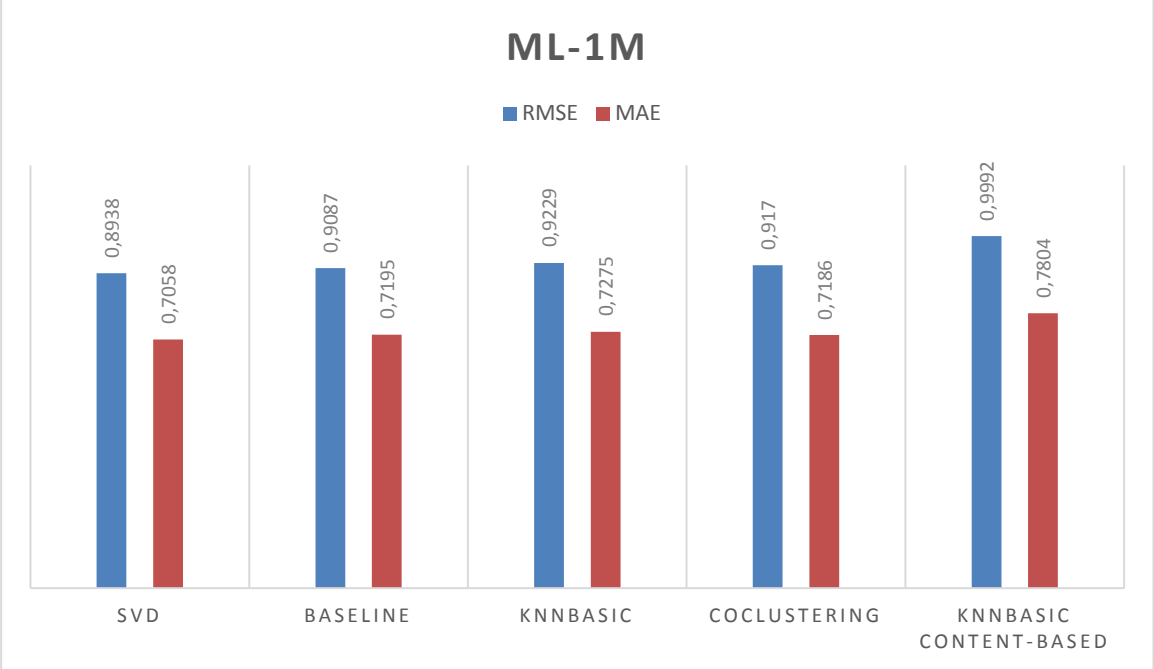


Figure 12: Algorithms run on MovieLens-1M

Now we have run the same algorithms on the MovieLens 100K dataset, and the values have changed, since this is a smaller dataset the algorithms are performing worse. That is because there is less ratings and information for the algorithms to learn patterns from. There is not much change in which algorithm performs better, except for the baseline algorithm. This algorithm is now getting better RMSE and MAE values than the CoClustering algorithm did from the MovieLens-1M dataset, but the rest stays the same. What we learn from this is that a baseline algorithm works better with a smaller learning curve, and this makes sense because the baseline algorithm is supposed to give recommendations when we do not have user ratings. However, overall the algorithms perform somewhat the same.

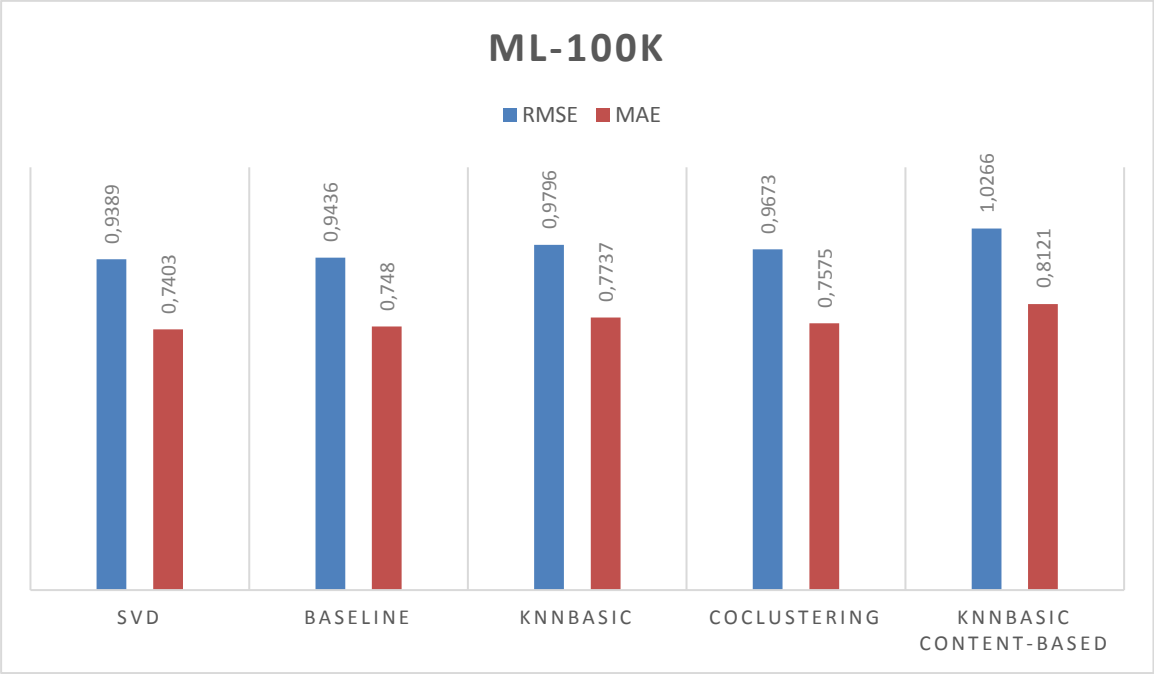


Figure 13: Algorithm run on MovieLens-100K

When we ran the Jester dataset we did not get the KNNBasic to give recommendation because of a memory problem, but we did get KNNBasic Content-based to run. Here the CoClustering is getting the lowest MAE values and baseline is getting the lowest RMSE values. Jester has more users and less items than MovieLens, and again the baseline algorithm is outperforming the others on RMSE values.

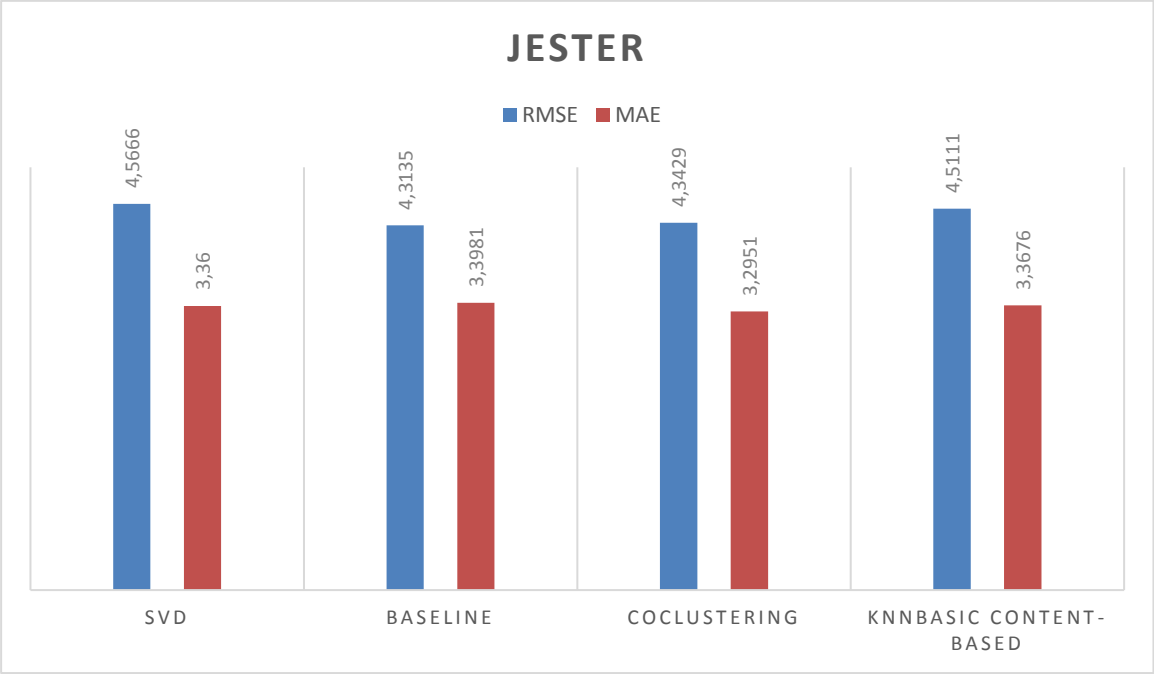


Figure 14: Algorithms run on Jester

In Figure 18 and 19 there are a better view of the average MAE and RMSE of the algorithms on the datasets MovieLens 100K and 1M. Here it is easier to see the comparisons of them and we left the Jester dataset out since it has a different rating system.

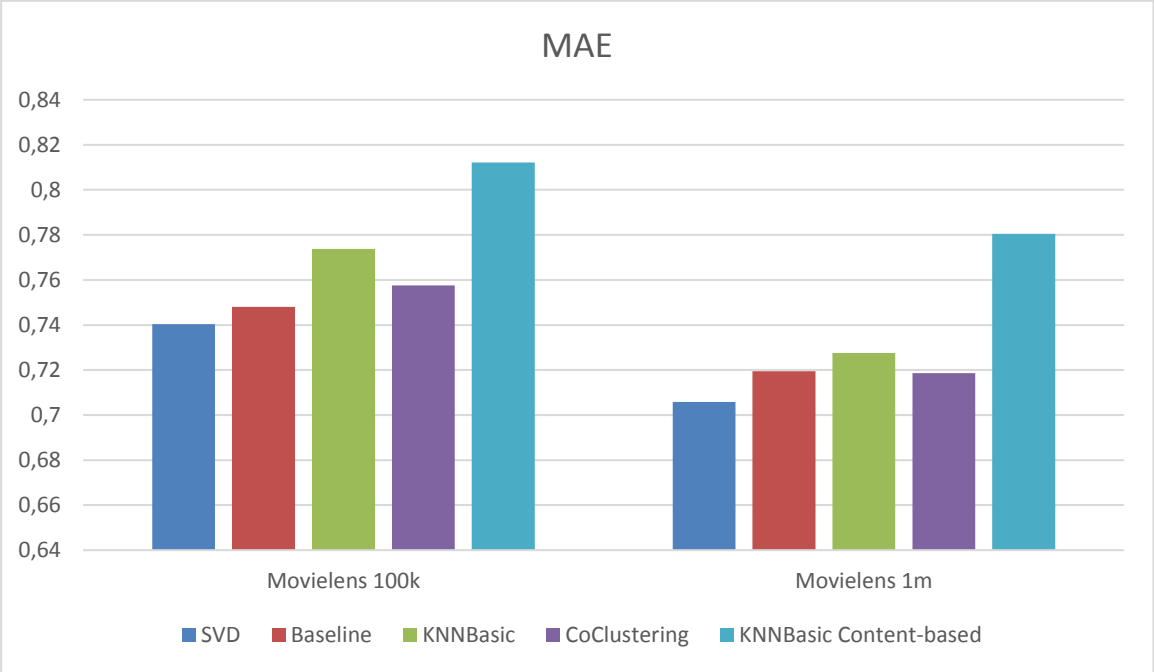


Figure 15: Mean average error of algorithms run on MovieLens-100K and 1M

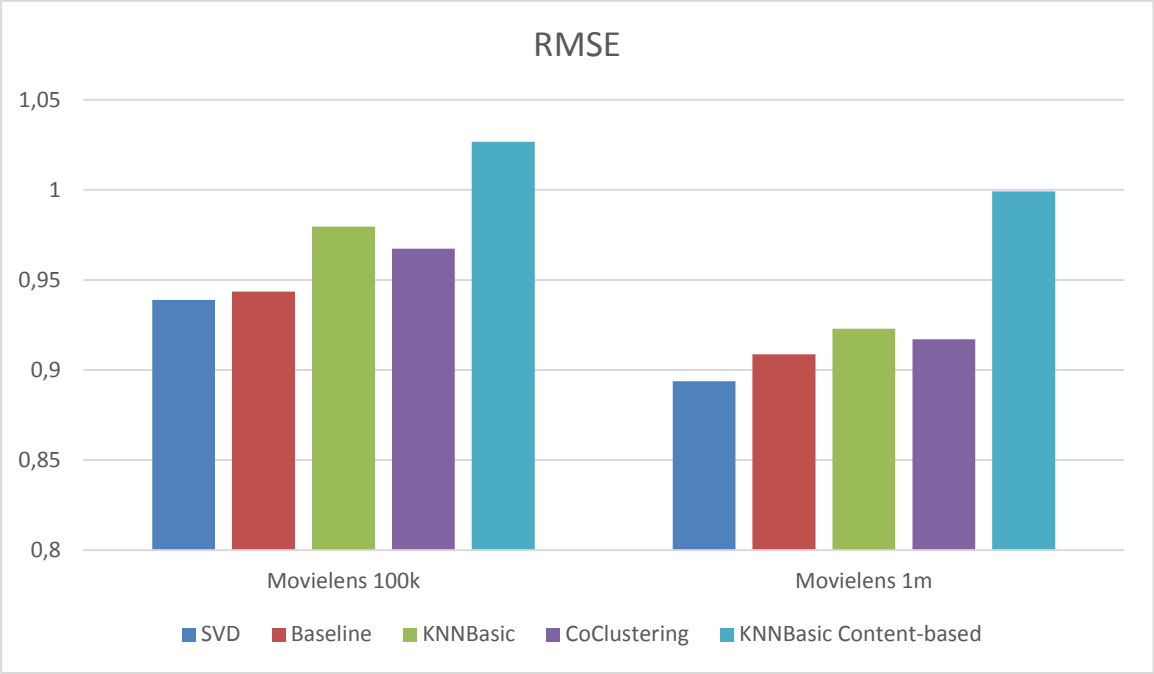


Figure 16: RMSE of algorithms run on MovieLens-100K and 1M

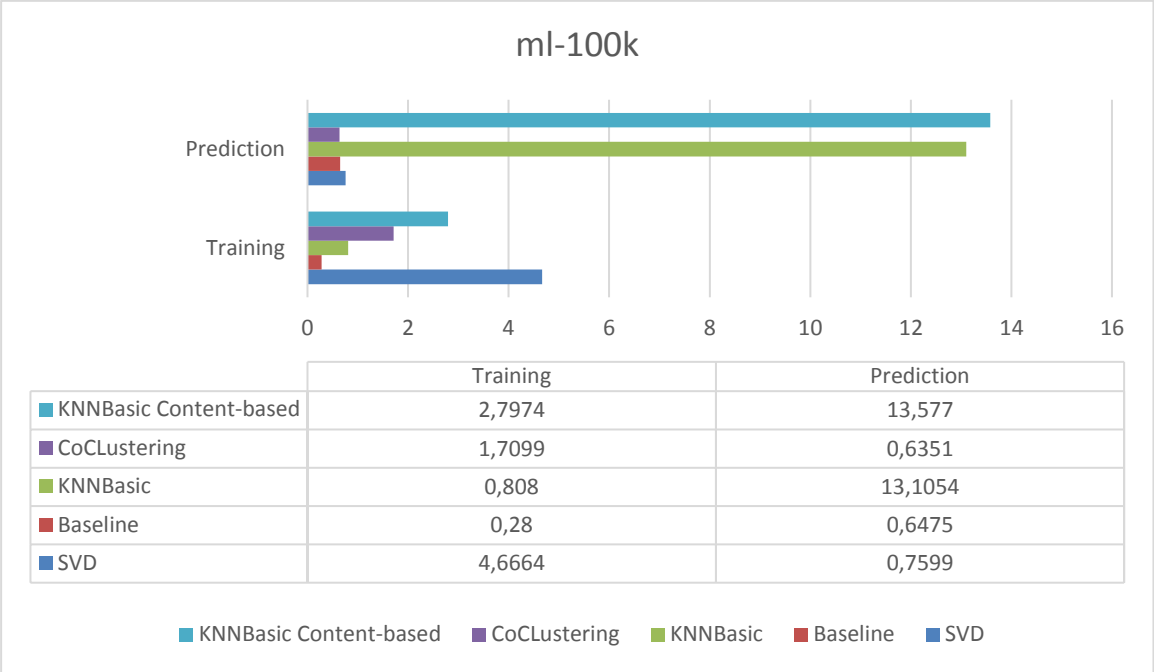
When we look at all the data we have gathered we can see that the SVD algorithm's RMSE and MAE values are better in both MovieLens datasets. However, in the Jester dataset, there is some differences. The CoClustering algorithm has the lowest MAE values, and the baseline algorithm has the lowest RMSE values. This means that these two algorithms perform better on a dataset where there are more users and much less items, than the SVD algorithm. In Simen Odden Røsten's thesis [31], the MAP is being calculated and here the item-based and model-based algorithm performs best. The SVD is a model-based algorithm which makes it good in both cases.

5.1.2 Scalability

In this section, we will consider the scalability of the recommendation algorithms we have done tests on. We will use the same datasets as we have before, and we will take the prediction and training times separately, separately. The purpose of this is that we then can see where the five algorithms differ from one another. The training time is the time it takes the algorithm to prepare the dataset for recommendations, and prediction time is the time it takes to give recommendations.

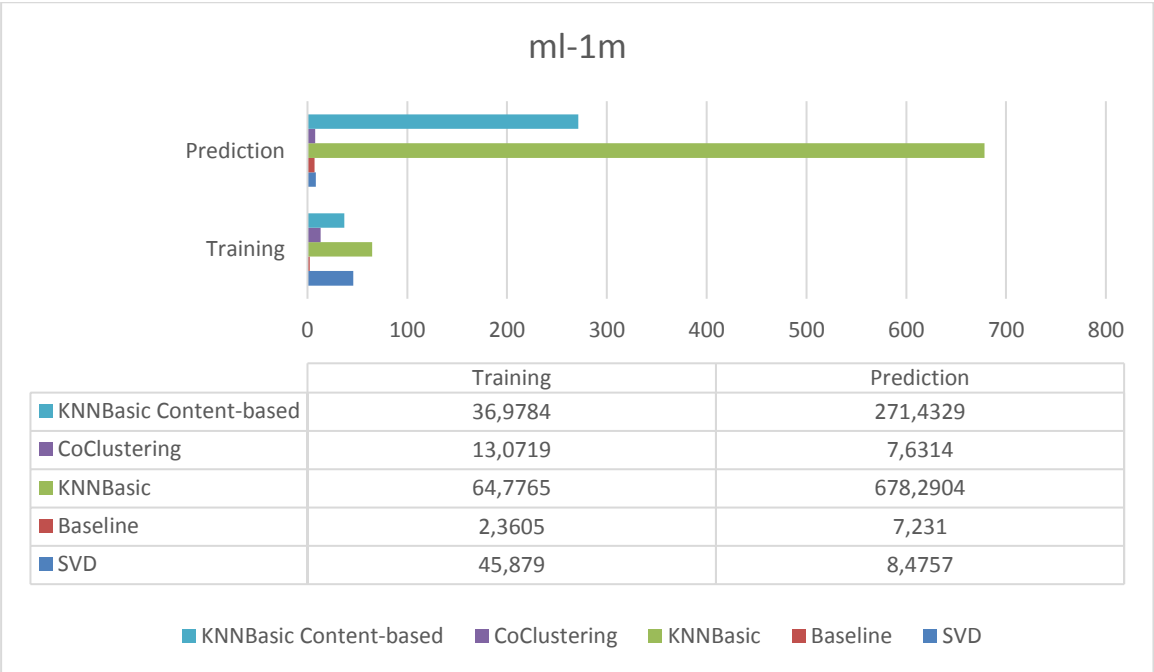
In Figure 20, we see the prediction and training time of the algorithms KNNBasic Content-based, CoClustering, KNNBasic, Baseline and SVD on the dataset MovieLens 100K. The Baseline algorithm uses 0,28 seconds to train its model and is the fastest to do the training, while KNNBasic uses 0,80 seconds. Next is CoClustering which uses 1,70 seconds, then KNNBasic Content-based with 2,79 seconds. The slowest algorithm is the SVD algorithm which uses 4.66 seconds. In Figure 21 we use the dataset MovieLens 1M, and we can see that there are some differences on training times. The Baseline algorithm is still the fastest with 2,36 seconds, and the next algorithm is CoClustering with 13,07 seconds. The third fastest algorithm is KNNBasic content-based with 36,97 seconds, the fourth is the SVD algorithm with 45,87 seconds, and last is KNNBasic with 64.77 seconds. The big difference here is that the CoClustering algorithm performs better on a bigger dataset than KNNBasic, while the rest is somewhat the same in terms of order.

Looking at the prediction times in Figure 20, we see that there are different algorithms that are performing better. The fastest algorithm is CoClustering with 0,63 seconds The performance of the algorithms in order from fastest to slowest is: Coclustering (0.63 seconds), Baseline (0.64 seconds), SVD (0.75 seconds), KNNBasic (13,10 seconds), KNNBasic content-based (13.57). In Figure 21 we can see the prediction times on the dataset MovieLens 1M and here the fastest algorithm is Baseline with 7,23, then CoClustering with 7,63, SVD with 8,47, KNNBasic Content-based uses 271,43 seconds and last is KNNBasic which uses 678,29 seconds. When we look at the different times of the MovieLens-100k and Movielens-1M we see that KNNBasic and KNNBasic content-based scale a lot more than what the other algorithms do.



	Training	Prediction
KNNBasic Content-based	2,7974	13,577
CoCLustering	1,7099	0,6351
KNNBasic	0,808	13,1054
Baseline	0,28	0,6475
SVD	4,6664	0,7599

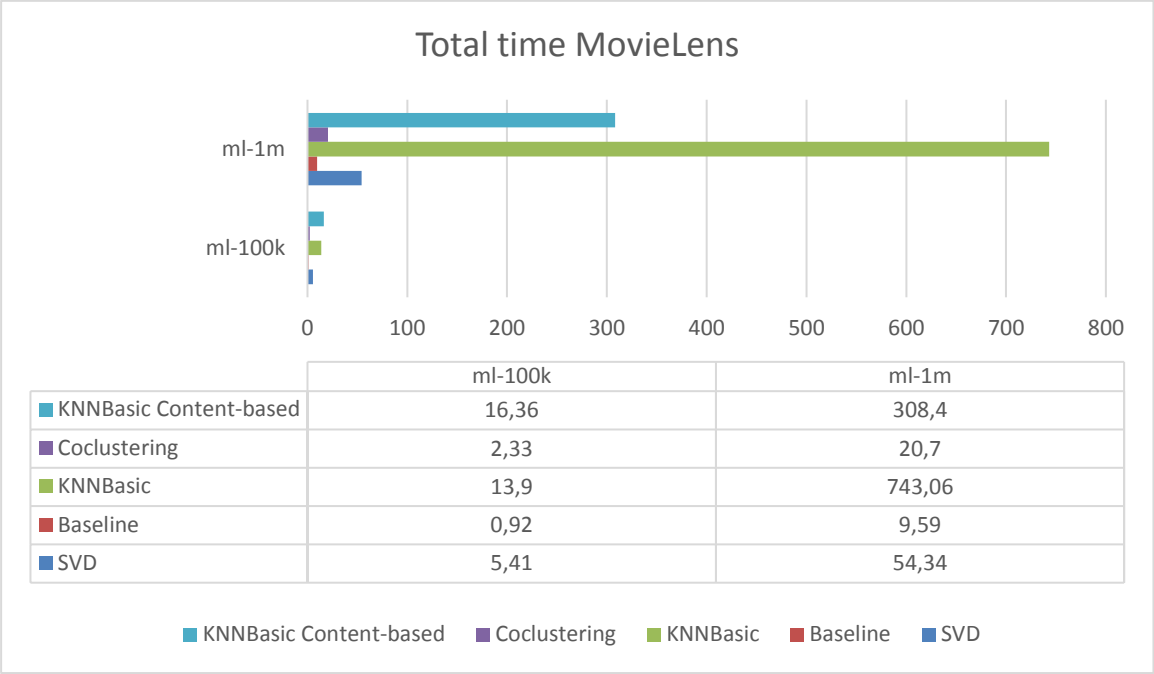
Figure 17: Prediction and training time on dataset ml-100k



	Training	Prediction
KNNBasic Content-based	36,9784	271,4329
CoClustering	13,0719	7,6314
KNNBasic	64,7765	678,2904
Baseline	2,3605	7,231
SVD	45,879	8,4757

Figure 18: Prediction and training time on dataset ml-1m

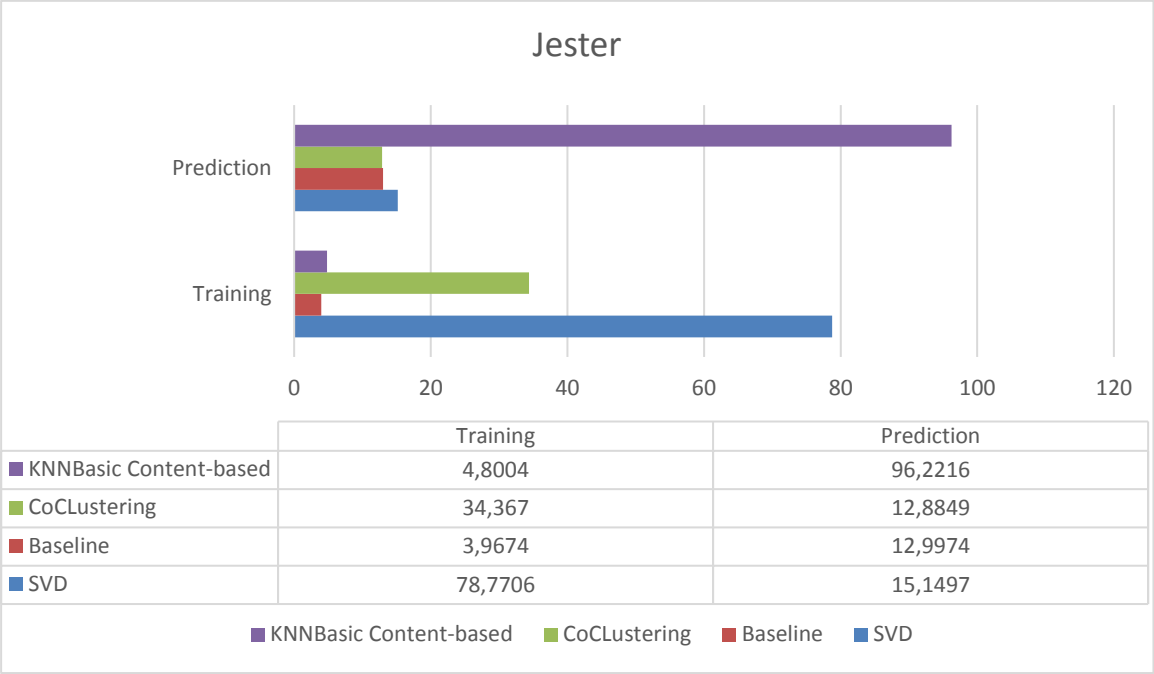
In Figure 22, we have a diagram of the algorithms total time on both MovieLens datasets, and we get a clear picture of which algorithms we should not use when we have a large dataset, such as KNNBasic and KNNBasic Content-based.



	ml-100k	ml-1m
KNNBasic Content-based	16,36	308,4
Coclustering	2,33	20,7
KNNBasic	13,9	743,06
Baseline	0,92	9,59
SVD	5,41	54,34

Figure 19: Total runtime of algorithms on MovieLens

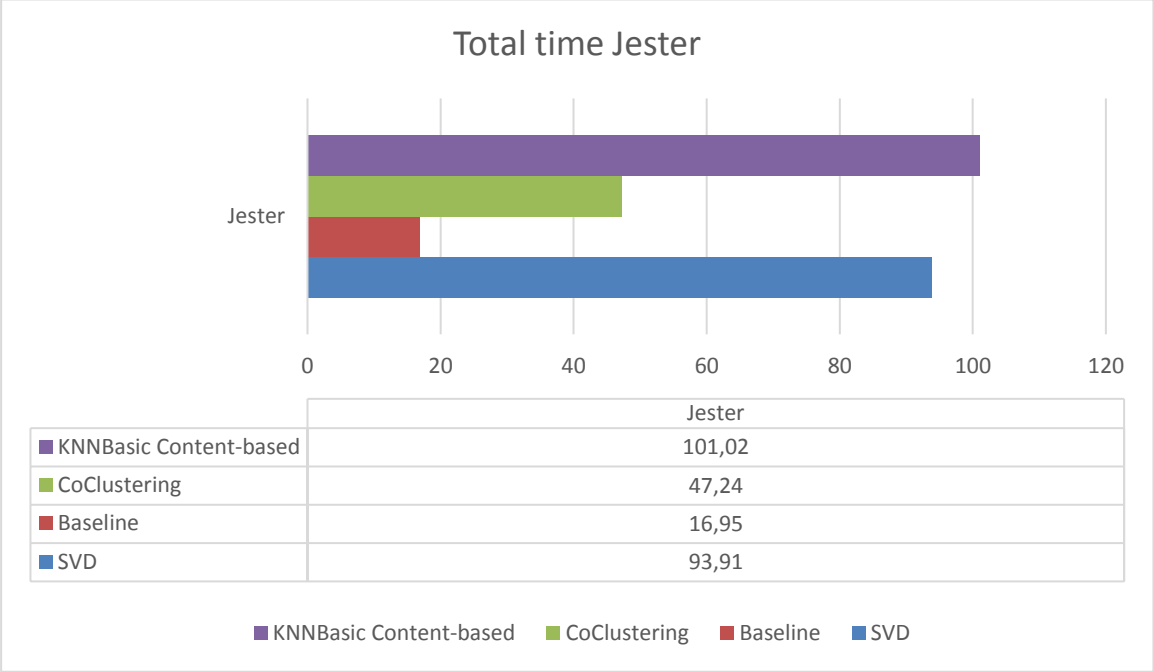
In Figure 23, we see the prediction and training time on the jester dataset. This dataset has more users and ratings, but less items than the MovieLens dataset. The training times are as follows: baseline uses 3.96 seconds, KNNBasic content-based 4,8 seconds, CoClustering 34,36 seconds and SVD 78,77 seconds. The prediction times are CoClustering 12,88 seconds, baseline 12,99 seconds, then SVD 15,14 seconds. The last algorithm is KNNBasic content-based with 96.22 seconds.



	Training	Prediction
KNNBasic Content-based	4,8004	96,2216
CoCLustering	34,367	12,8849
Baseline	3,9674	12,9974
SVD	78,7706	15,1497

Figure 23: Prediction and training time on dataset jester

In Figure 24 we have the total algorithm times on the Jester dataset.



	Jester
KNNBasic Content-based	101,02
CoClustering	47,24
Baseline	16,95
SVD	93,91

Figure 24: Total runtime of algorithms on Jester

In Table 20, we are comparing the data we have gathered from our tests, to see the differences in the algorithms. The algorithms with the most accurate accuracy is SVD and baseline, while

the SVD algorithm is the most accurate. However, the baseline algorithm is faster on both prediction and training time on both MovieLens datasets. The CoClustering algorithm is has good accuracy on a large dataset, and is quick on prediction and training times. The KNNBasic and KNNBasic content-based is to slow to use with a large dataset, and are the two worst algorithms on accuracy as well.

Algorithm / Property	SVD	CoClustering	Baseline	KNNBasic	KNNBasic Content-based
Accuracy on small data	+	-	-	-	-
Accuracy on large data	+	+	+	+	-
Prediction time	+	+	+	-	-
Training time	-	-	+	+	-
Scalability	+	+	+	-	+

Table 20: Summary of our algorithms' properties

The scalability tests we ran, show us that the Baseline algorithm is the fastest on training time on both MovieLens datasets, and the prediction time is fastest on the MovieLens-1M dataset. The CoClustering algorithm is faster on predicting with the MovieLens-100K dataset, which means that the bigger the data gets, the baseline algorithm will still perform faster than CoClustering. On the Jester dataset, the Baseline algorithm is still the fastest on overall time, but is beaten on prediction time by the CoClustering algorithm.

5.2 Best algorithm for Forzify

We will now discuss what algorithms would be best for Forzify if the data we gathered where based on user ratings. To determine what algorithm is best, we will look at the scalability and accuracy of each algorithm.

In the case of accuracy, the SVD algorithm where superior to the other algorithms on both the MovieLens datasets. However, the CoClustering and Baseline algorithms was having better values on the Jester dataset. When we look at the speed of each algorithm on the different datasets, it was clear that the Baseline algorithm where superior on both datasets, except with predicting on the MovieLens-100K dataset, where CoClustering was faster. However, the Baseline algorithm was faster with predicting on the bigger dataset, which we think is more valuable when it comes to a system that will scale upwards with data, like Forzify. To determine one of these three algorithms, is up to what Forzify want most, if they value accuracy over speed or vice versa.

5.3 Summary

We have now looked at how well the 5 chosen algorithms perform on the different datasets. We have done tests on what kind of accuracy they each have, and what kind of prediction and training time each have. We used the average RMSE and MAE from a 5-fold cross-validation procedure to get a good accuracy. We tested the algorithms SVD, Baseline, CoClustering, KNNBasic and KNNBasic content-based. On the datasets MovieLens-1M, MovieLens-100K and Jester, each with a different number of users, items and ratings. We compared our algorithm results and found out that there were three algorithms that where potential for Forzify, SVD, Baseline and CoClustering. The SVD algorithm did give the best accuracy of all the algorithms, Baseline where better than CoClustering in all cases, except for the MAE value in MovieLens-1M dataset. While the Baseline algorithm where the fastest on prediction and training times, on the MovieLens-1M dataset, however it where slower than CoClustering on the smaller dataset MovieLens-100K. The Baseline algorithm gave the worst MAE accuracy on the Jester dataset of the three and CoClustering where the second worst, while again the SVD algorithm gave the best MAE results. For the RMSE values it was opposite, here the Baseline algorithm was scoring best, then Coclustering and last the SVD algorithm. The prediction times on Jester, showed us that the CoClustering algorithm used a lot of time

on predicting, and the SVD was slowest on training. The Baseline algorithm did score best on prediction of the three, and best on training.

6 Conclusion

In this chapter, we will summarize our findings, related to our problem definition in section 1.2. Our main contributions will also be listed, and alongside potential future work.

6.1 Summary

In our problem definition, we questioned how it is possible to find the best recommendation approach for Forzify, when the application does not have sufficient data to be tested. To answer this problem, we research the available recommendation approaches extensively, and look at the following: the content-based, collaborative filtering, demographic-based and knowledge-based. After comparing these in terms of strengths and weaknesses, we investigate how some of them are used in practice by some of the largest actors in the world, which in some cases are similar to Forzify. Further, we look at what data is available in Forzify, which is both explicit and implicit user-data, in addition to content-data. We considered which recommendation approach suits this type of data, and arrived at the conclusion that we should use either the content-based or collaborative filtering approach, or a combination of both. These approaches enable system-learning, which will give better recommendations over time, but can still have the problems with the likes of scalability. We talk about this further when summarizing our evaluation.

To solve our problem of how to evaluate recommendation systems, we investigated different topics regarding testing. We concluded that it has to be done in an offline environment when it cannot be done on a system with barely any user-interaction data. To do this however, we need samples simulating real user- and item-data. Several frameworks were looked into, before deciding to use the Surprise framework. Surprise runs on different datasets, and among them, MovieLens. This dataset is fairly similar to that of Forzify, which made this a viable choice to run further tests on.

Our third problem revolves around finding which approach gives the most accurate recommendation, and which one scales better. To evaluate accuracy, we have used our candidate algorithms on the Jester dataset, and two MovieLens datasets which only differs in size. The output measure for accuracy is RMSE & MAE, and the results gathered concludes that the model-based collaborative filtering algorithm, singular value decomposition (SVD),

generally scores best. However, the measures RMSE & MAE provides best accuracy for datasets with user ratings in form of ordered ratings, which is not currently the case for Forzify. We believe that an ordered rating system could be applied to Forzify, and refer to Odden's evaluations [31], who has used MAP to measure accuracy in a modified MovieLens dataset using unary ratings. We can from both our own and Odden's data, conclude that a model-based approach results in the highest accuracy for recommendations on the tested data.

To measure the scalability of the different approaches, we look at the training time and prediction time of the algorithms. Mainly, we have looked at the execution time of our algorithms on the different sized MovieLens datasets, the 100k version and 1m version, as well as on the Jester dataset. Overall, we can see from our results that the baseline algorithm is fastest on all three datasets. While the model-based approach uses more time during the training-phase, it predicts relatively fast, and as we have concluded, it is the most accurate approach. Therefore, we recommend using a model-based approach for Forzify.

6.2 Main contributions

In this thesis, we have gathered information from available research done on recommendation systems, to figure out which approach is most applicable to Forzify. By comparing the different approaches, we learn their strengths and weaknesses, how they use different data sources to their advantage, but also which problems they can introduce. With this knowledge, we consider the Forzify application, in terms of what recommendation system it currently operates with, and what data we have available.

Because of the lack of user-data in Forzify, we look at a few different datasets we can use to simulate this data. To evaluate our candidate algorithms, we take advantage of some of the publicly available recommendation frameworks. The most important dimensions to measure in recommendation systems are accuracy and scalability, and in our evaluation, we provide results measuring these.

6.3 Future work

In this thesis, we have found a suitable recommendation approach for Forzify, and we have looked at the differences in accuracy and scalability for recommendation approaches across different datasets.

The most suitable way of finding the best approach for Forzify, is to test recommendation approaches on Forzify's data. This can be done with either offline evaluation or online evaluation, in this thesis we used offline evaluation, because we did not have sufficient user-data from Forzify. It would be very valuable to do accuracy and scalability tests with the chosen algorithm in this thesis, with gathered data from Forzify. With more time, we could have tested the chosen algorithms on even bigger datasets, to further see how well each algorithm scaled.

With more time and manpower, it would have been interesting to gather more algorithms and tune them better to the needs of Forzify, so that we may have come up with a solution that superior in both scalability and accuracy. It could also have been interesting to see what differences in data we would have, if Forzify did a rework on their data, so that it would use order ratings instead of only unary ratings.

References

[1]

Aggarwal, C. C. (2016) *Recommender Systems: The Textbook.* Springer. DOI: 10.1007/978-3-319-29659-3

[2]

Alexander, J. (2007, 7, april). *Netflix's new thumbs up, thumbs down rating system is not going over well*. Retrieved from https://www.polygon.com/2017/4/7/15212718/netflix-rating-system-disapproval

[3]

Anand, R., & Ullman, J. D. (2011). *Mining of Massive Datasets.* 307-341 Cambridge: Cambridge University Press.

[4]

Becker, N. (2016). *Matrix Factorization for Movie Recommendations in Python.* Retrieved from https://beckernick.github.io/matrix-factorization-recommender/

[5]

Bokde, D. K., Girase, S., & Mukhopadhyay, D. (2014). *Role of Matrix Factorization Model in Collaborative Filtering Algorithm: A Survey. 1*(6) Retrieved from https://arxiv.org/ftp/arxiv/papers/1503/1503.07475.pdf

[6]

Breese, J. S., Heckerman, D., & Kadie, C. *Empirical Analysis of Predictive Algorithms for Collaborative Filtering.* 43-52. Retrieved from https://ai2-s2-pdfs.s3.amazonaws.com/0fcc/45600283abca12ea2f422e3fb2575f4c7fc0.pdf

[7]

Burke, R. (2002) *Hybrid Recommender Systems: Survey and Experiments. 12*(4). 331-370 DOI: 10.1023/A:1021240730564

[8]

Castells, P., Vargas, S., & Wang, J. *Novelty and Diversity Metrics for Recommender Systems: Choice, Discovery and Relevance.* Department of Computer Science, University College London. Retrieved from http://ir.ii.uam.es/rim3/publications/ddr11.pdf

[9]

Denning, P., Comer, D. E., Gries, D., Mulder, M. C., Tucker, A. B., Turner, A. J., & Young, P. R. (1989). *Computing as a discipline*. Communications of the ACM, *32*(1), 9-23.

[10]

Deshpande, M., & Karypis, G. (2004). Item-based top-*N* recommendation algorithms. *ACM Transactions on Information Systems, 22*(1), 143-146.
DOI: 10.1145/963770.96377

[11]

Ekstrand, M. D., Ludwig, M., Konstan, J. A., & Riedl, J. T. (2011). *Evaluation Scripts*.
Retrieved from
http://lenskit.org/documentation/evaluator/walkthrough/
DOI = 10.1145/2043932.2043958

[12]

Ekstrand, M. D., Ludwig, M. Konstan, J. A., & Riedl, J. T. *Rethinking the Recommender Research Ecosystem: Reproducibility, Openess, And LensKit.*
Retrieved from http://files.grouplens.org/papers/p133-ekstrand.pdf

[13]

Ekstrand, M. D., Riedl, J. T., Konstan, J. A. (2010). *Collaborative Filtering Recommender Systems.* 82-164
DOI: 10.1561/1100000009
Retrieved from http://herbrete.vvv.enseirb-matmeca.fr/IR/CF_Recsys_Survey.pdf

[14]

Elasticsearch. (2017). *Getting started.* Accessed January 19. 2017
https://www.elastic.co/guide/en/elasticsearch/guide/current/getting-started.html.

[15]

Engine Template Gallery. PredictionIO

Retrieved from http://predictionio.incubator.apache.org/gallery/template-gallery/.

[16]

Felfernig, A., Jeran, M., Ninaus, G., Reinfrank, F., Reiterer, S., & Stettinger, M. Basic Approaches in Recommendation Systems.1-15.
DOI: 10.1007/978-3-642-45135-5_2

[17]

ForzaSys. 2017. “Forzify APP” accessed July 12, 2017. Retrieved from
http://home.forzasys.com/docs/forzify-app/

[18]

ForzaSys. 2017. “Forzify” accessed July 17, 2017. Retrieved from
http://home.forzasys.com/products/forzify/

[19]

George, T., & Merugu, S. (2005) *A Scalable Collaborative Filtering Framework Based on Co-clustering.*
DOI: 10.1109/ICDM.2005.14

[20]
Gielen, M., Rosen, J. (2016). *Reverse Engineering The YouTube Algorithm*.
Retrieved from
http://www.tubefilter.com/2016/06/23/reverse-engineering-youtube-algorithm/

[21]
Gomez-uribe, C. & Hunt, N. *The Netflix Recommender System: Algorithms, Business Value, and Innovation.*

[22]

Hug, N. (2015). *Co-Clustering.*
Retrieved from
http://surprise.readthedocs.io/en/stable/co_clustering.html#surprise.prediction_algorithms.co_clustering.CoClustering

[23]
Hug, N. (2017). *Surprise, a Python library for recommender systems*.
Retrieved fromhttp://surpriselib.com
"Surprise"
Retrieved from
http://surprise.readthedocs.io/en/stable/index.html

[24]

Jannach, D. (2011). Recommender Systems: an Introdictin. (Chapter 4) Cambridge: Cambridge University Press

[25]
Koren, Y. (2009) *Matrox factorization techniques for recommender systems*.
Retrieved from https://datajobs.com/data-science-repo/Recommender-Systems%5BNetflix%5D.pdf

[26]

Kumar, B. D., Sheetal, G., & Debajyoti, M. *Role of Matrix Factorization Model in Collaborative Filtering Algorithm: A Survey.* International Journal of Advance Foundation and Research in Computer(IJAFRC) Volume 1, Issue 6, May 2014
Retrieved from https://arxiv.org/ftp/arxiv/papers/1503/1503.07475.pdf

[27]

Kwasi Studios. *Email marketing 2.0 : How Amazon's campaign was almost amazing.* Retrieved from http://www.kwasistudios.com/resources/email-marketing-amazon-case-study/

[28]

Madhukar, M. (2014). *Challenges & Limitation in Recommender Systems*. (138-142) Retrieved from http://www.ijltet.org/wp-content/uploads/2014/10/21.pdf

[29]

Manning, C. D., Raghavan, P., & Schütze, H. (2009) *An Introduction to Information Retrieval.* Cambridge: Cambridge University Press. Retrieved from https://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf

[30]

McFee, B., Bertin-Mahieux, T., Ellis, D. P. W., Lanckriet, G. R. G. (2012). The Million Song Dataset Challenge. *Proceedings of the 21st International Conference on World Wide Web.* DOI: 10.1145/2187980.2188222

[31]

Odden, S. R. (2017). *Recommendation Systems for Sports Videos.* Retrieved from http://home.ifi.uio.no/paalh/students/SimenRosteOdden.pdf

[32]

Pasick, A. (2015) *The Magic That Makes Spotify's Discover Weekly Playlists so Damn Good. Quartz*. Accessed December 15, 2016. http://qz.com/571007/the-magic-that-makes-spotifys-discover-weekly-playlists-so-damn-good/

[33]

Ricci, F., Rokach, L., Shapira, B., & Kantor, P. B. *Recommender Systems Handbook.* Boston: Springer US.
DOI: 10.1007/978-0-387-85820-3

[34]

Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2001). Item-Based Collaborative Filtering Recommendation Algorithms. *GroupLens Research Group/Army HPC Reasearch Center Department of Computer Science and Engineering University of Minnesota.* Retrieved from http://files.grouplens.org/papers/www10_sarwar.pdf

[35]

Schneider, J. (1997). *Cross Validation.*
Retrieved from http://www.cs.cmu.edu/~schneide/tut5/node42.html

[36]

Shani, G., & Gunawardana, A. (2016). *Evaluating Recommendation Systems.* Accessed December 16, 2016. https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/EvaluationMetrics.TR_.pdf

[37]

Shchutskaya, V. (2016). *Approaching the Cold Start Problem in Recommender Systems.* https://indatalabs.com/blog/data-science/cold-start-problem-in-recommender-systems

[38]

Srivastava, T. (2014). *Introduction to k-nearest neighbors: Simplified.*
Retrieved from
https://www.analyticsvidhya.com/blog/2014/10/introduction-k-neighbours-algorithm-clustering/.

[39]

Willmott, C. J., & Matsuura, K. (2005). *Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance.* (30) 79-82. Retrieved from http://www.int-res.com/articles/cr2005/30/c030p079.pdf

[40]

Xavier, A. *Recommender Systems* (Machine Learning Summer School 2014 @ CMU)
Retrieved from
http://technocalifornia.blogspot.no/2014/08/introduction-to-recommender-systems-4.html