



Norwegian University of  
Science and Technology

# A Context Aware Recommendation System for movies

**Tan Quach Le**

Master of Science in Computer Science

Submission date: January 2017

Supervisor: John Krogstie, IDI

Norwegian University of Science and Technology  
Department of Computer Science



TDT4900 Computer Science, Master Thesis

Autumn 2016

**A Context Aware  
Recommendation System  
for movies**

**Tan Quach Le**

Supervisor: John Krogstie

# Abstract

People like to watch movies, but they do not always know what they want to see. Although there exist applications that help people solve this issue, they are not being used very often (see Appendix A). People tend to do their own research to decide what to watch. This thesis will be looking into this issue and provide a proof of concept prototype. Not only does the prototype recommend movies, but it also introduces a social feature that can be extended to so much more. This feature was found by doing a round of interviews, while trying to find the cause of why people do not use the recommendation systems.

The thesis describes some of the state of the art approaches to recommendation system algorithms and the implementation of the prototype. The prototype was evaluated by its accuracy and speed using data sets from MovieLens.

The results show that the chosen approach to recommendation systems is viable, and that the accuracy of the recommendation is not the only reason that recommendation systems are not being used that much. The results show that a social feature might increase the use of recommendation systems.

# Sammendrag

Folk liker å se på filmer, men de vet ikke alltid hva de vil se på. Selv om det finnes programmer som hjelper folk til å løse dette problemet, blir de ikke brukt veldig ofte (se vedlegg A). Folk har en tendens til å gjøre sine egne undersøkelser for å finne en film å se. Denne oppgaven skal se nærmere på denne saken og gi et "proof of concept" prototype. Ikke bare anbefaler prototypen filmer, men det innfører også en sosial funksjon som kan utvides til så mye mer. Denne funksjonen ble funnet ved å gjøre en runde med intervjuer, mens det ble prøvd å finne årsaken til hvorfor folk ikke bruker anbefalingssystemer.

Masteroppgaven beskriver noen av "state of the art" metoder for anbefalingsalgoritmer og implementering av prototypen. Prototypen ble evaluert av sin nøyaktighet og hastighet ved hjelp av datasett fra Moviens.

Resultatene viser at den valgte algoritmen for anbefaling system er brukbar, og at nøyaktigheten av anbefalingen er ikke den eneste grunnen til at anbefalingssystemer ikke blir brukt så mye. Resultatene viser at en sosial funksjon kan øke bruken av anbefalingssystemer.

# Preface

This is a master thesis and it is my last report as a master student. It has been written to complete my Master of Science (MSc) degree in Computer Science at the Department of Computer and Information Science (IDI) at the Norwegian University of Science and Technology (NTNU).

I would like to thank my supervisor John Krogstie for his guidance throughout my journey of the last twenty weeks.

Tan Quach Le

# Contents

<b>Abstract</b>	<b>4</b>
<b>1 Introduction</b>	<b>12</b>
1.1 Motivation . . . . .	12
1.2 Previous work . . . . .	13
1.3 Contribution . . . . .	13
1.4 Report structure . . . . .	14
<b>2 Background</b>	<b>15</b>
2.1 Recommendation systems . . . . .	15
2.2 Techniques . . . . .	15
2.2.1 Content based filtering . . . . .	15
2.2.2 Collaborative filtering . . . . .	18
2.2.3 Challenges with Collaborative Filtering . . . . .	25
2.2.4 Hybrid . . . . .	26
2.3 Context-aware Recommendation Systems . . . . .	27
<b>3 Research method</b>	<b>28</b>
3.1 Design Science Research . . . . .	28
3.1.1 Design as an Artifact . . . . .	30
3.1.2 Problem relevance . . . . .	30
3.1.3 Design Evaluation . . . . .	30
3.1.4 Research Contribution . . . . .	30

3.1.5	Research Rigor . . . . .	30
3.1.6	Design as a Search Process . . . . .	31
3.1.7	Communication of Research . . . . .	31
3.2	Evaluation tools . . . . .	31
3.2.1	Mean Squared Error . . . . .	31
3.2.2	Python time module . . . . .	31
3.3	Evaluation plan . . . . .	32
<b>4</b>	<b>Architecture and Implementation</b>	<b>33</b>
4.1	Data set . . . . .	33
4.2	Chosen collaborative filtering method . . . . .	34
4.3	Phase one . . . . .	34
4.3.1	Scenarios: . . . . .	35
4.3.2	Requirements: . . . . .	35
4.4	Phase two . . . . .	35
4.4.1	Scenarios: . . . . .	35
4.4.2	Requirements: . . . . .	36
4.5	Architecture . . . . .	36
4.5.1	Model-View-Controller . . . . .	37
4.5.2	Back End . . . . .	38
4.5.3	Front End . . . . .	40
4.6	Implementation approach . . . . .	41
4.6.1	Setup . . . . .	41
4.6.2	Defining context . . . . .	45
4.6.3	Genre . . . . .	46
4.6.4	Time . . . . .	47
4.6.5	Recommendations . . . . .	48
4.7	Prototype . . . . .	53
4.7.1	API . . . . .	53
4.7.2	GUI . . . . .	54
4.8	Technology . . . . .	59



<i>CONTENTS</i>	7
4.8.1 PostgreSQL . . . . .	59
4.8.2 GitHub . . . . .	60
4.8.3 scikit-learn . . . . .	60
<b>5 Evaluation</b>	<b>61</b>
5.1 Phase one: The interviews . . . . .	61
5.1.1 Anonymous A . . . . .	62
5.1.2 Anonymous B . . . . .	62
5.1.3 Anonymous C . . . . .	62
5.1.4 Anonymous D . . . . .	62
5.1.5 Anonymous E . . . . .	62
5.1.6 Anonymous F . . . . .	62
5.1.7 Anonymous G . . . . .	63
5.1.8 Conclusion . . . . .	63
5.2 Phase two: Performance . . . . .	64
5.2.1 Data set . . . . .	64
5.2.2 Environment . . . . .	64
5.2.3 User-Based Filtering based on Pearson Correlation . . . . .	65
5.2.4 Factorization Machine with SGD . . . . .	65
<b>6 Discussion</b>	<b>67</b>
6.1 Data set . . . . .	67
6.2 Factorization machine . . . . .	68
6.3 Context . . . . .	69
6.3.1 Genre . . . . .	69
6.3.2 Time . . . . .	69
<b>7 Conclusion and future work</b>	<b>70</b>
7.1 Conclusion . . . . .	70
7.2 Future work . . . . .	71
<b>Appendices</b>	<b>73</b>

<b>A Interviews</b>	<b>74</b>
<b>B Code</b>	<b>80</b>
B.1 Data set Usage License . . . . .	80
<b>C API documentation</b>	<b>82</b>
<b>Bibliography</b>	<b>86</b>

# List of Figures

- 2.1 Illustration of content-based filtering . . . . . 17
- 2.2 Example of feature vectors taken from [26]. Each row represents a feature vector  $x^i$  with its corresponding target  $y^i$ . The first row represents user A who has rated movie TI, has also rated NH and SW, no last ratings and the rating of movie TI was 5. As one may see, there are multiple features leading up to the rating of 5 and not only who rated what movie what rating like previous approaches. This matrix can be used as input to train the model. 24
- 3.1 Design-Science Research Guidelines, figure taken from [16]. . . . . 29
- 4.1 Illustration of overall architecture . . . . . 37
- 4.2 Caption for LOF . . . . . 38
- 4.3 Back end structure. There are individual classes for model and view, which acts like model and controller in MVC. The templates folder would have contained html files for rendering purposes if we wanted Django to do the rendering process (view in MVC). . . 40
- 4.4 Front end structure. Files ending with -view.js acts like controllers, -view.hbs acts like view and -model.js acts like model in MVC. . . . . 41
- 4.5 Database structure . . . . . 45
- 4.6 Django admin interface . . . . . 45

4.7	Dataset in a csv (Comma-separated values ) format. The first value represents userID, second value represents moveID, the third value represents the rating value and the last value represents the timestamp. . . . .	49
4.8	Feature vector representation of the data set. This kind matrix has been used as input for training the FM model. . . . .	50
4.9	Example of browsable API . . . . .	53
4.10	Login page . . . . .	54
4.11	Rating page . . . . .	55
4.12	Rating San Andreas . . . . .	55
4.13	Recommendation page . . . . .	56
4.14	Details of a recommended movie . . . . .	57
4.15	Current user profile . . . . .	58
4.16	This page lets you edit your profile, including adding and removing genres you like . . . . .	58
4.17	Search of user bob. We can see what movies he has rated and other information about him. User tanle and user bob has a similarity of 5 (between -10 to 10, calculated using the Pearson Correlation . . . . .	59
5.1	Performance of user based filtering . . . . .	65
5.2	Performance of Factorization Machine . . . . .	66

# List of Tables

# Chapter 1

## Introduction

A Context Aware Recommendation System is a system that recommends a certain item to a user of that system where not only the user's taste is considered, but also the context in that moment. The context could be anything from the mood of the user to the amount of effort needed to get that item. There are many approaches to recommend items [23]. There is no single best algorithm for a recommendation system. There are many approaches available, but a good recommendation system uses many different algorithms and finds a good combination of those to increase the different strengths and decrease the different weaknesses of each algorithm. In this thesis we will discuss different approaches and algorithms to recommend an item. Based on this, a prototype that recommends movies to a user will be created and evaluated.

This chapter is about the motivation for the research, the problems this thesis is going to discuss and finally the structure of the report.

### 1.1 Motivation

Recommendation systems have been an important research area since the mid-1990s [7]. Many recommendation systems already exist. You can find them in

Netflix, Spotify, Ebay, Amazon etc. They all use different approaches. Some of them use context (mood filtering in Spotify for instance), others do not use context at all. However there is one thing most of the big names have in common; they all recommend products they want to sell. To put it in other words: they use recommendation system to make the users browse more items and spend more money on their sites rather than to make the users aware of what they want. A movie that isn't owned by Netflix will never be recommended to the user by Netflix, no matter how good the movie is. There are a few platforms, however, that recommends movies for the sole purpose of helping you, and even fewer that takes context in to consideration. One such site that was found is "Jinni", however, it is not (anymore) open for any user. It is limited as an API, in business-to-business licensing, where it impacts businesses like Comcast's Xfinity product (and others whose capabilities benefit from smart entertainment search). We will therefore create a platform, free for any user where users will get movie recommendations and try to identify features that is needed/wanted to make recommender system more popular. Maybe Netflix is good enough, or maybe individuals prefer to find movies themselves without any help from an application.

## 1.2 Previous work

In the authors specialization project [28] in the course "Computer Science, Specialization Project (TDT4501)", we could see the drawbacks of using memory-based collaborative filtering. In this thesis, a better approach has been found and a prototype has been implemented based on different results that were found during the thesis.

## 1.3 Contribution

This section describes the main contribution from this thesis.

- **State of the art:** A review of existing methods to recommend items.

- **Prototype:** A proof of concept prototype to recommend movies.
- **Evaluation:** Evaluation of the performance, both speed and the accuracy of the recommendations. More detailed explanation in section 3.3 : Evaluation plan.

## 1.4 Report structure

This report begins with a state of the art of the recommendation systems literature in Chapter 2. This chapter contains approaches to individual recommendation Systems and context awareness. Chapter 3 describes the methodology used for research and evaluation. Chapter 4 describes how the prototype is implemented. Chapter 5 evaluates the prototype. Chapter 6 discusses the evaluation and the prototype, before the conclusion is made in chapter 7 with ideas for future work.



## Chapter 2

# Background

This chapter will focus on research that is related to this thesis and recommendation systems in general. We will specifically be looking at things like algorithms, challenges and strengths related to recommendation systems.

### 2.1 Recommendation systems

”The goal of a recommender system is to generate meaningful recommendations to a collection of users for items or products that might interest them” [22]. There are is ”state of the art” for recommendation systems in general, because different platforms has their own data, goals, features and more. Different approaches are more suited for different recommendation systems. It is however, important that one know the strengths and weakness of different approaches when choosing one for implementing a recommendation system.

### 2.2 Techniques

#### 2.2.1 Content based filtering

An approach when designing recommendation systems is content-based filtering [21], [25]. Content-based filtering recommends items to a user by matching users

preferences with items. .

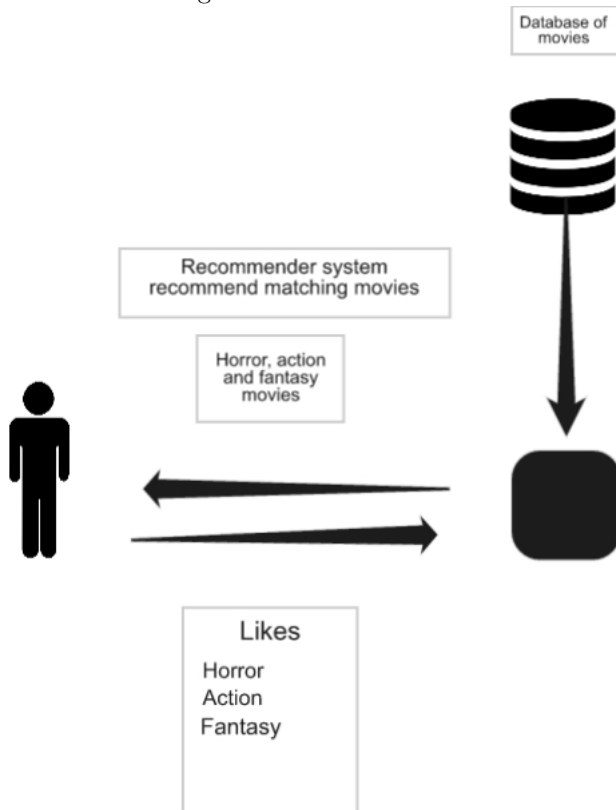
Content-based filtering in general has three steps, how to obtain these steps might be different from system to system.

- **Analyzing the content:** This step is about representing the content of the items. Somehow extracting the relevant information and then use this as an input for the next step.
- **Learning the profile:** This step takes the data from the previous step and creates a user profile based on that data.
- **Filtering:** This steps finds the items that matches the profile and recommends them.

### Advantages

- As we can see from previous work, collaborative filtering is depended on finding similar users to be able to predict ratings. Content based does not have this requirement. Content-based filtering is capable of recommending items regardless of others' ratings. The prediction is based solely on the user's preferences and item details. In previous work, we had scenarios where the algorithm could not find enough similar users, and therefore failed to predict any movies to the user. This is not a problem for a content-based filtering.
- It is easier to explain to the user why a certain item got recommended. This may cause the user to contribute to the system to create better recommendations. For example: if the user has previously stated that he likes action movies (on his profile for instance) or if all the movies he liked so far were action movies, the system would continue to recommend action movies. However, what if he changed his mind, or if it was just a coincident that all those movies were action movies? He could now tell the system that he does not necessarily like action movies and the system

Figure 2.1: Illustration of content-based filtering



would then take that into consideration and try to recommend other types of movies.

- When new items gets added to the system, unlike collaborative filtering, it does not need someone to rate it before it can be recommended to a certain user.

### Drawbacks

- For content-based filtering to be effective, the content needs to have enough information, so the system can discriminate the items user likes or dislike.

- Since it only recommends similar items that a user has liked, the recommendation will never be a surprise. The recommendation is very predictable. If a user likes an action movie, only action movies will be recommended. The user might also like comedy movies, but they will never get recommended, unless the user watches one and likes it.
- Since the recommendation is based on the user's history, making a recommendation for a new user without any history will be problematic. A way around both this and the previous point is to let the users define what they like during the account creation. However, some might find it difficult to define what one likes and dislikes.

### 2.2.2 Collaborative filtering

With content-based filtering we needed to analyze the user profile and the items to find out what they like. However, sometimes that is not simple and sometimes the users themselves do not know. What if there was a way to recommend an item to a user without knowing what he or she likes? Collaborative filtering deals with this problem. The idea of collaborative filtering is to recommend a movie based on similar users. If we look at the interviews (appendix A), we can see that people tend to ask other people that they know have similar taste for recommendations. This is the basic concept of collaborative filtering. It looks through all the ratings to find similar users, and then calculates a prediction for a certain movie based on what similar users rated that particular movie. It does not matter what genre you like for a movie. All it knows is that there are similar users to you, and since they like a certain movie, there is a chance that you also like that particular movie.

This section will explain different methods of collaborative filtering. [17, p. 13-23]

### Memory-based

Collaborative filtering is divided into two categories: memory-based and model-based. Memory-based [9] approaches use the past ratings to predict the ratings, while a model-based approaches first creates a model based on past ratings and then uses this model for every recommendation. Here are two examples of methods of memory-based approaches:

**User-based** The first approach we are going to talk about is called the User-Based Nearest Neighbor Recommendation. The idea is to find the K most similar users, to predict a rating for a certain item. The challenge here is to find a good value for K. In other words, how many similar users should the system take into account when predicting a rating for an item? A way to do this is to define a minimum similarity threshold, which will only take users that have a higher similarity value then the threshold into account. However, this raises another problem: How high should this threshold be to give the best accuracy

There are two common ways to calculate the similarities of two users: Cosine-based and Pearson correlation-based [12]. Cosine-based regards two users as two vectors, and then calculates the similarity between user u and v by calculating the cosine of the angle between the vectors:

$$simU_{u,v} = \cos(\vec{u}, \vec{v}) = \frac{\vec{u} \cdot \vec{v}}{|\vec{u}|^2 \times |\vec{v}|^2} = \frac{\sum_{i=1}^k r_{u,i} r_{v,i}}{\sqrt{\sum_{i=1}^k r_{u,i}^2} \sqrt{\sum_{i=1}^k r_{v,i}^2}} \quad (2.1)$$

Pearson correlation based finds similarities between users by calculating the Pearson Correlation, which is defined by:

$$simU_{u,v} = corr_{u,v} = \frac{\sum_{i \in I_{u,v}} (r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_{i \in I_{u,v}} (r_{u,i} - \bar{r}_u)^2} \sqrt{\sum_{i \in I_{u,v}} (r_{v,i} - \bar{r}_v)^2}} \quad (2.2)$$

Where  $\bar{r}_a$  means the average rating that user a has given among all the ratings user a has given.

Finally, the prediction for the rating of item i by user u can be calculated

by using this formula:

$$pred(u, i) = \bar{r}_a + \frac{\sum_{b \in N} sim(a, b)(r_{v,i} - \bar{r}_v)}{\sum_{b \in N} sim(a, b)} \quad (2.3)$$

Where N is the nearest neighbors/most similar users.

**Item-based** Instead of looking at similar users, the system can look at similar items. Again, we find similar items through past ratings, not through movie data. Meaning, it does not look at what type of genre the movie or book is, but what kind of ratings the item has recieved. Items with similar ratings may be similar. Therefore, if you like one of those movies, then there is a chance that you will like the other ones aswell. One can use the Pearson correlation to calculate the similarity  $sim(i,j)$  between items i and j, however, it has been reported that the cosine similarity measure consistently outperforms the Pearson correlation metric [17]. The basic cosine measure (formula 2.1), does not take the differences in the average rating behavior of the users into account. This can be solved by adjusted cosine measure, which take this into account:

$$simU_{i,j} = \frac{\sum_{u \in U} (r_{u,a} - \bar{r}_u)(r_{u,b} - \bar{r}_u)}{\sqrt{\sum_{u \in U} (r_{u,i} - \bar{r}_u)^2} \sqrt{\sum_{u \in U} (r_{u,b} - \bar{r}_u)^2}} \quad (2.4)$$

Finally, we can predict the rating for user u for an item p as follows:

$$pred(u, p) = \frac{\sum_{i \in ratedItems(u)} sim(i, p) * (r_{u,i})}{\sum_{i \in ratedItems(u)} sim(i, p)} \quad (2.5)$$

### Model-based

Memory-based collaborative filtering has one major drawback;scalability. Imagine going through millions of ratings for every prediction. This would take too much time and would therefore not be viable for big systems. This is where model-based collaborative filtering [18] comes in. The main idea is to create a model which the system can base the prediction upon instead of constantly going through all the ratings. It only looks through the ratings once to create the model. This takes much time, but it does not need to be done during production/live time. There are many different models. Here are some of them::

### Bayesian network

It is also possible to predict ratings by using probability. By calculating the probability of a rating by a certain user to a certain movie, we can then choose the rating with the highest probability. The first approach we will be looking at is using the Bayesian network as our model. There are many different ways of using Bayesian network for collaborative filtering [20][13] [11]. [11] presents two approaches to collaborative filtering. One is based on clustering. Here the probability of each rating are conditionally independent given membership in an unobserved class variable  $C$ . Meaning, there are many variables that decide which class you belong to and therefore decides what items you like. However, given the class, what you like is no longer depended on those variables. The idea is that there exist similar users with similar tastes and those should be clustered in the same group. The probability of an individual being in a particular class and a complete set of vote values can be expressed as:

$$Pr(C = c, v_1, v_2, \dots, v_n) = Pr(C = c) \prod_{i=1}^n Pr(v_i | C = c) \quad (2.6)$$

The other approach is a Bayesian network where each node represents each individual item. And each item would have different states which represents the possible ratings, including "no rating" to represent missing rating. We can then apply an algorithm for learning Bayesian networks to the training data. The resulting network would then contain items with associated parent items that are the best predictors of its votes. Each conditional probability table is represented by a decision tree encoding the conditional probabilities for that node.

### Matrix factorization/latent factor models

There might be underlying features/factors that make you like certain movies. These features are called latent factors. You might not know about them, but if the system is able to find them, it can find other movies with the same features to recommend to you. For instance, if there are two movies that you really like,

then they must share a common feature that made you like those two movies. It is possible to find these hidden features by looking at the rating database and use a method called matrix factorization. In short, the idea of matrix factorization is to find latent (hidden) factors based on the rating pattern to characterize items and users. This section will talk about the SVD (Singular Value Decomposition), which is one method of matrix factorization.

### SVD

In linear algebra, singular value decomposition is a factorization of a real or complex matrix. This can be used to predict the user's rating. The theorem states that: Suppose  $M$  is a  $m \times n$  matrix, then there exists a factorization of  $M$  of the form [8, p.371]:

$$M = U\Sigma V^T \quad (2.7)$$

where  $\Sigma$  is a  $m \times n$  diagonal matrix with non-negative real numbers on the diagonal, and  $U$  is an  $m \times m$ , and  $V$  is an  $n \times n$  matrix. However we can approximate the full matrix by observing only the most important features, hence picking  $\Sigma$  to only contain the  $k$  largest singular values of  $M$ . Matrix  $V$  corresponds to the users and matrix  $U$  to the items. By projecting these matrices into a two-dimensional space, one can see what items are similar, and which users are similar. To predict user  $A$ 's rating, one must first find where it would be in the two-dimensional space. This is done by multiplying  $A$ 's rating vector by the  $k$  subset of  $U$  and the inverse of the  $k$ -column singular value matrix .

$$A_{2D} = A \times U_k \times \Sigma_k^{-1} \quad (2.8)$$

After finding  $A$ 's data point, one can now use different methods to predict  $A$ 's rating. One can for instance use the cosine similarity as described earlier, using the users that are closest to  $A$ .



### Factorization machine

Matrix factorization has gained a lot of popularity since the last Netflix competition, because of its performances. However it only works well with specific data and it is not capable of general prediction tasks. FMs are a general predictor that can work with any real valued feature and it works really well with sparsed data vectors[26], [14]. In recommendation system where you have alot of movies, but each person has only rated a few of them, this can come in handy. Later, we will se that it can mimic other models, for instance matrix factorization, depending on the input data. The factorization machine model looks like this:

$$\hat{y}(x) := w_0 + \sum_{i=1}^n w_i x_i + \sum_{i=1}^n \sum_{j=i+1}^n \langle v_i, v_j \rangle x_i x_j \quad (2.9)$$

Where

$$\langle v_i, v_j \rangle \quad (2.10)$$

is the dot product of two vectors. Where you want to predict a rating  $\hat{y}$  based on the input variable vector  $x = [x_1, \dots, x_n]$  where each  $x$  is a variable that represents a certain feature. The features could be time of the day, genre, last movie rated and so on.

$W_0$ ,  $w$  (one  $w$  for each variable), and matrix  $V$  are parameters that have to be learned through training.  $W_0$  is a global bias,  $W_i$  represents the strength of the  $i$ -th variable and a row  $v_i$  in  $V$  is a vector representing the  $i$ -th variable. The product of two  $v$  vectors describes the interaction between those two vectors. For example, user A can be represented by a vector  $v_A$  and movie ST can be represented as a vector  $v_{ST}$ . This is where collaborative filtering comes in to the picture for Factorization Machine. If user A has not yet rated movie ST (see example figure 2.2), then a direct estimation of the interaction between those two would be 0 (no interaction). However, Factorization Machine uses collaborative filtering principles to estimate the interaction  $\langle v_A, v_{ST} \rangle$ . Since user B rated movie SW and ST with similar ratings (4 and 5), then  $v_{ST}$  and  $v_{SW}$  must be similar. That must mean that  $\langle v_A, v_{ST} \rangle$  is similar to  $\langle v_A, v_{SW} \rangle$ .

Intuitively, it looks like the computational time complexity of the model is quadratic ( $O(kn^2)$ ), because of the nested double sum. However, because of the factorization method, the formula can be reformulated to give us a linear time complexity ( $O(kn)$ ). This reformulation can be found here [26] Below is an example of what kind of data the Factorization machine can handle.

Feature vector $x$														Target $y$								
$x^{(1)}$	1	0	0	...	1	0	0	0	...	0.3	0.3	0.3	0	...	13	0	0	0	0	...	5	$y^{(1)}$
$x^{(2)}$	1	0	0	...	0	1	0	0	...	0.3	0.3	0.3	0	...	14	1	0	0	0	...	3	$y^{(2)}$
$x^{(3)}$	1	0	0	...	0	0	1	0	...	0.3	0.3	0.3	0	...	16	0	1	0	0	...	1	$y^{(2)}$
$x^{(4)}$	0	1	0	...	0	0	1	0	...	0	0	0.5	0.5	...	5	0	0	0	0	...	4	$y^{(3)}$
$x^{(5)}$	0	1	0	...	0	0	0	1	...	0	0	0.5	0.5	...	8	0	0	1	0	...	5	$y^{(4)}$
$x^{(6)}$	0	0	1	...	1	0	0	0	...	0.5	0	0.5	0	...	9	0	0	0	0	...	1	$y^{(5)}$
$x^{(7)}$	0	0	1	...	0	0	1	0	...	0.5	0	0.5	0	...	12	1	0	0	0	...	5	$y^{(6)}$
	A	B	C	...	TI	NH	SW	ST	...	TI	NH	SW	ST	...	Time	TI	NH	SW	ST	...		
	User				Movie					Other Movies rated						Last Movie rated						

Figure 2.2: Example of feature vectors taken from [26]. Each row represents a feature vector  $x^i$  with its corresponding target  $y^i$ . The first row represents user A who has rated movie TI, has also rated NH and SW, no last ratings and the rating of movie TI was 5. As one may see, there are multiple features leading up to the rating of 5 and not only who rated what movie what rating like previous approaches. This matrix can be used as input to train the model.

To learn the parameters needed in the model, one can use gradient descent methods like stochastic gradient descent, or alternating least squares.

**Stochastic Gradient Descent,** As one can see, the Factorization Machine model can be represented as a linear regression with multiple variables (multiple linear regression). Gradient descent is a known algorithm used to estimate the weights  $W_i$  for each variable. Since a normal (batch) gradient descent iterates through every single data point before updating the estimation, it will consume too much time with a big data set. Therefore, stochastic gradient descent, which

updates the estimation after a single data point, is preferred. GD tries to find a local minimum of a function, in our case, it would be the error of the predicted rating and the actual rating. Intuitively explained, this is how it works:

1. First initialize the parameters, for example setting them to 0.
2. Predict the rating with the parameters
3. Calculate the difference between the predicted value and the actual value
4. Update the the parameters according to the error
5. Repeat from 2.

The act of iterating through all the data points in a data set (as defined by step 1-4/ above) is called one epoche. A more detailed and mathematical explanation can be found here: [24]

### 2.2.3 Challenges with Collaborative Filtering

Although there are different ways to do collaborative filtering, they all share some common drawbacks. Those drawbacks will be discussed in this section.

#### **Cold start**

Probably the most known and intuitive drawback with collaborative filtering is the cold start problem. Since collaborative filtering does its predictions based on the users' history, what will happen when the system tries to predict a movie without any rating history to a new user? Some solutions to this can be either asking the user to rate several movies upon account creation and/or the use of content-based filtering until enough data is gathered to do collaborative filtering.

#### **The long tail**

The addition of new items gives rise to another problem; new items have no ratings and therefore cannot be recommended to anyone. This creates the long

tail problem with the rich get richer effect. Movies with few ratings will keep being ignored while movies with many ratings will keep getting more ratings. The problem is that the long tail of unpopular movies might contain preferable movies to certain users, but they stay unpopular because they never get recommended and not because it is a bad movie.

### **Sparse data**

In a recommendation system, the user has only rated a few movies out of millions of movies. Therefore, the data is very sparse. This makes it harder to provide satisfyingly accurate results. However, Factorization Machine seems to handle this problem quite well [26].

## **2.2.4 Hybrid**

As you can see, each method has its own weaknesses, therefore it is very common to combine different techniques to cover these up. This is known as hybrid recommender systems [10]. In a hybrid recommender system you combine multiple techniques to gain fewer drawbacks. Most commonly, collaborative filtering is combined with other techniques to avoid their biggest drawback; the cold start. There are different ways to implement hybrid recommender systems [27]:

- "Implementing collaborative and content-based methods separately and combining their predictions.
- Incorporating some content-based characteristics into a collaborative approach.
- Constructing a general unifying model that incorporates both content-based and collaborative characteristics"

## 2.3 Context-aware Recommendation Systems

Previously, we have been discussing which items match with which user based on their taste. Either through collaborative filtering or content-based filtering, the system tries to find data about the users and the items to match them. In this section we are going to factor in another element: the context. The context is a factor that has nothing to do with the items or the users, but the environmental state at the time of recommendation. For example, time of the day, the weather, the mood and more. Context is important for several recommendation system, and it is a factor this thesis is going to take into account for, but first we need to define what context is. This thesis is going to denote it as "situation parameters that can be known by the system and may have an impact on the selection and ranking of recommendation results" [17]. In the previous sections, we have only been looking at predicting ratings based on the users and items, where for example here [29], they have taken the location of the current user into consideration before making the recommendation.

$$R : User \times Items \implies Rating \quad (2.11)$$

Hence, we have only been looking at two-dimensional (2D) systems since they consider only the User and the Item dimensions in the recommendation process. This thesis however, wants to take context into consideration as well.

$$R : User \times Items \times Context \implies Rating \quad (2.12)$$

## Chapter 3

# Research method

### 3.1 Design Science Research

Design science research is a set of analytical techniques and perspectives for performing research in Information Systems. In short it involves creating an artifact and evaluating it. This thesis is going to follow seven guidelines provided by Hevner et al.[16] (figure 3.1).

Figure 3.1: Design-Science Research Guidelines, figure taken from [16].

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

### **3.1.1 Design as an Artifact**

In this thesis an artifact will be made. A web based proof of concept prototype of a context-aware recommendation system will be created.

### **3.1.2 Problem relevance**

Watching a movie that you like can be entertaining and fun. Watching a movie that you do not like is not that fun. How can you find out if you like a movie or not before even watching it? A recommendation system can help you with that. It might not be 100% correct, but if the system is good, it will be able to predict correctly most of the time.

### **3.1.3 Design Evaluation**

The system will be evaluated by its recommendations, how precise they are and how fast the recommendations are made. It will also be tested by real users where they will be asked if such systems are useful.

### **3.1.4 Research Contribution**

A prototype of a webapplication for recommending movies will be created. Factorization Machine will be evaluated through that webapplication. Finally, research will be done to see what kind of features are important for a movie recommendation system to be popular.

### **3.1.5 Research Rigor**

This thesis explains how the system is implemented, thus it should be possible to reproduce the results. The code is also available for anyone to clone. The dataset used for training and testing is from Movielens (<http://grouplens.org/datasets/movielens>).



### 3.1.6 Design as a Search Process

Design as a search process motivates us to use an iterative method. Although this thesis is about single user recommendation systems and not group recommendation system. This application has been implemented with the results from the previous work in mind. The application has also been through two different phases, where the prototype were improved based on interviews.

### 3.1.7 Communication of Research

This thesis shows both in detail how the algorithms are implemented and also a more general view of how the prototype works through the use of drawings and pictures.. The source code will also be available on Github for anyone to see.

## 3.2 Evaluation tools

### 3.2.1 Mean Squared Error

To calculate the accuracy of the predictions, the mean squared error has been used. MSE assesses the quality of the predictor by calculating the mean of the squared errors.

$$MSE = \frac{1}{n} \sum_{i=1}^n (\hat{Y}_i - Y_i)^2 \quad (3.1)$$

To calculate this, `mean_squared_error` module from `sklearn`<sup>1</sup> library has been used.

### 3.2.2 Python time module

To measure the running time of certain algorithms and certain pieces of the code, the python "time" module <sup>2</sup> has been used.

---

<sup>1</sup><http://scikit-learn.org/stable/>

<sup>2</sup><https://docs.python.org/2/library/time.html>

### 3.3 Evaluation plan

The prototype will be evaluated in two different ways. The first way will evaluate different collaborative filtering approaches by their accuracy. The second way will evaluate our prototype with only core features (phase one) to find out what features can be added to make the prototype more appealing. To evaluate the accuracy, MSE will be used. For the second part, a round of interviews will be held. The main difference between the two is that the evaluation of the accuracy will give us an idea of how good the predictions are while the evaluation of the features is for improving our prototype.

## Chapter 4

# Architecture and Implementation

This chapter describes how the system was implemented. First it will discuss the architecture, before it goes into the details of the implementation. The implementation was done in two phases. One prior to the interviews and one after. The first prototype contained the absolute core features. Then a round of interviews were made to seek out more features that could benefit the prototype.

### 4.1 Data set

The data sets this thesis is using were found at <http://grouplens.org/datasets/movielens/> under "recommended for education and development". Note that they update this data set continuously, therefore the data set will also be included with this thesis, with the same license conditions as MovieLens. The license usage can be found in Appendix B. One data set contains the rating, but does not include the movie name. Another data set contains movie name, the corresponding movie ID and its genres. We have used them both to connect each rating to the correct movie before adding them to the database.

## 4.2 Chosen collaborative filtering method

Before choosing one collaborative filtering method to be used in our prototype, two methods were implemented and compared. User-based nearest neighbor and Factorization Machine. After the testing (chapter 5.2 for evaluation and comparison of those), we decided that Factorization Machine was the way to go. Not only does it give high accuracy, but also high flexibility. The reason why factorization machine was picked over other model-based methods is because of the flexibility. Our prototype would need improvements and changes in the future before it can go live. Therefore we want something that is able to adapt easily to changes. Factorization Machine provides us with this because of how general it is. It is not implemented to a specific input. The model adapts to the training data. In the future we might get other data sets that we currently do not know about, and with Factorization Machine, we do not need to change the implementation for the model to work, only the part where we parse the data, not the algorithm itself. For example, for this thesis, the data we are using contains userID, movieID, rating and timestamp, where userID and movieID are the two features that have been used to predict the rating. However, in the future we might get data sets with even more features, like gender, "last movie rated", "post code" and so on. Then we will only need to implement a new way to parse these data, but the implementation of factorization machine stays untouched.

## 4.3 Phase one

In phase one, only core features were implemented. The prototype met the requirements below, which resulted in a very basic typical recommendation system. This prototype was then used to find other features through interviews.

### 4.3.1 Scenarios:

- Person A is bored, so he wants to watch a movie. However, there is nothing on the television and he does not know what to watch. He turns on his computer and googles "Best movies 2016", but different entries give different movies, and he does not know which one to choose. He also has a meeting at a certain time, therefore his time is limited.
- Person B, like person A, also wants to watch a movie. He does not have any time limits. However, he is in the mood for a horror movie.

### 4.3.2 Requirements:

- Recommendation: The system should recommend movies based on user ratings history (Faster than 0.5 seconds, and at least as accurate as user-based filtering.)
- Context: There are two contextual features the system should take into consideration when recommending a movie: the time and the mood of the user.

## 4.4 Phase two

Phase two consisted of improving the prototype from phase one based on the interviews (see Chapter 5: Evaluation for more details). The updated requirements will be described here. We will then introduce the architecture and implementation of the final product.

### 4.4.1 Scenarios:

- Person A is bored, so he wants to watch a movie. However, there is nothing on the television and he does not know what to watch. He turns on his computer and googles "Best movies 2016", but different entries give

different movies, and he does not know which one to choose. He also has a meeting at a certain time, therefore his time is limited. (From phase one)

- Person B, like person A, also wants to watch a movie. He does not have any time limits. However, he is in the mood for a horror movie. (From phase one)
- Person A receives a recommendation from person B, but he is not sure if he can trust his recommendation.

#### 4.4.2 Requirements:

- Recommendation: The system should recommend movies based on user ratings history (Faster than 0.5 seconds, and at least as accurate as user-based filtering.)
- Context: There are two contextual features the system should take into consideration when recommending a movie: the time and the mood of the user.
- Manual research: The user should be able to see the plot, actors and the IMDb rating of the recommended movies.
- Socializing: The user should be able to find other users on the application and see how similar they are. This requirement can be heavily expanded upon, which will be described in future work.

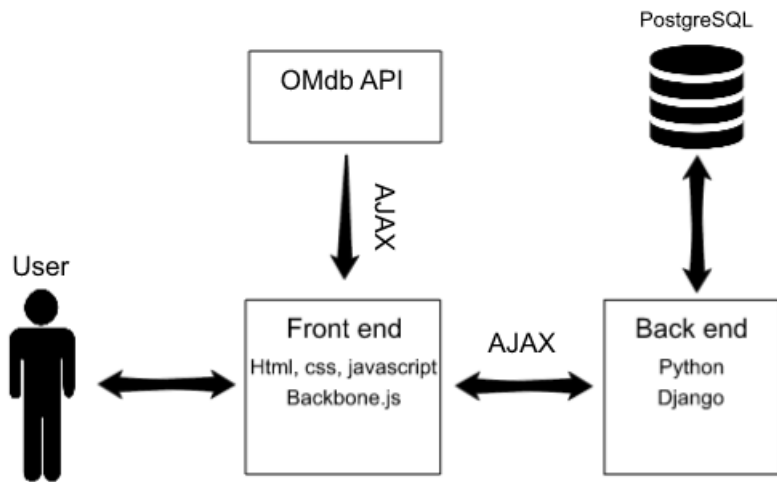
### 4.5 Architecture

This application is divided into two parts: Front End and Back End. The Front End is also connected to another API, OMDb API<sup>1</sup>, which is used to get posters and movie length. In this section we will discuss the architecture of the application further.

---

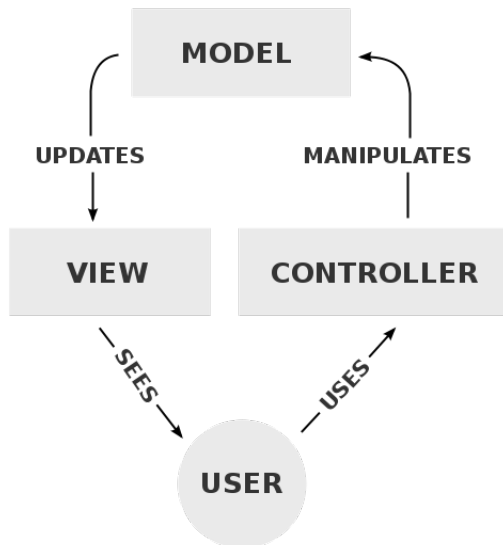
<sup>1</sup><https://www.omdbapi.com>

Figure 4.1: Illustration of overall architecture



#### 4.5.1 Model-View-Controller

MVC is a design pattern that divides the modeling of the domain, the presentation, and the actions based on user input into three separate classes [5].

Figure 4.2: MVC, image from Wikipedia<sup>2</sup>

Each class has its own purpose. The model manages all the data, while it is listening to changes from the controller. The controller handles the input from the user and then changes the state of the view and model accordingly. The view manages the display of the data. This design pattern has been used in both the Back End and the Front End of the application. By separating implementation into different classes we makes it easier to modify in the future, because one can modify the relevant classes without touching the others. In the next sections we can see how it has been used in the Back End and Front End.

## 4.5.2 Back End

The Back End is a Java Representational State Transfer (REST) service written in Python. It has been created using Django REST Framework<sup>3</sup> which is a toolkit for building Web APIs [3]. It is a framework for Django<sup>4</sup> which is a free and open source web application framework, written in Python. The Back

<sup>2</sup><https://en.wikipedia.org/wiki/Model-view-controller>

<sup>3</sup><http://www.django-rest-framework.org>

<sup>4</sup><https://www.djangoproject.com>



End is responsible for doing all calculations. Python has been chosen because of its simplicity in syntax, which makes implementing algorithms much simpler. It also has a lot of good libraries for artificial intelligence like Scikit-Learn <sup>5</sup>. We can then reuse implementations that have already been done, instead of spending much time doing it from scratch.

### **Structure**

The Back End, following Django's structure, is divided into three layers. Django calls these three layers Model, View and Template [1]. However, instead of looking at the names of each layer, we should be looking at the intent of each layer. The model manages the data, the view manages the requests from the user and the template manages the presentation of the data. As one can see, it is actually following the MVC principles, however, they changed the name of view to template and controller to view. Remember, our Back End does not manage the GUI, therefore the view part (template in Django) is non-existent in the Back End.

---

<sup>5</sup><http://scikit-learn.org/stable/>

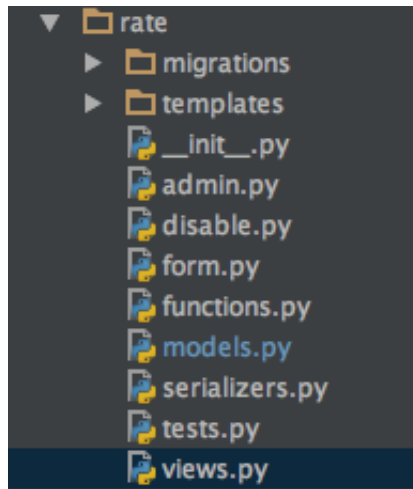


Figure 4.3: Back end structure. There are individual classes for model and view, which acts like model and controller in MVC. The templates folder would have contained html files for rendering purposes if we wanted Django to do the rendering process (view in MVC).

### 4.5.3 Front End

The front end was developed using the Backbone.js framework <sup>6</sup>. The front end is responsible for rendering the data from the back end. It creates a GUI for the users of the system. It is connected to the back end by sending AJAX calls.

#### Structure

Backbone also has a MVC framework and just like python, the naming convention is a bit different. Backbone has Models and Collections that acts like models in MVC, while View acts like Controller and Template acts like View. The model in the back end is typically connected to the database, the model in the front end however, is connected to the REST API from the back end.

---

<sup>6</sup><http://backbonejs.org>

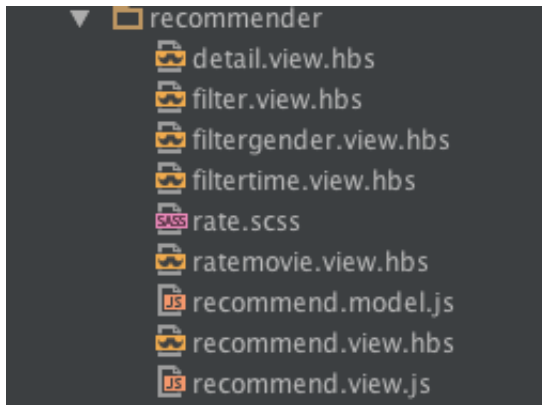


Figure 4.4: Front end structure. Files ending with `-view.js` acts like controllers, `-view.hbs` acts like view and `-model.js` acts like model in MVC.

## 4.6 Implementation approach

This section will go through the implementation and explain how the system recommends a movie in implementation detail. The recommendation consists of three stages:

- Setup
- Defining context (if any)
- Finding recommendations

### 4.6.1 Setup

Since our solution is using collaborative filtering to recommend movies it needs data. This stage consist of two phases: Adding data to the database, and training the model. Since we only need the training data for training the model, we did not add the rating data to the database, but just kept the csv file for when we needed to train the model. When another/better training set is found, then we only need to swap the old one out with the new one, instead of removing /adding from/to the database. However, since the system is going to recommend

movies, the movies and all the\* data about them were\* added to the database. Note that although we did not add the ratings from the training set, the new ratings that are made within the system will be added, because this gives us data about the users of the system. The different types of genres were added manually using the Django admin interface (Figure 4.2). After the genres were added, the movies were also added and connected to their corresponding genres using Python code. The model was then ready to be trained. A function `getdata(request)` was implemented to both upload data to the database and train the model. It will check whether or not the movies have been uploaded. If it has, it will train the model, if not, it will upload.

Listing 4.1: Function for either upload data or training the FM depending on whether the data is uploaded or not

```

1 def getdata(request):
2     global fm
3     global v
4     if Movie.objects.filter().count()>0:
5         def loadData(filename,path="data/"):
6             data = []
7             y = []
8             users=[]
9             items=[]
10            with open(path+filename) as f:
11                for line in f:
12                    (user,movieid,rating,ts)=line.split(',')
13                    data.append({ "user_id": str(user), "movie_id":
14                                str(movieid)})
15                    y.append(float(rating))
16                    users.append(user)
17                    items.append(movieid)
18
19                return (data, np.array(y), users, items)
20
21            (train_data, y_train, train_users, train_items) = loadData(
                "train.csv")
22            (test_data, y_test, test_users, test_items) = loadData("

```

```
        test.csv")
22     v = DictVectorizer()
23     X_train = v.fit_transform(train_data)
24     X_test = v.transform(test_data)
25
26     fm = pylibfm.FM(num_factors=10, num_iter=100, verbose=True,
        task="regression", initial_learning_rate=0.001,
        learning_rate_schedule="optimal")
27     fm.fit(X_train,y_train)
28     predictions = fm.predict(X_test)
29     from sklearn.metrics import mean_squared_error
30     print("FM_MSE: %.4f" % mean_squared_error(y_test,
        predictions))
31     context={
32         "text" : "Already_uploaded"
33     }
34     return render(request, "getdata.html", context)
35 else:
36     movie_list=[]
37     m = open('data/movies.csv', 'r')
38     for line in reader(m):
39         movie_list.extend(line)
40     m.close
41     for x in range(1,len(movie_list)):
42         if x%3==1:
43             s=movie_list[x]
44             text = s[s.find("(")+1:s.find(")")]
45             count=0
46             while (text.isdigit()==False and count <15):
47                 s=s[s.find("")+1:]
48                 text=s[s.find("(")+1:s.find(")")]
49                 count = count +1
50             if count == 15:
51                 year=-1
52             else:
53                 year=int(text)
54             movie = Movie(movieName=movie_list[x],movieId=
        movie_list[x-1], year=year)
```

```
55         movie.save()
56     elif x%3==0:
57         for genre in movie_list[x-1].split('|'):
58             if genre != '(no_genres_listed)':
59                 movie.movieGenre.add(Genre.objects.get(
60                     genre=genre))
61
62     context={
63         "text" : "Complete"
64     }
65     return render(request, "getdata.html", context)
```

To run this code, one may visit *yourdomain/getdata*. This is for practical reasons. One may rerun the training just by refreshing this page. However, this possibility should only exist during testing and not in production. Only authorized people should be able to rerun the training or upload data.

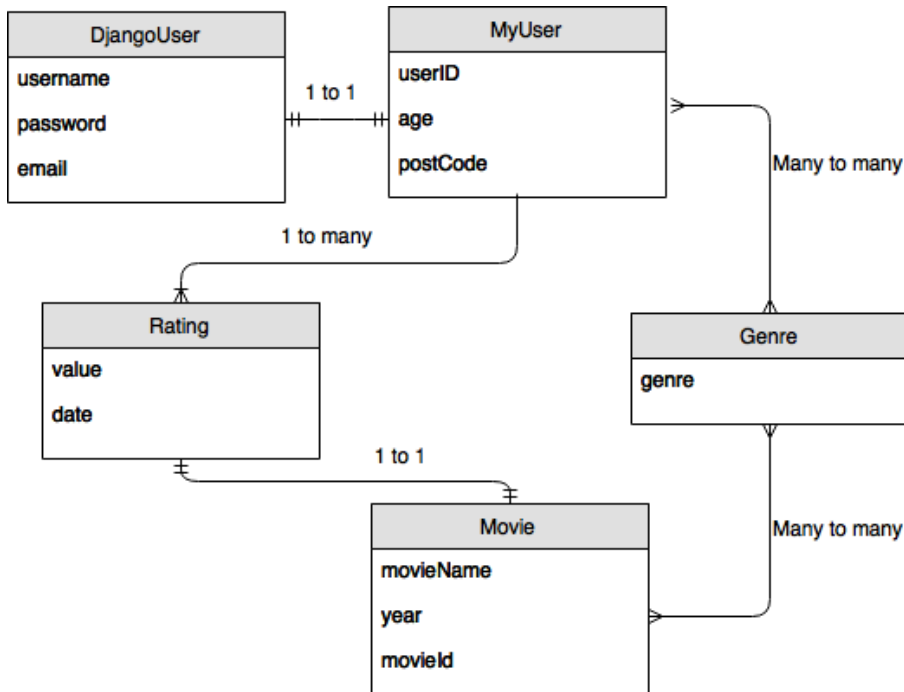


Figure 4.5: Database structure

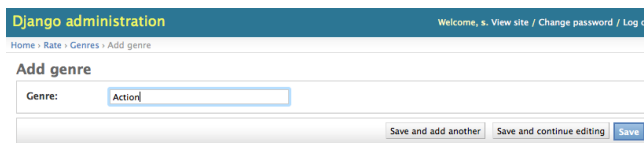


Figure 4.6: Django admin interface

### 4.6.2 Defining context

The context parameters that have been used in this thesis are genres (mood) and time length of the movies. Note that this differs from user preference, because the user might prefer a certain genre, but might want to see something else at that moment because he for instance saw it too many times last week. The

mood might also affect what kind of genre he wants to see. However finding correlation between genre and mood has not been prioritized in this thesis. Therefore, the user is allowed to directly choose the genre.

### 4.6.3 Genre

There are two approaches of adding genre to the recommendation. One; filter out unwanted genres and run the recommendation algorithm on that set. Two; instead of filtering out unwanted genres , add a higher score to the preferred genres. The idea is that there might exist one or more movies, that would be preferred despite its genre. The first approach rules certain genres completely out, assuming that the user does not want to watch anything else than the chosen genres, which is not always the case. The problem with approach number two is to find how much the chosen genre should count in the recommendation. Factorization machine does have a way to handle this. We could add "chosen genre" as a feature to our model. However, we do not have such data. Thus approach one was chosen because the model can not be trained to handle such input.

Listing 4.2: Genre filtering before creating a matrix of unseen movies that need prediction before sorting it from highest prediction to lowest prediction

```
1     elif 'filter' in request.GET:
2         genre=Genre.objects.get(genre=request.GET.get('filter')
3             )
4         movies = Movie.objects.filter(movieGenre=genre).exclude
5             (movieId__in=seenMovies)
6     else:
7         movies = Movie.objects.exclude(movieId__in=seenMovies)
8     predictList=[]
9     for movie in movies:
10        predictList.append({'user_id':str(user.userID),'
11            movie_id':str(movie.movieId)})
12    predict=v.transform(predictList)
13    y=fm.predict(predict)
```



11

```
newlist = sorted(predictList, key=lambda k: k['rating'],
                 reverse=True)
```

#### 4.6.4 Time

Just like with genres, we have to decide whether or not we want to rule out movies with undesired time lengths. Because time length is usually important when there exist a certain time limit for a user, we have decided to go for approach one. Meaning, the system assumes that it does not matter whether the movie is 60 minutes long or 120 minutes long, unless you tell the system that you are unable to (not only by preference) watch something that lasts longer than 60 minutes. In that case, everything that last longer will be filtered out. The main difference between filtering genre and filtering time is the order of when it is done. While filtering genre happens before Factorization Machine does its calculations, time filtering not only happens after the recommendation calculation has been done, but also from the Front End and not Back End. The reason for this is simple: we do not have the data of the time length in our database. We get this data from Omdbapi. This means that the Back End does its predictions first and responds with the top K rated movies first, before the front end collects these movies and then filters out too long lasting movies by checking with Omdbapi. This changes the pagination a bit. The pagination algorithm is done by the back end. When no time limit has been chosen, each page will show the top ten (the page size has been arbitrarily chosen) movies that have been recommended. For each page the front end will request movie recommendations from the back end and the back end will respond with ten movies. However, when a time limit has been chosen, the back end will respond with 100 (also arbitrary chosen) movies for each page, but not all of them will be shown. The front end will only show the movies that are short enough. This means that each page has different amounts of movies, ranging from 0-100 movies, depending of how many of those 100 movies fit the criteria.

Listing 4.3: Finally checks if user has chosen a time limit. If no time limit has been chosen, a normal page of 10 movies will be returned. If a time limit has been chosen, a page of 100 movies will be returned where it will be filtered from the front end

```
1     if "time" in request.GET:
2         paginator = TimeCustomPaginator()
3         result_page = paginator.paginate_queryset(newlist,
4             request)
5     else:
6         paginator = CustomPaginator()
7         result_page = paginator.paginate_queryset(newlist,
8             request)
9     return paginator.get_paginated_response(result_page)
```

### 4.6.5 Recommendations

For recommendations we want to find the top ten movies and show it to the user. We do this by predicting ratings for each movie and show the top ten highest rated movies. As shown in section 4.2, Factorization Machine is the chosen approach to predict movies. When Steffen Rendle introduced the Factorization Machine to the world, he also showed us an implementation which can be found at <https://github.com/srendle/libfm>. Although it is not in Python there exists a Python wrapper for it (created by someone else) <https://github.com/jfloff/pywFM>. However, it does not separate training and predicting. Meaning, everytime the module wants to predict a new rating it will have to retrain the model, which is time-consuming and not ideal for our prototype. Another Python Factorization Machine module was found and used: PyFM<sup>7</sup>. Although it has some flaws (more about this in evaluation), it was good enough for our proof of concept prototype.

---

<sup>7</sup><https://github.com/coreylynch/pyFM>

### Factorization Machine

We have earlier seen how Factorization Machine works. This section will describe how it works with our data sets, and how the chosen Python module has been used in our prototype.

```
1,16,4.0,1217897793
1,24,1.5,1217895807
1,32,4.0,1217896246
1,47,4.0,1217896556
1,50,4.0,1217896523
1,110,4.0,1217896150
1,150,3.0,1217895940
1,161,4.0,1217897864
1,165,3.0,1217897135
1,204,0.5,1217895786
1,223,4.0,1217897795
1,256,0.5,1217895764
1,260,4.5,1217895864
1,261,1.5,1217895750
1,277,0.5,1217895772
1,296,4.0,1217896125
1,318,4.0,1217895860
1,349,4.5,1217897058
1,356,3.0,1217896231
1,377,2.5,1217896373
1,380,3.0,1217896030
1,457,4.0,1217896015
1,480,3.5,1217895972
```

Figure 4.7: Dataset in a csv (Comma-separated values ) format. The first value represents userID, second value represents moveID, the third value represents the rating value and the last value represents the timestamp.

This is how our data set looks like, which then needs to be parsed to create a matrix that will be used as input for training the Factorization Machine model. When the model is done training, it is then ready to predict ratings. PyFM uses stochastic gradient descent to estimate the missing model parameters:  $w_0$ , matrix  $V$ , and  $w$ .

	User1	User2	User3	...	Movie1	Movie2	Movie3	
x1	1	0	0	...	1	0	0	5
x2	1	0	0	...	0	1	0	3
x3	1	0	0	...	0	0	1	2
x4	0	1	0	...	0	1	0	4
x5	0	1	0	...	1	0	0	1
x6	0	0	1	...	0	0	1	2

Figure 4.8: Feature vector representation of the data set. This kind matrix has been used as input for training the FM model.

In this case, where we only have userID and movieID, the Factorization Machine would actually mimic a matrix factorization model [19], because the only non-zero feature  $x_i$  is  $x_u$  (userID) and  $x_m$  (movieID). Formula 2.9 then becomes:

$$\hat{y}(x) := w_0 + w_u + w_m + \langle v_u, v_m \rangle \quad (4.1)$$

When Factorization Machine uses this as input for the training (Listing 4.1), values for  $w_0$ ,  $w_u$ ,  $w_m$ , and  $\langle v_u, v_m \rangle$  will be stored and then used to predict a rating whenever `fm.predict()` is called .

### User based with Pearson correlation

Although this approach is not used in the prototype, it was implemented to compare it with Factorization Machine. This section will describe how it was implemented.

Listing 4.4: First, a dictionary is created to store information about who rated what

```

1 ratingDictionary= {}
2 movieSets=[]

```

```

3     for x in range(0, len(train_data)):
4         movieSets.append((str(train_data[x]['user_id']), str(
5             train_data[x]['movie_id'])))
6         if train_data[x]['user_id'] in ratingDictionary:
7             ratingDictionary[train_data[x]['user_id']].append({
8                 train_data[x]['movie_id']: y_train[x]})
9         else:
10            ratingDictionary[train_data[x]['user_id']] = [{
11                train_data[x]['movie_id']: y_train[x]}]

```

Listing 4.5: Second, similar users are found by iterating through every user and comparing common movies and using the Pearson Correlation

```

1     countIndex=0;
2     predictions=[]
3     for user in test_users:
4         userratings=[]
5         similarusers= {}
6         for userrating in ratingDictionary[user]:
7             for key in userrating:
8                 userratings.append(userrating[key])
9         for userTwo in ratingDictionary:
10            if str(user) != userTwo and (userTwo, str(test_items
11                [countIndex])) in movieSets:
12                movieListOne= []
13                movieListTwo=[]
14                for rating in ratingDictionary[user]:
15                    for movieId in rating:
16                        for ratingTwo in ratingDictionary[
17                            userTwo]:
18                            for movieTwo in ratingTwo:
19                                if movieId == movieTwo:
20                                    movieListOne.append(rating[
21                                        movieId])
22                                    movieListTwo.append(
23                                        ratingTwo[movieTwo])
24                                    continue
25            if (len(movieListOne)>3):
26                similarity=pearson(movieListOne,

```

```

23         movieListTwo)
24         if similarity>0.6:
25             similarusers[userTwo]=similarity
                similarusers[userTwo+'list']=
                    movieListTwo

```

Listing 4.6: Finally, the rating for the current movie by the current user of the current data point in the test set is predicted using the formula (2.3). The prediction is then added to a list of predictions. The prediction list will then contain all the predicted ratings in the same order as the data sets in the test set. It is then ready to be evaluated

```

1         teller = 0
2         nevner = 0
3         for simuser in similarusers:
4             if isinstance(similarusers[simuser],int):
5
6                 nevner = nevner + similarusers[simuser]
7                 for movie in ratingDictionary[simuser]:
8                     if test_items[countIndex] in movie:
9
10                        teller=teller+similarusers[simuser]*(
                            movie[test_items[countIndex]]-
                            average(similarusers[simuser+'list'
11                                ]))
                            continue
12            if nevner==0:
13                predictions.append(average(userratings))
14            else:
15                predictions.append(average(userratings)+teller/
                            nevner)
16            countIndex=countIndex+1;

```

## 4.7 Prototype

### 4.7.1 API

Django rest framework gives us a browsable API, which makes it easier to test our API. Here is an example of an API response when a recommend request has been done to the back end. To be able to get this response, the user has to be logged in. The recommendation is of course personalized accordingly to the Factorization Machine model. All new users without any rating history will therefore get the same recommendations until ratings have been done and updated in the model. A document of API details is provided in the appendix.

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
  "count": 8920,
  "movies": [
    {
      "movieName": "Godfather, The (1972)",
      "movie_id": "858",
      "user_id": "2000",
      "rating": 4.5185156392377301
    },
    {
      "movieName": "Lives of Others, The (Das Leben der Anderen) (2006)",
      "movie_id": "44555",
      "user_id": "2000",
      "rating": 4.5041668567019784
    },
    {
      "movieName": "Eat Drink Man Woman (Yin shi nan nu) (1994)",
      "movie_id": "232",
      "user_id": "2000",
      "rating": 4.4960770513463189
    },
    {
      "movieName": "Capturing the Friedmans (2003)",
      "movie_id": "6380",
      "user_id": "2000",
      "rating": 4.4952840525216446
    },
    {
      "movieName": "Rashomon (Rashômon) (1950)",
      "movie_id": "5291",
      "user_id": "2000",
      "rating": 4.4917787159926421
    }
  ]
}
```

Figure 4.9: Example of browsable API

## 4.7.2 GUI

Previously, we have been looking at the architecture of the system. In this section, the GUI of the system will be presented. The GUI is a web-based recommendation system, where each page of the site has its own functionality where it gets the data from the mentioned API.

### Login

The login page is the first page the user is introduced to, because to get access to any of the other features the user has to log in. To log in, the user has to create an account first.

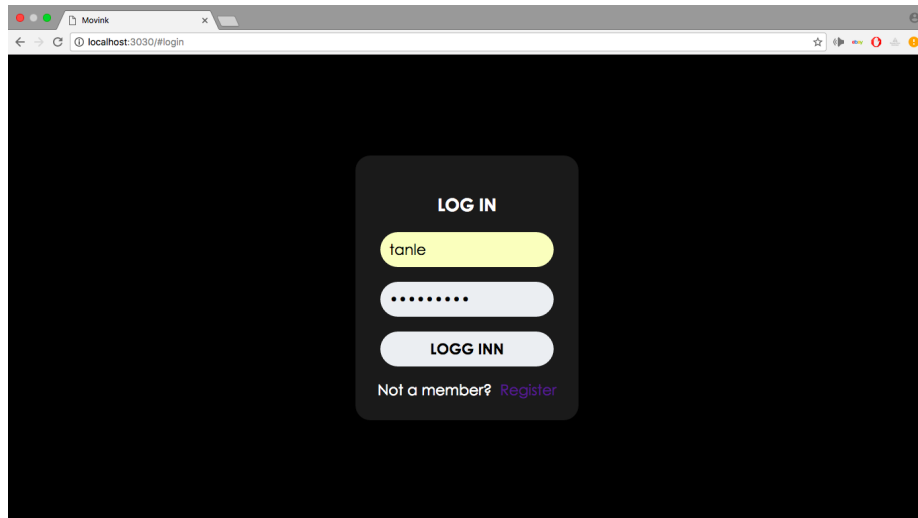


Figure 4.10: Login page

### Rate

When the user is logged in, they will be redirected to a page where they can rate movies. They can either search for a movie by name specifically or browse until an interesting movie is found. By clicking on the poster a screen to rate the movie will be presented.



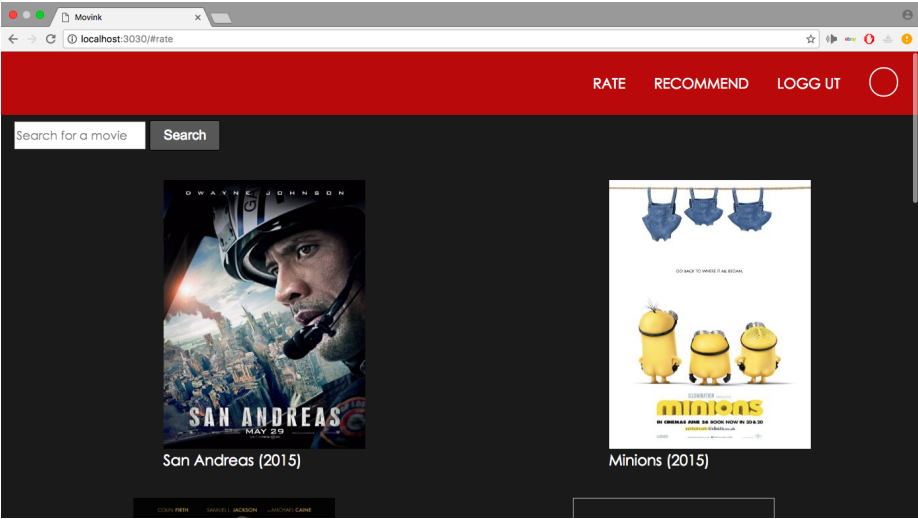


Figure 4.11: Rating page

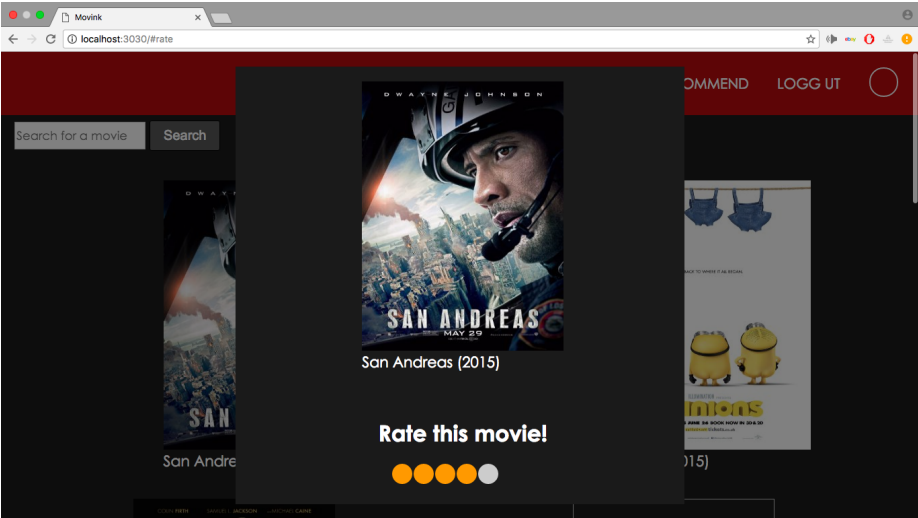


Figure 4.12: Rating San Andreas

## Recommendation

This page will give the user recommendations of movies. The user can filter by either year, time length, genre, or time length and genre. More detailed explanation of the filtering is already explained in section 4.5.2. Clicking on a particular recommended movie will present the user a window containing more details about the movie. The information that is given to the user is main actors, plot, IMDb<sup>8</sup> rating and the estimated rating personalized for the user. This feature was added in phase two when we learned that many people actually prefer to do their own research. This includes finding out which actors are in the movie and the plot of the movie.

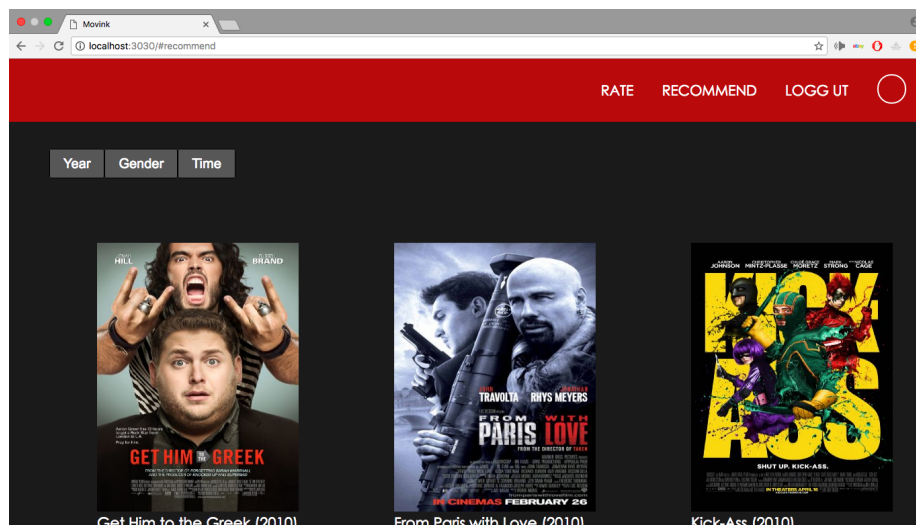


Figure 4.13: Recommendation page

---

<sup>8</sup><http://www.imdb.com>

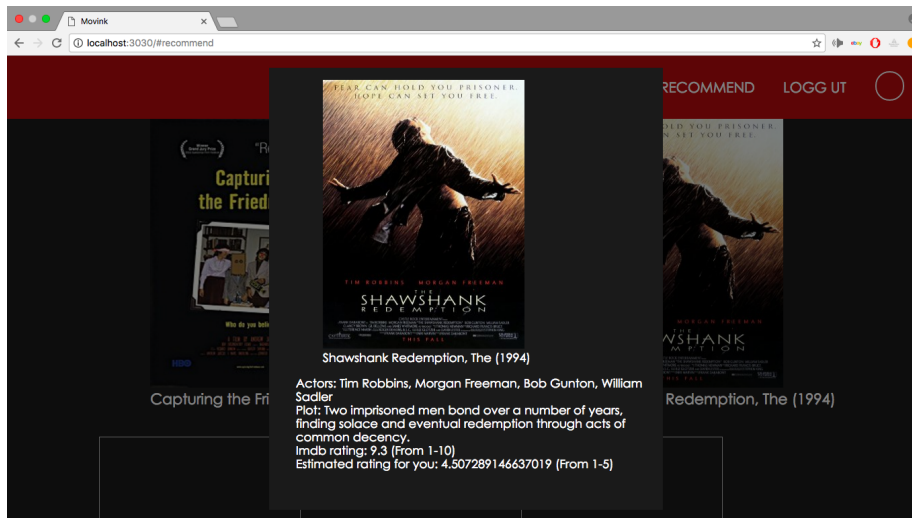


Figure 4.14: Details of a recommended movie

## Profile

Initially, the profile page was created to provide the user a even more personalized (adding more features to Factorization Machine) recommendation if we were to get a data set with the relevant features. It could also be used to create the data set with the relevant features. However, after the interviews, we learned that socializing was a feature that was wanted. Therefore, the profile page became more than we initially intended it to be. It became a page where you could search for other users. You can find out if a certain user is similar to you and look at their ratings. This feature lets you recommend yourself a movie. This feature has potential to grow to something even bigger. More of this will be discussed in Chapter 7: Conclusion and future work.

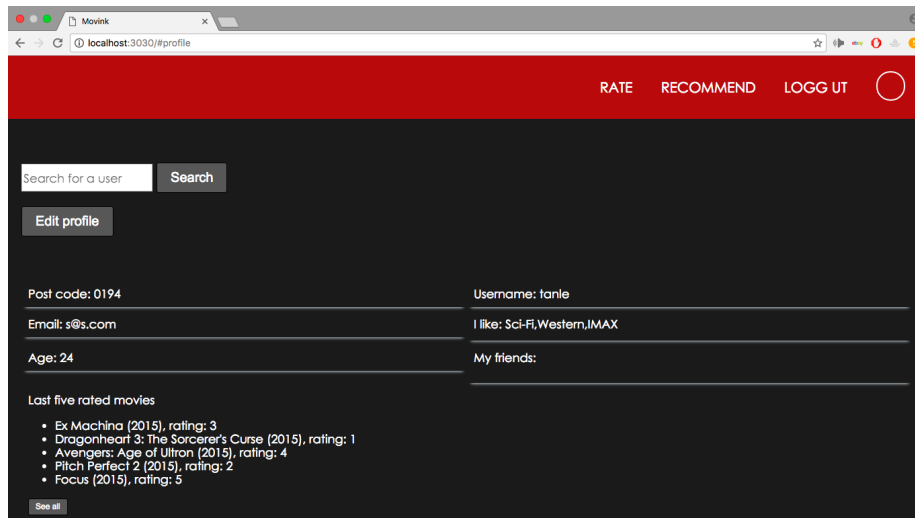


Figure 4.15: Current user profile

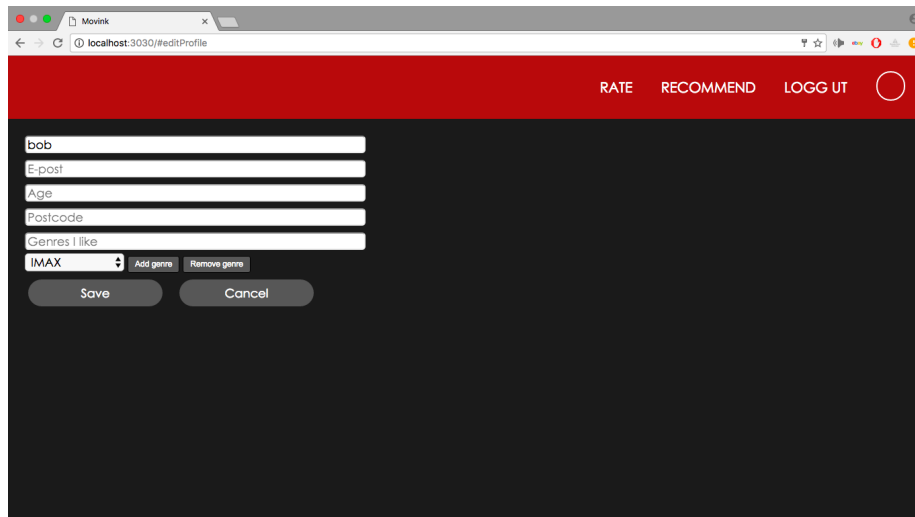


Figure 4.16: This page lets you edit your profile, including adding and removing genres you like

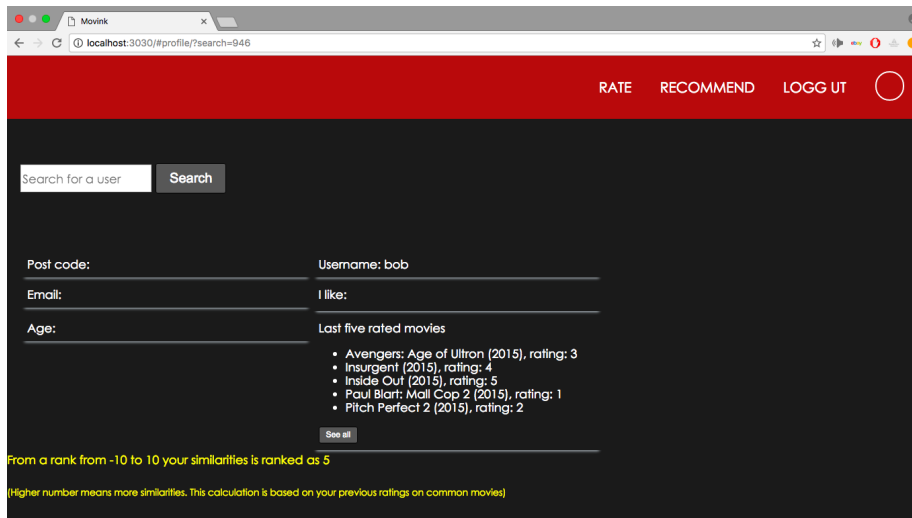


Figure 4.17: Search of user bob. We can see what movies he has rated and other information about him. User tanle and user bob has a similarity of 5 (between -10 to 10, calculated using the Pearson Correlation

## 4.8 Technology

So far, Python, Django, Django REST Framework, OMDb API and Backbone have been mentioned. However, there are some other noteworthy technologies that have been used. They will be described here.

### 4.8.1 PostgreSQL

For the database we had a couple of choices. Since our web framework of choice is Django, we've found that MySQL, PostgreSQL and SQLite are the ones that are most relevant as they are best documented on their official web page [2]. SQLite however, is not viable because it does not work with applications where multiple clients share the same database. Both MySQL and PostgreSQL would have worked fine. PostgreSQL was chosen due to the known limitation

of MySQL and we want a prototype that is open for any feature in the future.

### 4.8.2 GitHub

GitHub<sup>9</sup> is a web-based Git repository hosting service. This is used mainly for sharing the prototype to anyone who wants to use it.

### 4.8.3 scikit-learn

scikit-learn is a free software machine learning library for the Python. This library comes in handy when doing recommendation calculations. PyFM uses it, we also use it to create the needed matrices for our Factorization Machine model and evaluate the accuracy of the recommendations with its mean squared error function.

---

<sup>9</sup><https://github.com>

## Chapter 5

# Evaluation

This section contains the evaluation of the prototype. The evaluation consists of two phases. Interviews were made in phase one to improve our prototype. Then an evaluation of the accuracy and speed of our approach was done in phase two. A comparison of a user-based collaborative filtering and Factorization Machine will also be described.

### 5.1 Phase one: The interviews

In phase one of the evaluation we did a round of interviews. There were seven participants in total. The original interview can be found in the appendix. In this section we will briefly go through each participant's answer. The anonymous participant is called A, and the interviewer is called I (in the original interview that has been included with this thesis). The goal of this phase is to collect ideas that will help improve our prototype. The questions were made as the participant answered. Only the first question "Bruker du anbefalingssystem slik som denne (our phase one prototype) for å få anbefalt filmer? For eksempel Netflix?" was predefined.

### 5.1.1 Anonymous A

Anonymous A trusts people more than computers. Uses trusted websites like IMDb to get an overall idea of the movie.

### 5.1.2 Anonymous B

Anonymous B uses recommendation systems. B also trusts recommendations made by friends over those made by computers/algorithms.

### 5.1.3 Anonymous C

Anonymous C uses recommendation systems (Netflix) , but does not rate movies. This can cause bad recommendations. C trust friends more, but only those who have the same taste. C thought the idea of social feature would be cool.

### 5.1.4 Anonymous D

Uses recommendation systems (Netflix), but does not rates movies often. This can cause bad recommendations. Trusts friends more, but only those who have the same taste. D likes to do some research on their own, like finding out the actors of the recommended movie before actually watching it.

### 5.1.5 Anonymous E

Anonymous E does not use recommendation systems because they does not trust them at all. E trusts friends more. E usually chooses interesting movies she randomly finds.

### 5.1.6 Anonymous F

Anonymous F uses Netflix recommendations to get exposed to movies, then do research with IMDb. Like many others, F also trust friends more, depending on their taste.



### 5.1.7 Anonymous G

Anonymous G mainly gets recommendations from YouTube reviews from a known youtuber. Also likes the “Next” funksjon of youtube, where they recommend you more movies based on what you are watching now and have watched in the past.

### 5.1.8 Conclusion

From the interviews, we can see that most people do not trust algorithms. Although they do not know how the algorithms work they do not trust it. It does not matter if the recommendation algorithms has been improved significantly the last decade, they still do not trust them. The reason could be that they think of algorithms as machines without feelings, and can not replicate the human brain and therefore prefer recommendations from friends, because it feels more personal. Recommendation algorithms based on collaborative filtering also requires a good amount of ratings for it to be accurate. Many might try the recommendation system with too few ratings and therefore get disappointed with the recommendation. There are also several cases where people want to do the research themselves, by using sites like IMDb. Based on this conclusion, two new functions were added to our prototype. Instead of having the user first get the recommendation from the application, then do their own research from IMDb, we chose to include the data about actors involved, plot and ratings from IMDb in the recommendation. Since people prefer to get recommendations from people that have the same taste as them, the profile page was extended to a page where you can search for other users. When you have found a user, the similarity will be calculated and you can see what the other person has rated. You can then pick a movie with high ratings if the similarity is high enough for you. For the final product see section 4.7: Prototype.

## 5.2 Phase two: Performance

This is the second phase of our evaluation. The final prototype has been evaluated by its speed and accuracy. This section will describe those results. In previous work we experienced how slow memory-based filtering was, and therefore wanted to use a model-based filtering. The reason why specifically Factorization Machine were chosen is explained in section 4.2: Chosen collaborative filtering method. However, if Factorization Machine was not fast or accurate enough, then another approach would have been considered. The upper limit of how many seconds the approach should take was chosen to be 0.5 seconds. The reason can be found here [4]. The article describes how slow a page should load. Slow loading time can of course be caused by other things like upload/download speed. However, we do not want the slow loading to be caused by the algorithm itself. For the accuracy we want something that is at least as accurate as memory-based filtering.

### 5.2.1 Data set

The data set we are using was made available on Movielens webpage the 11. January of 2016. It contains 105339 ratings, which were split into a training set and a test set. 85% of the data became training and the last 15% became tests. The reason behind the low percentage of test sets (15800 tests) is because memory-based filtering could spend a couple of seconds or more to predict one rating, which could take days if the amount of test cases became too big. In fact, with 15800 tests, it took roughly 29 hours (105610.86 seconds) to compute all 15800 predictions. The original data set (ratings.csv), training set (train.csv) and test set (test.cs) are all being included with the thesis.

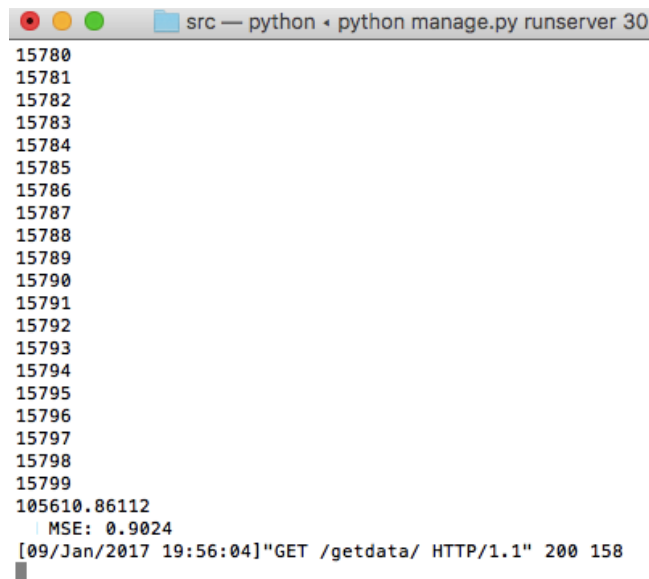
### 5.2.2 Environment

The environment of the tests has an effect on the evaluation, especially the speed of the algorithms. The tests have been done on a macOS Sierra with 1,6

GHz Intel Core i5 processor and 4 GB 1333 MHz DDR3 memory.

### 5.2.3 User-Based Filtering based on Pearson Correlation

As you can see in figure 5.1, the MSE of Pearson User-based filtering using the data set mention above, is 0.9024. This is the accuracy that Factorization Machine has to beat.



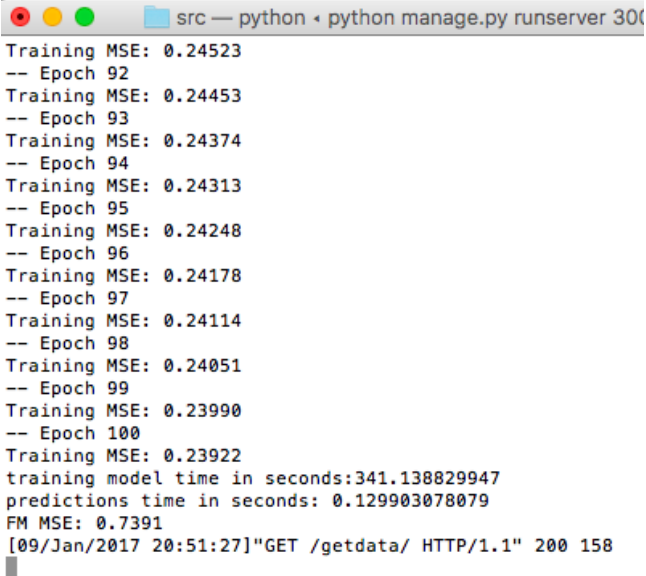
```
src — python • python manage.py runserver 30
15780
15781
15782
15783
15784
15785
15786
15787
15788
15789
15790
15791
15792
15793
15794
15795
15796
15797
15798
15799
105610.86112
MSE: 0.9024
[09/Jan/2017 19:56:04]"GET /getdata/ HTTP/1.1" 200 158
```

Figure 5.1: Performance of user based filtering

### 5.2.4 Factorization Machine with SGD

Factorization Machine was both fast and accurate. It did spend some time to train the model, depending on how many epoches we wanted it to do. For our prototype and test we chose to do 100 epoches, which means that SGD iterates through the whole training set 100 times. This training took 341 seconds. However, this waiting time is not something that the user needs to be experiencing. This should be done when the server is down and periodically (for example once a week). The speed of the actual prediction is much faster. To predict all 15800 test cases, it needed 0.1 seconds. The accuracy, the MSE with this training og

tests, was 0.74. These results show that Factorization Machine is more accurate than user-based biltering and faster than the limit we had set. It is therefore viable for our prototype.



```
src — python ◀ python manage.py runserver 3000
Training MSE: 0.24523
-- Epoch 92
Training MSE: 0.24453
-- Epoch 93
Training MSE: 0.24374
-- Epoch 94
Training MSE: 0.24313
-- Epoch 95
Training MSE: 0.24248
-- Epoch 96
Training MSE: 0.24178
-- Epoch 97
Training MSE: 0.24114
-- Epoch 98
Training MSE: 0.24051
-- Epoch 99
Training MSE: 0.23990
-- Epoch 100
Training MSE: 0.23922
training model time in seconds:341.138829947
predictions time in seconds: 0.129903078079
FM MSE: 0.7391
[09/Jan/2017 20:51:27]"GET /getdata/ HTTP/1.1" 200 158
```

Figure 5.2: Performance of Factorization Machine

# Chapter 6

## Discussion

Evaluating recommendation systems and their algorithms is not easy. Different algorithms may be better or worse on different data sets. Some algorithms work well when the data set contains many more users than items. However, they might not be appropriate in a domain where there are many more items than users. There also exist other properties for data sets, like rating density, rating scale, and more [15]. This section will discuss the system, the results and their validity.

### 6.1 Data set

Movielens keeps updating the data set that has been used in this thesis. They do not archive all versions, and that is why the data set has been included with the thesis, with the same license usage. There are many ways to split a data set into training- and test set. The way it has been done here includes adding a random data point from the data set to either training or test. A problematic scenario (although unlikely) would be that a certain user ID is not in the training set, meaning there will be at least twenty test cases of a user without any training. Those predictions will be inaccurate because of the cold start problem mentioned earlier in this thesis. Different kinds of splits can be

done to have a more varied evaluation. We will discuss this matter further in Chapter 7: Conclusion and future work.

## 6.2 Factorization machine

Factorization Machine has only been evaluated with two features, userID and movieID. This does not reflect the strength and potential of FM. FM with a data set that contains more features needs to be evaluated to see if that actually makes the predictions more accurate. One obvious drawback with the current data set is that even if users rate new movies, these ratings will not affect new predictions before new training has been done. This means that ratings done between two training sections, will not be considered during that period. A possible solution to this is to have a feature called "last movie rated". Although this is not as efficient as retraining the model, it might suffice.

The approach we used has an accuracy of MSE: 0.7391, giving RMSE of 0.8597. The value itself is very closed to the winning algorithm of the Netflix competition in 2009, which has a RMSE of 0.8567 [6]. However, this does not mean that they are equally good. The data sets that has been used are so different that they are not comparable. However, the value itself is good. If this evaluation had used the same data sets as the netflix contest, and got the same value then they would have been equally good. A bigger data set needs to be used to compare the approach with the bigger movie recommendation systems like Netflix. From the results in this thesis the only conclusion regarding the accuracy we can make is that it is more accurate than our implementation of user-based collaborative filtering. However, choosing different values for similarity threshold can also impact the accuracy of the algorithm. Factorization machine also has different ways to train the model and each way will make the model predict different ratings. We do not know if SGD is the best way (both in speed and accuracy) to train the model.

## 6.3 Context

### 6.3.1 Genre

When a user adds a genre that they are tired of, then all movies that contain that genre will get removed from the recommendations completely. This is not always appropriate. Sometimes the user might still want to watch a movie of a genre they are tired of if they think the movie is good enough. However, finding a good way to weigh it is not easy. An alternative way of doing it, is to allow the user to set a minimum threshold, where only movies with genres the user is tired of with a lower predicted rating than this threshold is discarded.

### 6.3.2 Time

If a user does not have more time than 90 minutes, then all movies longer than 90 minutes should not be considered. However, factoring in time in the recommendation has another issue. We will describe this issue with an example. Imagine a scenario where the user has set the time limit to 120 minutes. There exist movies A, B, C with time lengths 60 minutes, 60 minutes and 100 minutes respectively. The recommendation system predicts the rating of movie A: 4, movie B:4 and Movie C: 5. The question now is whether it is better to watch the best movie, but only one or to watch two good movies. By adding time to the context it can be interesting to not only find the best fit, but also the best combination of movies to watch within the time limit.

## Chapter 7

# Conclusion and future work

### 7.1 Conclusion

In this thesis, a proof of concept prototype of a Context Aware Recommendation System was created. The prototype tested the Factorization Machine in practice and contained a social feature that is uncommon to most recommendation systems. The main goal of the thesis was not to improve the accuracy of the state of the art algorithms, but to improve the recommendation system by finding what people think it lacks. Interviews were held to find out these features and added to the prototype. These features were relevant data from IMDb and social feature where users can find each other. Data from IMDb were added because the interviews showed that people tend to go there to do their own research. The social feature was added to increase the trust of the prototype. The interviews showed that the main reason of why people did not use recommendation systems was because they did not trust computer recommendations. They trust their friends more. A social feature was therefore created.

The goal of the prototype was to recommend movies to a user taking mood/-genre and time into account. To evaluate the accuracy of the system, MSE was used. To evaluate the speed of the recommendation, Python time module were used. The evaluations showed that Factorization Machine gave accurate and



fast predictions. It beats user-based filtering both in speed and accuracy with a good margin.

## 7.2 Future work

From this point, there are many directions one can take. These are some of them:

### **Better GUI**

The prototype needs a better design and the GUI needs to be responsive. Which means that the GUI should change and adapts to the device, so it looks nice independent of the screen of the device.

### **Factorization Machine**

Evaluating Factorization Machine with data with more features to see if it gives better predictions. We have not yet seen the full potential of FM.

### **Social feature**

The social feature that was added to the prototype can be extended further. More research needs to be done to find out how and what should be extended. Should it be able to add friends, like on Facebook? Or follow an user like Instagram or Twitter? There are many social platforms out there that have features that can added to the prototype.

### **Evaluation**

A more comprehensive evaluation of the accuracy of the algorithms should be done. Like it was described earlier, only one randomized 85/15 split were made of a data set of approximately 100 000 ratings to evaluate its accuracy. A larger data set would most likely give a more correct MSE/RMSE value. Splitting it

up in several ways and evaluate each of them should also provide higher validity of the results.

# Appendices

# Appendix A

## Interviews

### Anonymous A

I - Bruker du anbefalingssystem slik som denne (our phase one prototype) for å få anbefalt filmer? For eksempel Netflix?

A- Sjeldent. Føler ikke at slike systemer ”treffer” riktig noe særlig ofte.

I - Så du stoler mer på vennene dine enn datamaskiner? Eller hva gjør du for å finne en film? Beskriv et par scenarior

A - Dersom jeg blir anbefalt en film av en venn (deg, Lars, etc.) er det mye mer sannsynlig at jeg vil se på den/sjekke den ut enn om jeg skulle blitt anbefalt en film av en eller annen algoritme.

For å se filmer hjemme pleier jeg å sjekke topp 100 på torrent sites. Ser jeg noe som ser interessant ut så sjekker jeg filmen på IMDB for å sjekke plot/rating/et par reviews for å forsikre meg at det ikke er helt katastrofe.

For filmer på kino er det ofte reklamer (TV/YouTube) som får meg til å gå. F.eks. Arrival og Inferno atm vil jeg se. Hadde ikke hørt om noen av de to filmene om det ikke var for reklame på TV/YouTube.

I - Hva er det du ser på når du sjekker imdb?

A - Plot/rating/reviews/skuespillere

**Anonymous B**

I - Bruker du anbefalingssystem slik som denne (our phase one prototype) for å få anbefalt filmer? For eksempel Netflix?

A - Tror den står på automatisk (Netflix recommendation) og har ikke manuelt tatt den av, så ja bruker den på en måte da.

I - Men synes du det er en viktig funksjon av netflix?

A - Synes det er veldig greit at de har den funksjonen egentlig, fordi det ikke alltid er sånn at jeg vet hva jeg vil se på og da er det greit å få en anbefaling på hva jeg kanskje kan komme til å like. Stemmer ikke nødvendigvis alltid, men av og til har de noen gode anbefalinger.

I - Hvem stoler du mest på? Anbefalingssystem eller venner?

A - Det kommer helt an på, men går for en film jeg har fått anbefalt av venner som regel. Men har jeg ingen spesifikke anbefalinger fra venner så bruker jeg anbefalingssystemet for å finne noe

I - Rater du filmer på netflix?

A - Ja, det gjør jeg

I - Hva syntes du om anbefalingene på netflix?

A - De er greie nok. Noen er anbefalinger av filmer som jeg alt har sett og det synes jeg er litt teit da, vil jo ha anbefalinger på andre filmer. Men utover det så er de greie nok. Noen ganger anbefaler de noen bra og andre ganger ikke

**Anonymous C**

I - Bruker du anbefalingssystem slik som denne (our phase one prototype) for å få anbefalt filmer? For eksempel Netflix?

A - Ja, ganske ofte

I - Hva bruker du?

A - Pleier å se på flere nettsider, siden en side fort kan bli subjektivt IMBd. Også søker jeg bare reviews av filmer på net, men det er mest IMBd og netflix det går på

I - Så du bruker imbd til å se reviews? Beskriv en scenario der du bruker imdb.

A - Jeg søker filmen, ser på stjerner den har fått. også går det an å se på kategorisering. Sånn om hvor mange menn og kvinner som har stemt osv Eller aldersgrupper.

I - Stoler du anbefaling fra slike systemer (som netflix) eller fra venner mest?

A - Det spørs litt. Jeg stoler jo ikke på alle venner sine anbefalinger. Jeg velger jo å stole på de som har samme type smak som meg. Har mye med preferanse å gjøre og det samme gjelder vel også nettet.

I - Men du har venner som har samme smak som deg? altså det finnes folk som du heller spør enn å bruke netflix?

A - Det finnes et par venner som jeg heller spør ja og om de ikke har noen ideer, så sjekker jeg ut nettet.

I - Hva hvis det fantes en anbefalingssystem som hadde en sånn sosial feature, type du kan se hva dine venner liker og få film tips utifra det.

A - Det hadde jo vært kult

I - Rater du filmer på netflix?

A - Nei, ganske sjeldent

### **Anonymous D**

I - Bruker du anbefalingssystem slik som denne (our phase one prototype) for å få anbefalt filmer? For eksempel Netflix?

A - Yes jeg bruker som regel "nylig lagt til" hvis jeg ser på filmer som er gitt ut, ellers så ser jeg på hva som går på kino hvis det er nye filmer.

I - Hva er det som får deg til å velge en spesifikk film?

A - At den virker spennende, gjerne kjente gode skuespillere

I - Rater du på netflix?

A - Kun hvis det er superbra bra/serie

I - Stoler du mer på anbefalingssystemer eller på folk?

A - Spørs om man har samme smak, så spørs veldig hvilken type film og person. Har vi samme smak så tar jeg venn-tips fremfor anbefalingssystemer. Sjekker imbd noen ganger

I – Beskriv et scenario der du skal finne en film å se

A - Bladde gjennom mye rart, så først på bildet av filmen/Serien, så sjangeren, så stjerner, så kjente skuespillere! Det er film med bra stjerner og gerhard butler - han var grunnen til jeg valgte så der har du den!

### **Anonymous E**

I - Bruker du anbefalingssystem slik som denne (our phase one prototype) for å få anbefalt filmer? For eksempel Netflix?

A - Jeg bruker som regel popcorn time

I - Anbefaler den filmer til deg? eller velger du selv

A - Det dukker opp bare nye filmer, så bare blar jeg, jeg har aldri trykket på anbefalt, tror ikke det finnes

I - Ok. Så synes du at det er nyttig med anbefalingssystemer? Eller foretrekker du heller anbefalinger fra venner?

A - Anbefalinger av venner

I - Hva er grunnen til at du ikke bruker anbefalingssystemer? Stoler ikke på dem eller? A – Føler ikke den treffer min smak

I - Så du stoler ikke på den altså. Så du hadde brukt anbefalingsfunksjonen hvis du hadde trodd at de treffer smaken din?

A – Ja

### **Anonymous F**

I - Bruker du anbefalingssystem slik som denne (our phase one prototype) for å få anbefalt filmer? For eksempel Netflix?

A - Ja, noen ganger

I - Hva syns du om det? hva er det som er så bra med det?

A - Si jeg har sett thrillere en periode, fordi jeg har fått dilla på trillere. Så dukker det opp en triller jeg aldri har hørt om før. Så ser jeg at den har bra rating kanskje. Og da tar jeg sjansen og ser den Andre ganger bare fordi jeg ikke orker å lete så lenge etter en film på nett. Men ser den har anbefalt en del

filmer. Så ser jeg en av dem

I - Rater du på netflix?

A - Jeg gjorde det før, men ikke så mye nå lenger. Men jeg rater på imdb. Grunnen til at jeg sluttet å gjøre det på Netflix var nok fordi jeg stoler mest på IMDB av en eller annen grunn. Så begynte å gå dit for å se på ratings og rate. Jeg hadde en extension til Netflix i Chrome som gjorde at jeg kunne se IMDB rating på filmer i netflix. Så å stole på rating-kilden betyr en del for meg.

I - Så når du bruker anbefalingen til Netflix så går du ikke etter ratings, men du bruker det for å få oversikt over potensielle filmer også sjekker du ratings på imdb? Eller hvordan er et typisk scenario?

A - Akkurat sånn egentlig

I - Stoler du mer på anbefalinger fra venner eller fra systemet?

A - Tror jeg stoler mer på venner. Men det spørs nok hvem det kommer fra.

### **Anonymous G**

I - Bruker du anbefalingssystem slik som denne (our phase one prototype) for å få anbefalt filmer? For eksempel Netflix?

A - Imdb og youtube

I - Men før du bruker IMDb så må du jo vite hva du skal søke om hvordan får du vite om filmen?

A - Bare youtube, trailere som får mange views, og reviewers like jeremy jahns og chris stuckman.

I - Men er du subscriba på en channel eller bare dukker det opp recommendations

A - Sånn "På vei opp" greie på youtube iblant

I - Foretrekker du anbefaling fra et system eller fra venner?

A - Youtube er best for meg.

I - Hvorfor er det så bra

A - Det er bare det mest populære som dukker opp. Generelt youtube er best fordi du bare kan høre på en video liksom.

I - Så egentlig anbefaling av en youtube bruker?



A – Ja, alt ligger på et sted liksom

I - Så det er ikke youtube sin anbefalingsalgoritme du liker, men en spesifikk kanal på youtube som anbefaler filmer

A- Eller jo ,det også. Den “anbefalte” funksjonen på høyresida er nice.

# Appendix B

## Code

All the code produced while developing the prototype presented in this thesis is available at <https://github.com/tanql/RecommendApi> for the back end and <https://github.com/tanql/RecSystem> for the front end. It is released under a MIT License. The data set is also provided in the back end repository. See below for the usage license of the data set. The code will also be uploaded as an attachment to the thesis.

### B.1 Data set Usage License

”Neither the University of Minnesota nor any of the researchers involved can guarantee the correctness of the data, its suitability for any particular purpose, or the validity of results based on the use of the data set. The data set may be used for any research purposes under the following conditions:

The user may not state or imply any endorsement from the University of Minnesota or the GroupLens Research Group. The user must acknowledge the use of the data set in publications resulting from the use of the data set (see below for citation information). The user may redistribute the data set, including transformations, so long as it is distributed under these same license conditions. The user may not use this information for any commercial or revenue-bearing

purposes without first obtaining permission from a faculty member of the GroupLens Research Project at the University of Minnesota. The executable software scripts are provided "as is" without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of them is with you. Should the program prove defective, you assume the cost of all necessary servicing, repair or correction. In no event shall the University of Minnesota, its affiliates or employees be liable to you for any damages arising out of the use or inability to use these programs (including but not limited to loss of data or data being rendered inaccurate)."

# Appendix C

## API documentation

### **api/register**

- Type: POST
- Data: Username (string) , password (string) and repeated password (string)
- Description: Creates a new user to the database with the required data (username, password and repeated password)

### **api/login**

- Type: POST
- Data: Username (string) and password (string)
- Description: Log in the user if username and password is correct.

### **api/logout**

- Type: GET
- Description: Log out the current user.

**api/users**

- Type: PUT
- Data: Interests (array, optional), age (int, optional) and postCode (string, optional)
- Description: Edit the user and updates in the database according to the data.

**api/users**

- Type: GET
- Data: User: username (string) and email (string), userID (int), age (int), postCode (string), interests (Array of genres) and ratedMovies (Array of rated movies)
- Description: Return the profile of the current logged in user.

**api/users/?user={id}**

- Type: GET
- Data: User: username (string) and email (string), userID (int), age (int), postCode (string), interests (Array of genres), similarity (float) and ratedMovies (Array of rated movies)
- Description: Return the profile of id.

**api/rate**

- Type: POST
- Data: MovieID (int) and ratingValue (int)
- Description: Add a rating to the database

**api/rate/?page={id}**

- Type: GET
- Data: count (int, amount of movies in total) and movies (Array of movies, the name and genres, movieID and release year)
- Description: Return a list of movies.

**api/movies**

- Type: GET
- Data: Movies (array)
- Description: Return a list of movies rated by the current logged in user.

**api/recommend/{filtertype(optional)}/?page={page}**

- Type: GET
- Data: Array of recommended movies
- Description: Return a list of recommended movie based on filtering type.

# Glossary

**AJAX** Asynchronous JavaScript And XML. 40

**GUI** Graphical user interface. 39, 40, 54, 71

**MSE** Mean Squared Error. 31, 70

**SGD** Stochastic Gradient Descent. 65, 68

# Bibliography

- [1] Django. <https://docs.djangoproject.com/en/1.10/>. Accessed: 2016-12-20.
- [2] Django database. <https://docs.djangoproject.com/en/1.10/ref/databases/>. Accessed: 2016-12-20.
- [3] Django rest framework. <http://www.django-rest-framework.org>. Accessed: 2016-12-20.
- [4] How slow is too slow in 2016? <http://www.webdesignerdepot.com/2016/02/how-slow-is-too-slow-in-2016/>. Accessed: 2016-11-10.
- [5] Model-view-controller. <https://msdn.microsoft.com/en-us/library/ff649643.aspx>. Accessed: 2016-12-20.
- [6] Netflix winner. [http://www.netflixprize.com/community/topic\\_1537.html](http://www.netflixprize.com/community/topic_1537.html). Accessed: 2017-01-10.
- [7] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions on*, 17(6):734–749, 2005.
- [8] Sudipto Banerjee and Anindya Roy. *Linear algebra and matrix analysis for statistics*. CRC Press, 2014.



- [9] John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, UAI'98*, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.
- [10] Robin Burke. Hybrid recommender systems: Survey and experiments.
- [11] JohnS Breese DavidHeckerman CarlKadie. Empirical analysis of predictive algorithms for collaborative filtering. *Microsoft Research Microsoft Corporation One Microsoft Way Redmond, WA, 98052*, 1998.
- [12] Michael D Ekstrand, John T Riedl, and Joseph A Konstan. Collaborative filtering recommender systems. *Human-Computer Interaction*, 4(2):81–173, 2010.
- [13] Shengbo Guo. *Bayesian Recommender Systems: Models and Algorithms*. PhD thesis, Australian National University, 2011.
- [14] By Brad Harris. Factorization machines: A new way of looking at machine learning, Nov 2015.
- [15] Jonathan L Herlocker, Joseph A Konstan, Loren G Terveen, and John T Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53, 2004.
- [16] Alan R Hevner. Design science in information systems research. *MIS Quarterly*, 28(1):75–105, 2004.
- [17] Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. *Recommender systems: an introduction*. Cambridge University Press, 2010.
- [18] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. 2008.
- [19] Yehuda Koren, Robert Bell, Chris Volinsky, et al. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

- [20] Helge Langseth. Bayesian networks for collaborative filtering. In *Proceedings of Norwegian Artificial Intelligens Symposium*, pages 67–78, 2009.
- [21] Pasquale Lops, Marco de Gemmis, and Giovanni Semeraro. *Content-based Recommender Systems: State of the Art and Trends*, pages 73–105. Springer US, Boston, MA, 2011.
- [22] Prem Melville and Vikas Sindhwani. Recommender systems. In *Encyclopedia of machine learning*, pages 829–838. Springer, 2010.
- [23] Frank Meyer. Recommender systems in industrial contexts, 2012.
- [24] Andrew Ng. Supervised learning, lecture notes. <http://cs229.stanford.edu/notes/cs229-notes1.pdf>, Fall 2012. Accessed: 2017-01-04.
- [25] Michael J. Pazzani and Daniel Billsus. *Content-Based Recommendation Systems*, pages 325–341. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.
- [26] Steffen Rendle. Factorization machines. In *2010 IEEE International Conference on Data Mining*, pages 995–1000. IEEE, 2010.
- [27] Nafiseh Shabib. *Novel Approaches to Group Recommendation*. PhD thesis, NTNU, 1 2015.
- [28] Le Tan. A context aware group recommendation system for movies. Specialization project in Computer Science, Department of Computer and Information Science, Norwegian University of Science and Technology, 2016.
- [29] Wolfgang Woerndl, Christian Schueller, and Rolf Wojtech. A hybrid recommender system for context-aware recommendations of mobile applications.