Optimizing Movie Recommendations in a Social Context

Programming and problem solving (P2)

Project Report

Group A313b

Aalborg University
Computer Science

AALBORG UNIVERSITY
STUDENT REPORT

Computer Science
Aalborg University
http://www.aau.dk

Title:
Optimizing Movie Recommendations in a Social Context

Project Period:
Fall Semester 2018

Project Group:
A313b

Participant(s):
Daniel Thomsen
Niki Zakariassen
Simon Steiner

Supervisor(s):
Peter Dolog

Page Numbers: 82

Date of Completion:
May 28, 2018

Abstract:

This report examines the opportunities of placing a recommender system into a social aspect, that allows it to recommend both movies and friends based on similar movie interests. To make these recommendations a hybrid system between a collaborative k-nearest neighbors algorithm and a content-based artificial neural network has been implemented. The kNN algorithm is implemented with Pearson's correlation and ANN uses mini-batch to train the network. To train and verify the implementations, the MovieLens dataset was used, which was separated into five sections that allowed the use of 5-fold cross validation to ensure the comparability of the results to other implementations. The predictions and user profiles are visualized through a graphical user interface that outlines the basic principle of how such a platform would work.

Contents

Introduction

People are expected to make a number of different choices online every day, some of which might be easier if they are in some way limited or assisted [1]. Therefore it might be beneficial to find ways to guide the user towards relevant and valuable information that might otherwise be overlooked. An automatic and intelligent recommender system could thereby be a very helpful tool for easing the decision-process by suggesting choices to the user.

An effect of the digital age is a decrease in the amount of direct contact between people, and an increased ability to communicate with people globally over the internet [2]. This specific feature is a product of several tech-giants like Facebook, Instagram and Snapchat. Their services specialize in allowing people to communicate easily and connect with people across the globe [3].

Compared to meeting people at sports and hobby clubs, you do not necessarily encounter and befriend people online based on specific common interests. For example, oftentimes the act of befriending someone may be based on temporary physical proximity or simply pure happenstance. Other times the platforms such as Facebook are primarily being used to cultivate friendships that have already been made in the real world instead of making new friendships based on similar online activities [4]. The ability to personalize social encounters online according to peoples' interests and hobbies could therefore be a more beneficial way to create social engagement.

The fact that a Netflix subscriber on average spends 93 minutes on Netflix and that a person spends 135 minutes on various social media platforms every day opens up the possibility that combining these services could potentially allow for a more interactive and personalized way of meeting new people and simultaneously serving up relevant targeted information based on shared interests [5][6]. This idea was utilized by Facebook —among others —when they created Facebook Watch for the purpose of personalizing recommendations and allowing the discovery of new video content through friends [7]. For the purpose of creating the best possible platform, a case study about existing solutions like Facebook Watch is necessary to deviate and improve upon aspects of those platforms.

For a project that looks to personalize social engagement based on common interests, it is crucial to identify the users and other various relevant stakeholders to accurately identify their needs and interests. Predictably, doing this analytic work should increase the chance of developing a program that will be of use to the users.

Chapter 1

Societal context

A central part of solving a problem is being aware of what effect it might have in a broader societal context and thereby narrow down the problem to a few central points. This section will consequently uncover some of the societal problems which the project will try to resolve, by examining the key topics of social networking, choosing the right product and a recommender system's potential effect on company revenue.

1.1 Initial Problem

1.1.1 Social networking

As previously mentioned, people spend an average of 135 minutes on various social media everyday [6], and according to a project named "Shaping the Future Implications of Digital Media" the digital media has had a positive effect in the work sphere, but had the least positive effects in the social and personal spheres [8]. One of the 5 least positive impacts digital media have had, has been on peoples ability to find a significant other. [8]

It is well known that people with similar interests tend to form groups and cliques together, and since the digital development where people on social media like Facebook, tend to like pages and groups that mirror their own interests, it has become easier than ever to group people [9].

1.1.2 Individual choice

According to Finder.com, Netflix currently has a total of about 5000 titles in its library in the US [10], and having to decide between such an amount of different titles can be both difficult and time-consuming, since you might have to go through a lot of titles before you pass one which might fit your needs and interests. A recommender system can help alleviate this problem by suggesting a limited number of suitable and personalized items for the user, and thereby ease the decision-making process.

Another of the 5 least positive effects digital media has had, was on our stress levels. People reported an increasing amount of stress by using digital media [8] and according to an article in *The Guardian*, having too many choices does not help prevent stress, but instead increases stress in our daily lives [11]. By reducing the amount of choices people have to make throughout the day, the project group might help combat everyday stress, or at least ease the decision-making process and reduce unnecessary cognitive workload. [11]

1.1.3 Revenue

Trying to improve society by bringing people together based on shared interests and hobbies is beneficial in and of itself, but a company that provides such a service has to generate revenue in order to last. Luckily, a recommender system is not only helpful for its users, but also for its creators, since having a recommendation system has shown to increase upselling revenue in a range from 10% to 50% [12].

In addition to increasing revenue, a recommender system can help a company create more loyal customers by continuously catering to its users, and help form habits by always serving accurate user-specific content [12].

1.1.4 Sub-conclusion

The project group aims to alleviate the identified problem areas with the help of a software solution.

On the basis of the initial analysis and discussion, the following problem statement is formulated:

People face a general and persistent problem relating to information overflow in their everyday life, making decision-making in specific areas unnecessarily difficult. Additionally, people do not necessarily form social connections online based on common interests. This further limits the stream of relevant information users receive, and exacerbates the problem of information overflow.

1.1.5 Further problem definition

At this point, general problems have been identified relating to both information overflow, as well as the undesirable foundations on which social connections (e.g. "friendships") are oftentimes formed online. As these issues are broad and domain independent, further decisions must be made to delimit the project's scope to a more specific domain wherein a solution to the aforementioned problems can be approached.

The first step in delimiting the project scope consisted of brainstorms and discussions internally within the project group regarding numerous viable categories. Eventually, the choice fell on movies, as the numerous genres in this category allow for representation of dissimilar people with varied tastes. In addition, there is a wealth of high quality user-provided data available in this domain, since viewers have generally been willing to show their level of satisfaction with a specific movie through ratings [13].

1.2 Stakeholder Analysis

In this section an analysis of the stakeholders will be represented. The stakeholder analysis is used to describe the interests of the various stakeholders, and determine what effect they might eventually have on the finished product.

An in-depth discussion of the roles, interests and interconnectedness of the stakeholders will be presented in the following sections.

1.2.1 Developers

Role

The developers of the program (the project group in this instance) are responsible for prioritizing and subsequently accommodating the interests of the other stakeholders via the program implementation.

Program specifications

Given that the intended users of the platform are of varying technical skill level, the final program implementation must also be usable by novices. This is in line with the project catalogue, which states, "The project shall have a user friendly interface." [14]

What exactly the nebulous "user-friendly" epithet entails for the platform will be described in sections discussing and detailing program design.

1.2.2 Movie Companies, Producers and Crew

The companies own the rights to the movies, and this arguably puts them in a favorable negotiation spot where they can decide whether or not their movie should run on the platform. The platform should present the movies in a way so the film companies are satisfied, because good presentation of their movies will potentially increase their revenue [15].

In this context it is relevant to consider biases that certain recommender systems have. Specifically, popularity biases favor items that are already frequently recommended to users. This bias consequently makes it harder for other items to compete for exposure. Recommender systems thus also have to deal with ethical questions relating to fairness. Otherwise, niche movies and "hidden gems" are likely to get entirely trampled on by the most popular choices. Stakeholders in this category will predictably view such built-in recommender biases completely differently, depending on the type of movies they produce.

1.2.3 Users

The target user group of the project is —broadly —people who have a special interest in movies and want to find both new suitable movies to watch, as well as like-minded individuals to exchange ideas and experiences with.

Movie selection

To accommodate the users' interest it is necessary that the selection of movies needs to be broad enough to provide a selection of movies to a heterogeneous group of users. This will also improve the recommendation and the precision of the recommended movies. A broad selection of movies increases the chance that the platform can provide suitable recommendations for the disparate user preferences.

Social needs and engagement

For a social media platform it is crucial that there are enough users of different personality types for everyone to find someone they enjoy spending time with.

It is widely accepted that people tend to be homophilic with regards to how they choose their social relations, referring to the principle that similarity between people tends to create new connections. In popular terms —"birds of a feather flock together" [16].

Additionally, people have a basic need for relatedness or a sense of belonging. People want to interact with others and feel connected to one another.

Fulfilling this need is one piece of the puzzle when it comes to increasing the users' intrinsic motivation for using the platform, meaning they will be more likely to use the platform simply because they enjoy the activity in itself, and not because their motivation is maintained by external factors, such as rewards or regulations [17]. Finally, the need for relatedness is most likely to be met if the recommender system can accurately pair people according to their interests and personalities.

Role

The users of recommender systems are partially "hostages" of the technology, given that they have little to no insight into how the underlying algorithms work. On the other hand, they have real-time influence on both input and output of the system, since the recommendations are continuously generated by their choices.

1.2.4 Governments

Although the government does not impact the project directly, laws and regulations (e.g. regarding sensitive data collection on the web) might have restrictive effects on the final product. Accordingly, the program will have to abide by such laws and manage personal information with care by using appropriate safeguarding methods, such as encryption.

1.3 Stakeholder Priority

The different stakeholders are affected by and affect the project in different degrees and can therefore be prioritized. This allows one to narrow down the list of stakeholders to only the most important ones based on the two criteria: Interest and power.

The priorities of the stakeholders are presented and displayed in figure 1.1.
As seen in Figure 1.1 the users are placed within the bottom-right quadrant. This is due to their limited power to make changes to the project and the importance of their interest, seeing as the usage of the platform is dependent on the users. The limited power that users have is through feedback where they might be able to affect the project slightly. This means that they primarily need to be kept informed and partially involve them in feedback processes.

Movie companies are mapped in the top-right quadrant of Figure 1.1. Their interests in the project and their power to change its outcome is substantial. Looking

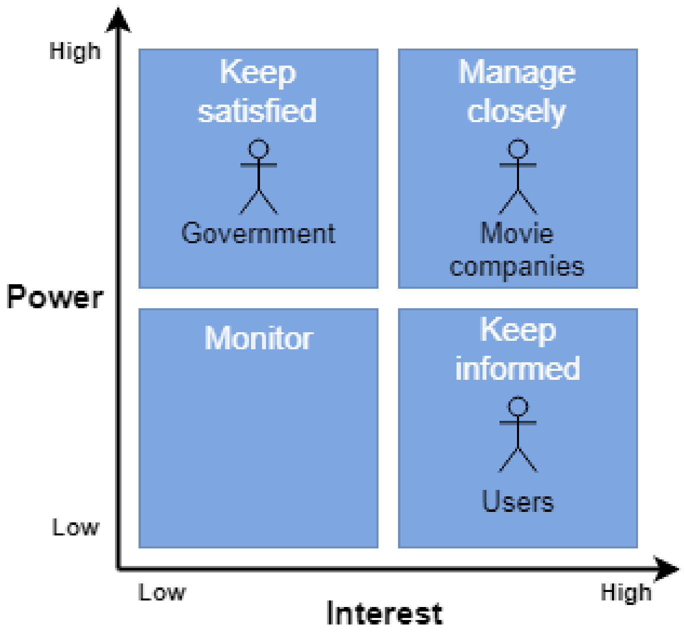


Figure 1.1: Model that maps the core stakeholders

into interests, a recommender system recommends some product which will often increase its sales. The companies creating these products thereby also have an interest in having their products recommended to the people who might be more likely to buy it. Because the platform is reliant on them allowing their movies to be shown they also have power to negotiate [15]. Furthermore regarding their power, they can set demands of how their products are displayed and used, and is therefore categorized as someone that should always be managed closely to constantly keep the interests of both parties in check. Both parties have an interest in making the relationship profitable, which means that even though the company has a lot of power that might also in some cases apply the other way around.

Government branches are in the top-left quadrant of Figure 1.1 because regulations both nationally and internationally can quickly affect the recommender system, as there is no way to negotiate or get around them. Therefore, various government entities might have a quite noticeable power on how the project unfolds, but these government entities will have little to no interest in the success of the platform. Their primary benefit from this platform would be through corporate taxes.

1.3.1 Sub-conclusion

As the analysis of this section has revealed, the interests of all stakeholders are not always perfectly aligned. Of special note is the partially antagonistic relationship between users and service providers or developers. The users want the recommender system to ease their decision-making processes, but they also wish to both spend quality time on the platform, as well as have their horizons broadened by novel content.

On the other hand, developers are financially incentivized to create programs that exploit faulty human psychology by maximizing short-term gratification and creating filter-bubbles. Splitting interests into *wants* and *needs* may thus be more beneficial to get a more accurate picture, as a user's *wants* are likely to coincide with the developer's interests, whereas *needs* are less likely to. To create a platform that caters more to the *needs* of users, one would have to retroactively collect and evaluate the users' self-reported measures of "time well spent" on the platform.

Despite having outlined some of the complicated ethical dilemmas in the stakeholder analysis, the focus of this report is primarily on using object-oriented programming to create recommendation algorithms and user interfaces. As such, solving the aforementioned issues is outside the scope of this project. However, these perspectives have still been included as the dilemmas are highly relevant to the societal context, and it is important to critically reflect on the societal impact of technology —even if said technology is seemingly innocuous.

1.4 Case study

To uncover the features and effects of successful recommender systems, and the potential lacks, a case study on Facebook and their new service 'Facebook Watch' is conducted.

1.4.1 Facebook Watch

Social media has become a central source of social interactions between people, where platforms like Facebook have started making use of this fact by combining their social network with a service for watching original video content [7]. Facebook Watch offers personalized video recommendations based on factors like previously watched content and content that is liked by your current network of friends. [7]

The social aspect of Facebook thereby focuses primarily on expanding preexisting friendship circles instead of trying to create new ones. Seen from a standpoint of the recommender system, it is presumed that you will like the same movies as your friends like, which might not necessarily be true. It is also the case that the average number of friends on Facebook is around 200 people, which is more relations than it is realistic to manage on a day to day basis. This means that many of the friends that affect your recommendation of movies is someone you might have few or no things in common with. [18]

Merging a recommender system with a social platform could thereby potentially be optimized by narrowing down the circle of the recommender system to only the people who share the user's interests. This feature would also allow a user to meet people with similar interests and thereby tailoring a friends list for the user.

1.4.2 Sub-conclusion

Facebook has successfully created a social media platform, which as of 2018 has more than 2 billion active users every month, making it the most used social media platform. [19]

With the addition of the Facebook Watch feature, they are trying to further drive the evolution of social media platforms, making way for people to connect in ever more ways through these.
Facebook Watch has not been an immediate success, with the creators saying that producers are not taking full advantage of the possibilities that Facebook Watch offers [20]. Despite this, the service still believes to hold a lot of potential, and is still undergoing development, with the creators trying to find the exact model of features that is right.
Overall, the addition of the Facebook Watch to the already well-established social platform shows that the evolution of social media is on-going, and making a platform that contains both friendship and video recommendation, is relevant in modern technology.

1.5 Chapter conclusion

Until now our analysis has shown that people are in fact overloaded with information and choices in the digital world. This gives people a vast freedom of choice, but the additional load to the choice-making process adds a great amount of stress to people as well. At the same time, the amount of digital services, such as social media platforms, are still rising and play a still bigger part of our everyday life. An example of this is seen in the ongoing development of new features to already established services, such as the addition of Facebook Watch to the Facebook platform.

There is an ongoing shift towards recommender systems to become the main tool for navigating through the vast amount of information, where assistance in the choice-making process can be of great help. Furthermore, it is shown that user-based recommendation holds great profit potential for businesses dealing in these areas.

In addition to this, the section on stakeholder analysis concludes some of the potential requirements and wishes from key stakeholders that could arise, in the creation and implementation of a recommender system.

Overall this section of the report concludes that the information overload and choice-related stress addition from various digital services poses a relevant societal problem that a personalized recommendation system could in part help to reduce.

Chapter 2

Recommender systems

In the task of trying to optimize recommendations, the following chapter seeks to provide a basis for understanding the various different approaches to implementing recommender systems.

2.1 About recommender systems

A recommender system is a system used to recommend items in different areas, such as movies, music, restaurants, news articles, books and others [21]. Broadly speaking, the recommender system uses an algorithm that aims to create a list of items the user will be interested in. The list is created by trying to predict the likelihood that the user will like the item based on what items he previously liked and disliked. [22]

Several different techniques can be used to produce the list of recommended items. Among these techniques, collaborative filtering and content-based filtering are the two most commonly used. Sometimes, these methods are combined to form a so- called hybrid recommender systems [21]. These approaches will be discussed more in-depth at a later point.

2.2 Collaborative filtering

When a recommender system is using collaborative filtering to create predictions, the predictions could be created from data collected from the user's past activities, and from other users' activities and behaviours [22]. This means that the list is based on what a user has liked in the past, and on how similar the user is to other users.

There are two ways in which data collection of the users' preferences are collected; explicit and implicit data collection [22]. With explicit data collection, the user is directly targeted and asked to rate items in some way by either giving a rating to an item or choosing a preferred item from a list or creating a list of items he likes. With implicit data collection, the user's behaviour online is tracked and analyzed. This is done by observing which items a user views, and which of them the user actually watches. The other users can also be analyzed in order to find the similarities to other users, and then form a list of items, similar to theirs. [22]

As with any method, there are a few key issues with collaborative filtering. The ones worth mentioning are; the "cold start" problem, scalability and sparsity [21]. A cold start is when the system does not have enough data to get going. This is a problem because this method is dependent on already existing data, to be able to predict what users will like in the future. Scalability can become a problem when the method is used on a large environment. The more items available means that the amount of computing power needed increases drastically. The last issue worth mentioning is sparsity. Sparsity can be a problem with collaborative filtering, when the amount of items is large. That's because often, only a small amount of users are very active, and thus only the very popular items have a reasonable amount of ratings. [21]

2.3 Content-based filtering

Content-based filtering is different to collaborative filtering in the way, that here, the main focus of the recommender system is the properties of what the user himself liked and disliked previously, which is also represented by his user profile. Each item is described from a number of preset traits. These traits are then compared to the user's profile to determine whether or not, this is an item that fit the user, and if it is, then it will be recommended to the user. [21]

An issue with content-based filtering, is that the system often is not universal, in the way that it may only function correctly in the environment for which it is first designed for. This is the case because the system relies on the items, and therefore may not work on a different type of content [22]. It can also be a problem in content based filtering that large increases in the amount of data can cause scalability issues.

2.4 Hybrid recommender system

A hybrid recommender system, is a method which combines the key features of several methods. It might cover the issues from one method by implementing a feature from another method [21]. As an example, consider collaborative filtering. As mentioned earlier, collaborative filtering mainly suffers from an issue called a cold start. Collaborative filtering doesn't work very well at the start, that is, when a user first creates his/her profile. At this point the user hasn't given any ratings yet, therefore collaborative filtering takes a while to get to a point where it functions properly. This issue can be counteracted by weighing the prediction made by content-based filtering higher than the one made by collaborate filtering. As such, the recommender system only needs one item, to be able to recommend similar items.
This is the general idea with a hybrid recommender system, to improve already existing methods by combining them, so they can alleviate each others weaknesses.

An example of a hybrid recommender system, is Netflix's. Netflix uses a combination of collaborative filtering and content-based filtering [22]. They recommend movies by finding similarities between the user and other users, who have seen the same movie, as well as recommending movies based on the description of the movie that a user has just watched [23]. In addition to that, Netflix makes use of both explicit and implicit data collection. When a user creates a new profile, the user explicitly has to choose which kind of movies he/she likes. After that point, Netflix analyzes the users behaviour on the website i.e. implicit data collection. [23]

2.5 Chapter conclusion

As stated earlier in this chapter, there are different approaches to creating a recommender system - each with its own strengths and weaknesses which will be further examined in technical detail at a later point.

The following problem definition will conclude the problem analysis as it will outline the problem that has been reached and based on that present a problem statement that will be attempted solved throughout the report.

Problem Statement

Many major companies on the internet use recommender systems to simplify and personalize choices for their users. This is relevant in movies due to the growing number of choices, and need to tailor genres to a users interests. As previously mentioned Facebook recently launched the Facebook Watch service that centres around trying to use ones existing network of friends to recommend movies. This model could potentially be improved by focusing on shaping new friendships based on their common interests in movies, instead of getting recommendations from people one might have nothing in common with. Furthermore, when you have friendships based on similar interests, the quality of movie recommendation based upon these friendships, could improve as well.

Enabling a recommender system to both recommend movies accurately and shape friendships based on movie interests, one could use methods from both content-based filtering and collaborative filtering. The collaborative filtering method is easily used to find users with similar interests and its recommendations could be combined with a content-based method into a hybrid system to ensure the accuracy of the prediction.

These methods could be modeled by creating an overview of movies, user profiles, and making it possible to watch a movie online while interacting with other people with similar watching interests.
On the basis of the problem analysis and project specifications, the following problem statement is reached:

"How can a hybrid recommender system based on content-based and collaborative filtering be used to simplify personalized movie choices and allow people with similar interests to connect?"

2.6 Changes in problem statement

The problem statement has been narrowed down due to a significant decrease in the number of members at a late stage in the project. This caused the aforementioned problem statement to describe too large a problem. This means that the following problem statement will be the working focus of the report:

"How can a hybrid recommender system based on the combination of a content-based and collaborative filtering method be used to simplify making movie choices and increase their accuracy based on previous movie ratings?"

This effectively removes the part of the problem that seeks to allow people to connect and find other people with similar movie interests. The recommendations made will be visualized through a user interface which also enables the interconnectivity between the recommendations and the user input.

Chapter 3

Recommendation methods

In the task of designing a recommender system, several approaches can be used. Among these, the most pivotal approaches are Content-Based and Collaborate filtering as described in Chapter 2. The technical implementation methods of these approaches are thereby examined in the following chapter to get an idea of how the existing methods and theories work, and which ones might be favorable to implement in a platform of our needs.

3.1 Content-based filtering

From the onset, the project group's primary focus has been on machine-learning (ML) algorithms, as opposed to purely rule-based approaches. The reasons for this delimitation are fairly straightforward: Rule-based approaches comparable to state of the art methods tend to become too complex for humans to reliably understand and maintain. Additionally, rule-based recommender systems require preexisting knowledge about the data [21]. As the number of exceptions and special cases increases, such systems tend to become brittle, error-prone and unmanageable. While rule-based systems may be easier to initially implement in the form of e.g. if-else chains, a ML system will typically be a significantly more scalable method in the context of large and complex problems [21, p. 240]. However, a downside to certain ML approaches, such as neural networks, is the loss of transparency and explainability, since the inner working of such a model is effectively a "black box" where knowledge extraction is not possible [24, p. 11]. However, rule-based approaches can more easily incorporate domain-specific expert knowledge into its model, so it can be an advantage to hybridize the approaches. [21]

3.1.1 Artificial Neural Network

In this section we will give a basic introduction to Artificial Neural Networks —abbreviated as ANN. We will present a short description of its structure, and the basic functionality of its elements. This is done as ANN is one of the recommendation methods that can be used for data classification and approximation in regards to content-based filtering for a recommender-system - and in some cases also collaborate filtering.

To describe the basics of a ANN it is relevant to compare it with the structure and functions of neurons in the human brain, since these served as the inspiration for the first versions of ANN. As a result of this, an ANN is on a basic level modelled similarly to the brain's structure. [25]

This structure of neurons connected together is what came to be known as Artificial Neural Networks [26]. An example can be seen in figure 3.1.

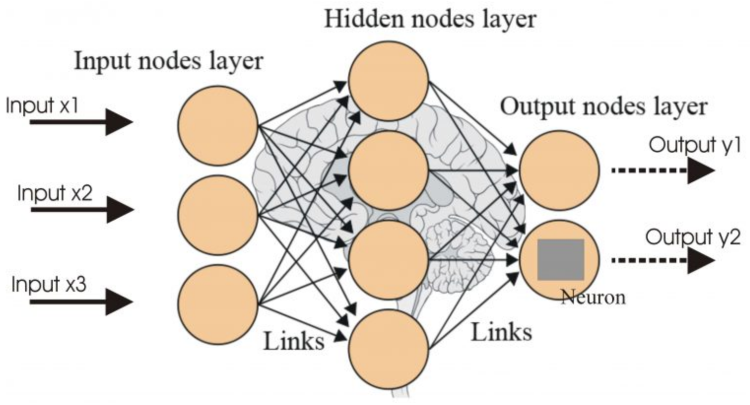


Figure 3.1: Fully-connected neural network - Basic structure of a neural network. [26]

The network is divided into layers where each layer in the network consists of an arbitrary number of nodes that can be described as the artificial equivalent of neurons [27]. Each of these nodes will then be connected to nodes in the preceding and following layer by using the links as is represented in Figure 3.1. This connection allows each node to get input from multiple [27] nodes and similarly send its output to multiple other neurons. Typically, these connections to other nodes are scaled by a corresponding weight that helps determine how much a specific input should effect a given node. [27]

Furthermore, each node consists of an activation function that decides what the output of the node should be based on the input [27]. This is again inspired by neurons in the brain, which only fire when a specific electrical threshold is met. This is determined based on the result of the total input that a neuron receives. [28]

For nodes this activation function can in its simplest form be binary, meaning that it will either be activated (1) or deactivated (0) [29]. Newer versions tend to use non-linear functions, such as the normalized Sigmoid function, which returns values in the range between 1 and 0, which makes way for ANN to be used on non-trivial problems. [29]

The specific structure of an ANN is highly customizable, and new layouts are developed regularly. One of the most simple structures is the "fully-connected networks" which is a way to say that all nodes in a layer is connected to all nodes in the preceding and the following layer - which is also represented in Figure 3.1.

After the structure and components of an ANN are set, it must be trained in relation to the desired purpose of the network and the data available, hereby "learning" from the data given, and carrying out its intended purpose. [27]

3.1.2 Training Methods

When we wish to train a neural network to make accurate prediction, we have multiple methods available to accomplish this. These methods each have their own pros and cons.

In this section we will cover some of the most popular ways of training a neural network, and end with a sub-conclusion on which method will be the best for our project.

Genetic algorithm

A genetic algorithm is based on natural selection and the principle of "survival of the fittest". [30]

The general method is quite simple if you have prior knowledge of the natural selection process, and how the "fittest" organisms survive and pass on their genes. A simple flowchart of the genetic algorithm can be seen at Figure 3.2
The first step is to initialize the population, which means that we for example generate 1000 random agents (Or animals), which currently have no knowledge of the problem they are supposed to solve. [32] The next step is "Evaluation". As the name indicates, this step is about evaluating every single agent we have generated.

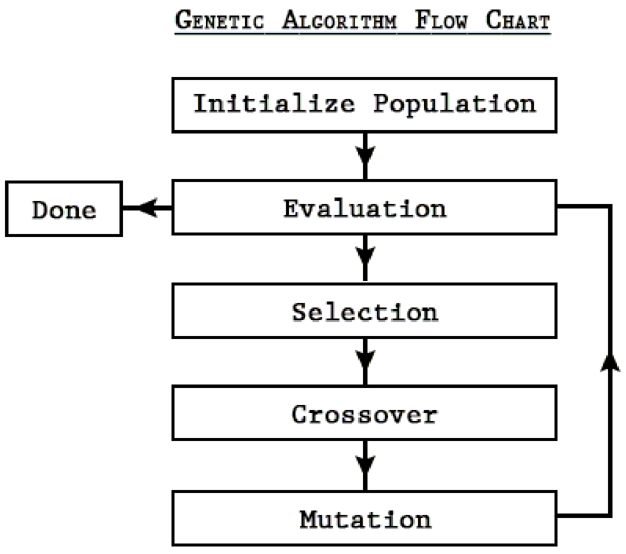


Figure 3.2: Flowchart of a genetic algorithm. [31]

This is done by placing every agent in a simulated environment, and then assigning the agent a score depending on how well it performed. [32]

The next step is *selection*, which is supposed to simulate "Survival of the fittest" by only letting a smaller portion survive the current round by choosing a percentage and letting the x% of best agents survive. The remaining agents are then eliminated. [32] At the *crossover* step we pair the surviving agents and let them produce an offspring. This can be done using multiple methods, but we will only cover the crossover method which also one of the most used. The remaining methods are outside the scope of this section. [32]
As Figure 3.3 shows, in the crossover step each of the parents are described as a binary string —also know as its genome. [34] One or more random crossover points are chosen, and then the offspring are generated by combining different sections of each of the parents [34]. The idea is that two passable solutions combined should have a better chance at producing a better solution by letting the helpful genes survive and eliminating the bad ones. [32]

But a problem one might quickly notice is that the gene pool is shrinking by every iteration of the process, and all agents quickly become almost equivalent in their features —i.e. they have been subjected to inbreeding [34]. To combat this aspect, another step is introduced. The so-called *mutation* step makes a few simple alterations to the offspring's genome by simply switching a few bits, as Figure 3.4

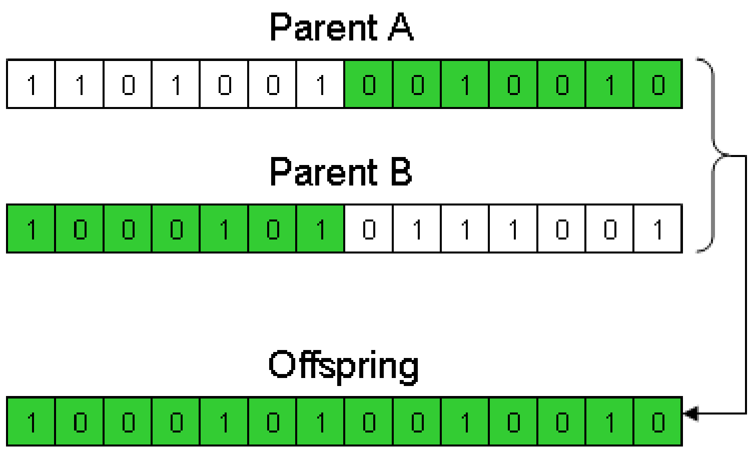


Figure 3.3: Genetic algorithm: Crossover example. [33]

shows. [34]

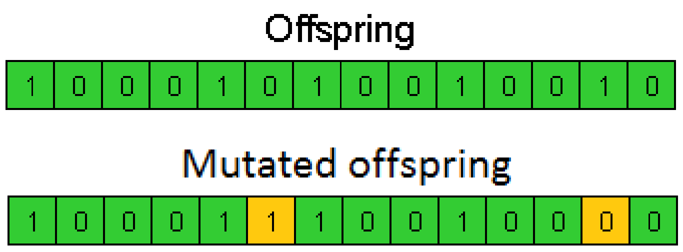


Figure 3.4: Genetic algorithm: Mutation example. (Edited version from [33])

This process of continuous evaluation, selection, crossover and mutation is then repeated through as many iterations as desired. However, at a certain point one should no longer expect development, and the algorithm can then be stopped and the best agent exported and saved for use in future implementations.[32]

In the context of neural networks when no labeled data is used, genetic algorithms represent an unsupervised learning method [35]. Therefore, genetic algorithms are most often used when there is either no data available, or the data requires too many resources to collect. A simulated environment also needs to be accessible, and a fitness function must be possible to create [36]. This method of training can be seen as an optimized brute force method, since the way of learning is based on random solutions, testing and then with an educated guess trying to generate a new semi-random solution and repeat the process. Therefore, this method can be quite time-consuming, since a lot of agents have to be generated and tested before an "optimal" solution can be chosen [37]. The method can however be useful when faced with a somewhat complex problem where gathering data is not doable, and learned methods can be ruled out. [36]

Backpropagation

Backpropagation is a supervised learning method, which can be used to train neural networks, by using gradient descent. [38]

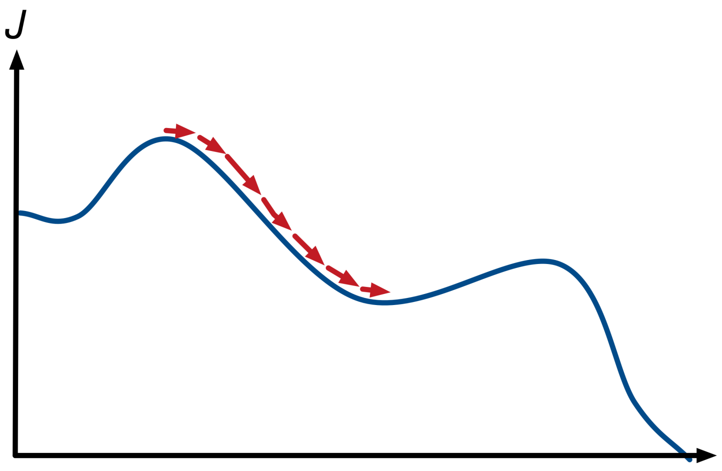


Figure 3.5: Gradient descent example. [39])

Gradient descent is graphically shown on Figure 3.5, and the method is quite simple to understand, even though the math can be somewhat confusing. The math is outside the scope of this section, and only the general method will be covered.

Gradient descent is the search or hunt for the most optimized combination of weights and bias values of the network, so the estimation of the neural network much better match the expected output. This allows it to better make predictions based on unknown data. [40]

You can think about gradient descent as a multidimensional graph where each parameter is a weight, and your goal is to optimize the loss of the weights. In other words, the lower you get on the graph, the better the solution. In the context of a neural network, we have some input data, and some expected output data, which is our goal, and by changing the values of the weights slowly in some direction, we should be able to move closer to the goal, and therefore increase the quality of our solution [41].
To find the direction in which you want to move is done by calculating the partial derivative of each weight, [41] starting from the last layer and moving backwards through the network. By calculating the partial derivative, we know if we have to

decrease or increase the weight to better match our output goal. [41]

One problem with back-propagation is that most training methods is non-convex, which means that it might have several solutions. This is illustrated in Figure 3.5, where the neural network gets "trapped" in a local minimum instead of finding the global minimum. Gradient descent only aims to go downhill, and if it first have to go uphill to go downhill, it will stay still as it can't see past the so-called "mountain top". [42] Luckily there are some methods to combat this problem, but most of these methods are outside the scope of this section. One of the most used and simple methods is "Mini-batch" or "Stochastic Gradient Descent", which instead of training over the whole dataset at once, trains over a few data points or a single data point. This helps one move through a non-convex problem, by making small continuous changes throughout the dataset. [42]

Backpropagation is a supervised machine-learning method, which means that it requires some sort of data to know what corresponding output to expect. [41] By using a big enough dataset, it should be capable of learning the underlying patterns. [40] Since backpropagation requires data, it is unusable if the problem is so abstract that no data can be extracted. Yet backpropagation is one of the most used training methods, and one of the fastest ways to train a neural network. [40]

Sub-conclusion

Having analyzed genetic algorithms (unsupervised) and backpropagation (supervised), the method of choice comes down to a single question:

Is there any data available and in sufficient quantities?

Since a lot of movie data sets containing user ratings are available online, the answer is *yes* and *backpropagation*. Therefore, backpropagation seems like the best choice for this project.

3.2 Collaborate filtering

3.2.1 Nearest Neighbor

K-nearest neighbors (k-NN) is a simple and intuitive scheme based on finding similar users or items in a vector space. [21]
The vector indices consist of the different features relevant to movies, such as "action" or "drama". Alternatively, the vectors can contain movie ratings, and the nearest neighbor approach can then be used to find user vectors that have an overlap in their movie ratings.

There are several different similarity measures at one's disposal. Some of the most common ways of deciding similarity is to either calculate the euclidean distance between two vectors [43], or find the angle between them using the Pearson correlation.

Pros and cons

One significant advantage of k-NN is that its inner workings are easy to implement and explain, and it is applicable to a broad range of problems [44].

On the other hand, k-NN risks scaling poorly, as it becomes very computationally expensive with an increase in the number of dimensions as it has a time-complexity of $\Theta(n^2)$ [45]. Another potential downside to the k-NN approach is having to explicitly specify the k number of neighbors to search for. In wide search spaces this often means traversing unnecessary branches, ultimately resulting in $k \cdot n$ searches.

3.2.2 Conclusion: Recommender Technology

The information gathered about different recommender technologies can be concluded in Table 3.1. The difficulty of implementation, time complexity and accuracy are often the most indicative features of whether a given method is right for the task at hand or not. The first column called ANN refers specifically to the "mini batch" training method which was chosen instead of the genetic algorithm (GA) due to the accuracy and the fact that large amounts of training-data is available which would not be used by a genetic algorithm due to the unsupervised nature of it.

The rule based system (RBS) is often not the suitable choice as it can often become increasingly difficult to implement as one tries to increase the accuracy of its predictions. The other features also follow this pattern which in many cases make it unreliable and difficult to make changes to. The k-NN algorithm - as opposed to the rule based system - is easily implemented and can be highly accurate which makes it one of the most used collaborate filtering methods in simple recommender systems.

	ANN	GA	k-NN	RBS
Implementation	Hard	Medium	Easy	Depends
Unsupervised		$\times$		$\times$
Supervised	$\times$		$\times$	
Scalability	Good	Good	Poor	Depends
Time complexity	$\Theta(n)$	$\Theta(n)$	$\Theta(n^2)$	Depends
Accuracy	High	Medium	High	Depends

Table 3.1: Table over features of different recommendation methods

Chapter 4

Architecture

This chapter seeks to outline the overall architecture of how the movie platform and recommender system will be made and work together. It will further examine the specifics of the recommender methods we chose to implement, the data that will be used to make recommendations and some preliminary sketches of the user interface.

4.1 Strategy

The problem that we wish to solve is formulated in the problem statement in the previous section. It is specified that a hybrid recommender system consisting of collaborative and content-based filtering should be used which respectively will be a kNN algorithm and an artificial neural network. These algorithms require data for training and validation purposes. The dataset chosen is 1 million data points of movie ratings from 6000 users on 4000 movies and will be provided by MovieLens [46].

4.1.1 Model-View-Controller

The recommendations will be made for users through a graphical user interface which they can interact with. The recommendation part and user interface will be connected by a controller to manage the requests from the users. Figure 4.1 outlines the architectural pattern of the proposed program. It is based on the model-view-controller diagram with the added section on how the model uses and changes the data about the user. The figure is based around the idea that each diagram describes the flow of information for one user, which thereby allows the user to get individual recommendations. The user looks at the user interface after which he uses it based on what he decides to do - and this information is given to the controller. The controller notifies the model based on what action the user makes. The user might have rated a new movie after which the DataLogger logs these changes in the proper data files. The GUI Logic might update the user interface to show the front page which requires recommendations which are then made by the recommendation part. To make recommendations the proper data files are used for both the collaborative and content-based filtering which both are taken into account through the hybrid system to make the recommendations. This concludes the use of the model after which the user interface should be updated and the user will once again be ready to interact with it.

4.1.2 Data Files

Each of the data files shown in Figure 4.1 is either derived from the MovieLens dataset or some other file that is derived from the dataset. The Ratings and SeenBy files are changed directly by the data logger. This is the case because the only immediate change that happen because of an action made by a user is that he rates a new movie which is then updated in these files.
The Ratings file keeps track of all movies that each user has rated and the respective ratings.
The SeenBy file keeps track of which users have rated each movie to ensure that these are not recommended again.
The MovieProfiles and UserProfiles are both data files keeping track of how much each genome score is present in a movie or liked by a user.
The Correlations file keeps track of the similarities between all users.

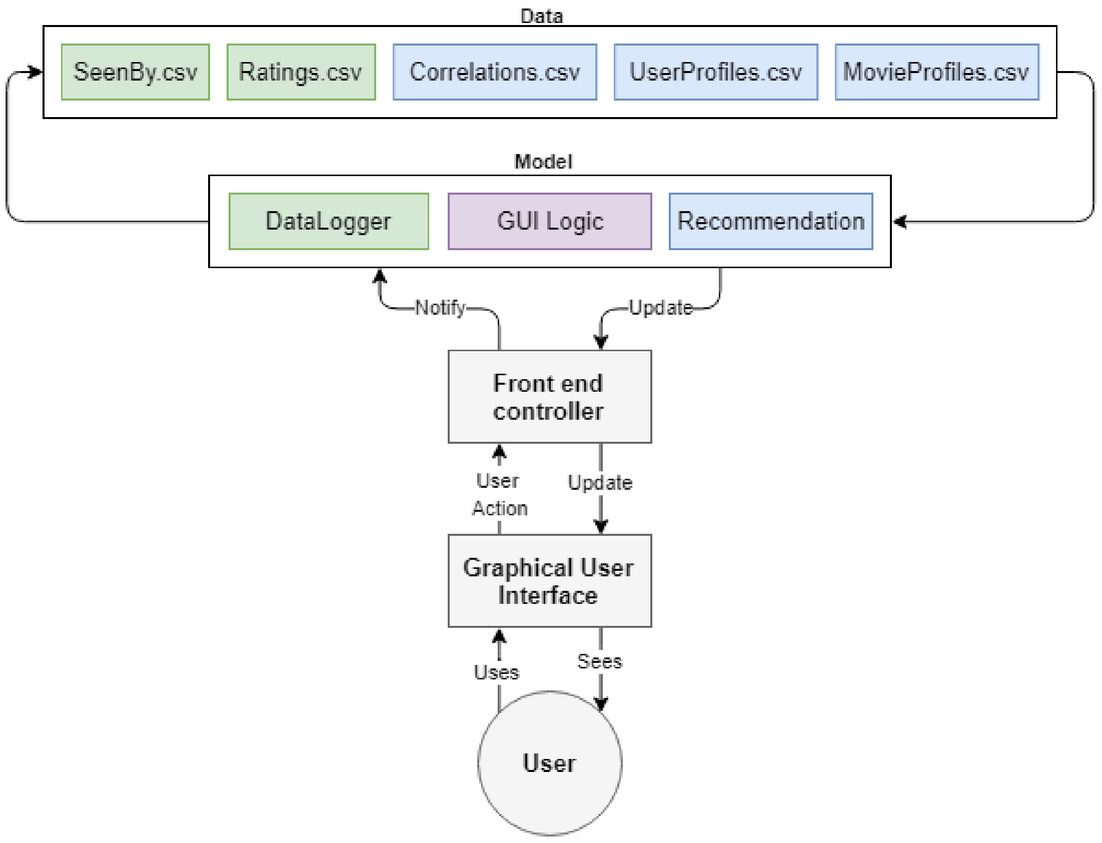


Figure 4.1: Model-View-Controller structure of proposed solution

4.2 Dataset

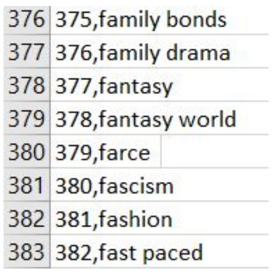
It has been decided that the pre-processed dataset called *MovieLens* will be used. This dataset has been collected and processed with the help of machine-learning techniques by a group known as *GroupLens* - at a computer science and engineering research lab at the University of Minnesota [47]. In the following section, the reasoning for this data delimitation choice will be discussed.

The alternative to using pre-processed data would be to generate fictitious and arbitrary data from scratch. As the initial objective of this project report is to develop a program with real-world application and relevance, using fictitious data is entirely undesirable. Additionally, the task of gathering enough valid and usable real-world user data is a time-consuming task that lies beyond the scope of this project.

Notable features of the MovieLens dataset

The *MovieLens* dataset is packed into comma-separated-value(csv) files. Next, the most noteworthy files from the dataset will be briefly discussed.

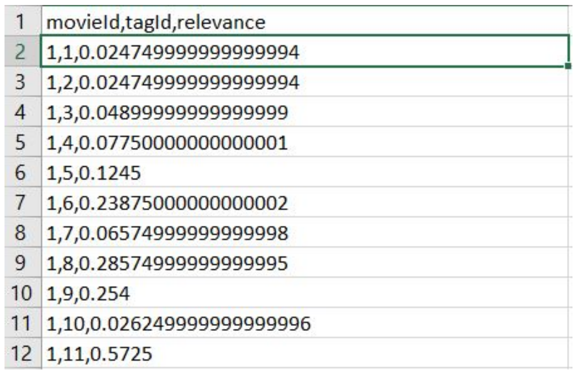
The file genome-tags.csv contains the 1128 different genome tags. Each tag is a word describing some feature of the movie such as "Darkness" or "Blood". These genome tags are derived from a combination of user-submitted reviews (including tags, ratings and text), as well as the result of a machine-learning algorithm applied to the aforementioned user reviews [48].



376	375,family bonds
377	376,family drama
378	377,fantasy
379	378,fantasy world
380	379,farce
381	380,fascism
382	381,fashion
383	382,fast paced

Figure 4.2: Sample of MovieLens genome tags

The file genome-scores.csv contains a mapping of movie IDs to genome tags. The strength of this association is given by a relevancy score as shown below. The non-binary score ranging from 0 to 1 helps give a more nuanced evaluation of the movies, and allows users to make more accurate decisions regarding their future choice of movies [48].



1	movieId,tagId,relevance
2	1,1,0.024749999999999994
3	1,2,0.024749999999999994
4	1,3,0.048999999999999999
5	1,4,0.077500000000000001
6	1,5,0.1245
7	1,6,0.23875000000000002
8	1,7,0.065749999999999998
9	1,8,0.28574999999999995
10	1,9,0.254
11	1,10,0.026249999999999996
12	1,11,0.5725

Figure 4.3: Sample of MovieLens genome scores for movie IDs

4.3 UI-design

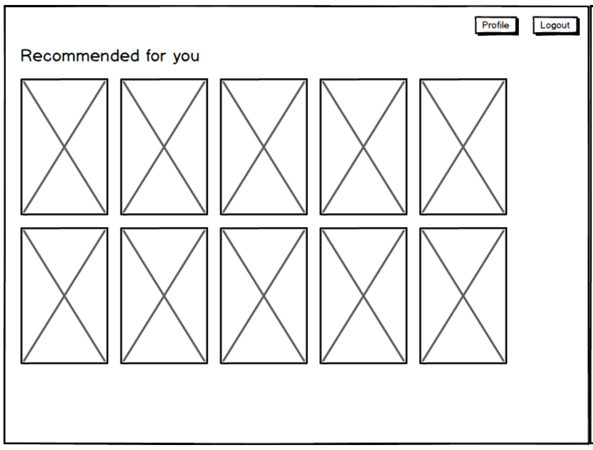
This section seeks to outline the general ideas, regarding the design of the platform's graphical user interface.

Due to time constraints, the user interface is restricted to only have three main pages: A front page, a log in page and a "create profile" page. Of these, the most significant will be the front page, where the user is presented with the results of the recommender system. An example of what the front page will look like is shown in figure 4.4a. The front page mainly consists of 10 tiled images next to each other in two horizontal rows. Each image will show the name of a movie that is recommended to the user. In addition to the recommended movies, the front page will have a bar of buttons at the top of the page, which will allow the user to access the other pages of the platform. As figure 4.4a shows, the bar will have two buttons: A profile button and a log out button.

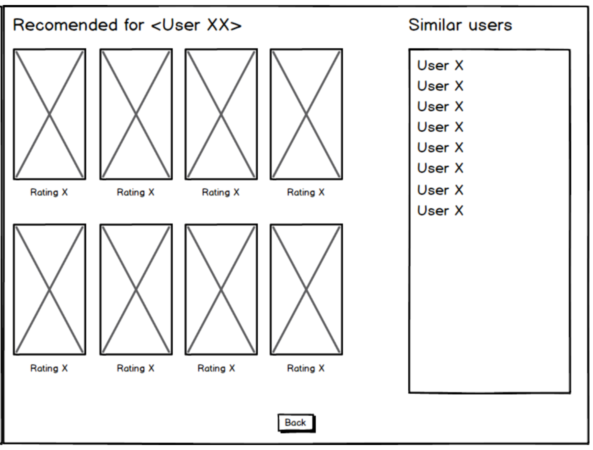
When the profile button is clicked, the user is redirected to the profile page, shown in figure 4.4b. Here the user will be presented to movies he has rated highly in the past. On the right side of the page, is a list of other users that like the same kind of movies. When one of the users are clicked, the tiled images to the left will change, to show the history of rated movies for the new user. When the back button is clicked, the user is redirected back to the front page.

When the log out button on the front page is clicked, the user will be redirected to the log in page, visualized in figure 4.5a. The log in page will feature very few elements: Two text fields, where the user can input his User ID and password, and two buttons; a "log in" button and a "create profile" button. The idea with the User ID text field is that it will be possible to enter a user ID from the dataset used by the algorithm. This will mean that the list of recommended movies on the front page will be tailored to that specific user ID. The password field will not be functional, as there will be no validation of the password entered, it only exists to show a realistic view of a typical log in page. When either of the buttons are clicked the user will be directed to another page of the platform. The log in button will display the front page as mentioned above, and a click on the create profile button, will redirect the user to the create profile page.

The create profile page is shown in figure 4.5b. Due to the time limit, the contents of this page will be mostly visual, as we do not intend to make is possible to create a new profile. The page exists as an example to show how such a page would look, in a version of the platform where this functionality is implemented. The page will have a text field, allowing the user to select a username. Then the user will be asked to rate a number of movies on a scale from one to five. This is to prevent the system from suffering from a cold start. The create profile button at the bottom of the page would then create the user's profile and display the front page, but for this project, it will remain nonfunctional. The cancel button redirects the user back to the log in page.

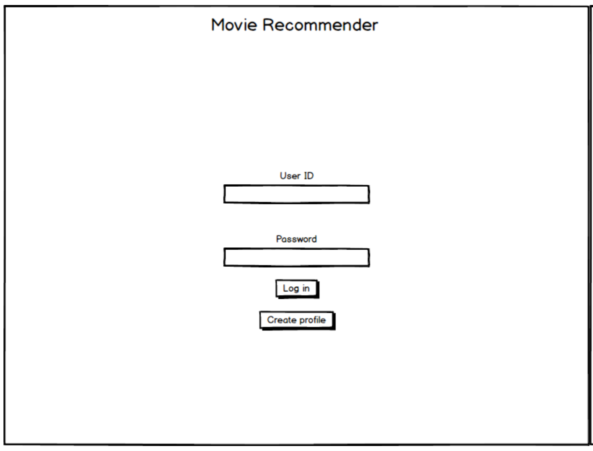


(a) Front page

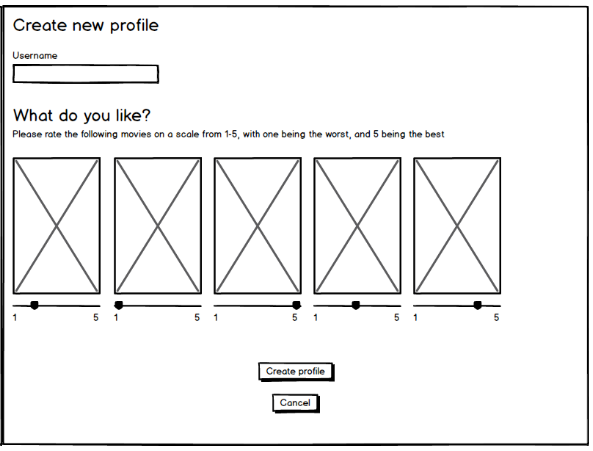


(b) Profile page

Figure 4.4: Examples of how the front page, and the profile page would look like



(a) Log in page


Create new profile
Username
What do you like?
Please rate the following movies on a scale from 1-5, with one being the worst, and 5 being the best
1 5
1 5
1 5
1 5
1 5
Create profile
Cancel

(b) Create profile page

Figure 4.5: Examples of how the log in page, and the create profile page would look like

4.4 Content-based filtering - Artificial Neural Networks

In Section 3.2.1 we introduced the basic principle to understand artificial neural networks. This section will describe the feed forward backpropagation process in more detail, and add the mathematical definitions and equations required to perform these tasks.

4.4.1 Section notation

The mathematical notation for defining which node or weight of the neural network one might be talking about can often get quite confusing, so we'll begin this section describing the notation used. Figure 4.6 illustrates the terms describing the neural network.

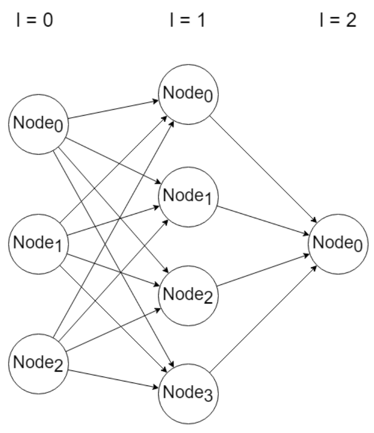


Figure 4.6: Example network to support the section notation.

Nodes

Every node in layer $[l]$ is described as a vector with the notation:

$$N^{[l]} \tag{4.1}$$

An example could be the node vector of layer 0:

$$N^{[0]} = \begin{bmatrix} Node_0 \\ Node_1 \\ Node_2 \end{bmatrix} \tag{4.2}$$

$N_0^{[0]}$ then describes:

$$N_0^{[0]} = Node_0 \tag{4.3}$$

Weights

Every weight connecting into a layer $[l]$ (Left to right) is described as a n x m matrix where $n = |N^{[l-1]}|$ $m = |N^{[l]}|$:

$$W^{[l]} = \begin{bmatrix} w_{00} & w_{01} & \dots & w_{0m} \\ w_{10} & w_{11} & \dots & w_{1m} \\ \vdots & \vdots & \dots & \vdots \\ w_{n0} & w_{n1} & \dots & w_{nm} \end{bmatrix} \tag{4.4}$$

Where each column is equal to the weight vector for each node in layer $[l]$.

$$W^{[l]} = \begin{bmatrix} \vdots & \vdots & \dots & \vdots \\ W_0^{[l]} & W_1^{[l]} & \dots & W_m^{[l]} \\ \vdots & \vdots & \dots & \vdots \end{bmatrix} \tag{4.5}$$

An example could be the weights connecting into node 0 in layer 1:

$$W_0^{[1]} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} \tag{4.6}$$

Activation function

The activation function of a layer is defined as:

$$g^{[l]}(z) \tag{4.7}$$

4.4.2 Feed forward

Feed forward is the process where inputs are passed though the network from the input layer through the network to the output layer. The input is information about the user and the movie which outputs a movie score prediction based on the users preference towards this certain kind movie.

The feed forward process is done by updating each layer consequently, starting from layer 1 (The first hidden layer) all the way until you reach the output layer, where a movie rating is returned. To update a layer one must update each node in the layer, which is done by taking the sum of all nodes connecting into the node multiplied by their respective weight, and then applying the activation function.

Example, update the first node in layer l.

$$N_0^{[l]} = g^{[l]}(N^{[l-1]} * W_0^{[l]}) \tag{4.8}$$

Here we take the sum of the output values of the nodes in layer $l-1$ multiplied with their respective weights going into node $N_0^{[l]}$ and then applying the activation function $g^l(z)$ for layer l, to update the node.

We can extend this equation to calculate the output of the whole layer at once, by multiplying with the weight matrix of the layer instead of the weight vector of each specific node:

$$N^{[l]} = g^{[l]}(N^{[l-1]} * W^{[l]}) \tag{4.9}$$

4.4.3 Backpropagation

Back in section 3.1.2 we covered a few different ways to train a neural network, to perform better at predicting movie ratings, and we made the sub-conclusion that backpropagation would be the appropriate method for our project. This section will cover the method in further detail than previously, and takes a look at the mathematical equations required to perform this training method.

The Chain Rule

The backbone of backpropagation is the chain rule, and when performing back-propagation, the tactic is to keep using the chain rule until completely done going backwards through the network.

The chain rule can be seen below at Equation 4.10, which states that the derivative to a function, with respect to the change of an inside function is equal to a multiple of each individual functions derivatives.
As shown below at the right side of the *or*, the derivative to the function f with respect to x is equal to the derivative to f with respect to the change of g times the derivative of the function g with respect to the change of x.

This can also be described as dependencies. If f depends on g which depends on x, then f also depends on x - hence *the chain rule*.

$$\frac{d}{dx}f(g(x)) = f'(g(x)) * g'(x) \quad or \quad \frac{df}{dx} = \frac{df}{dg} * \frac{dg}{dx} \tag{4.10}$$

The cost function

When using feed forward to evaluate some user and movie data, into some predicted movie score output, there will most likely be some margin of error. To calculate this error a cost function is applied to the output prediction. The most popular cost function is the modified MSE (Mean Squared error), which is simply the MSE function multiplied by a half, which is done to make future steps easier. Multiplying by a constant is no problem, since we later multiply by other constants too.

The modified MSE cost function is defined as:

$$Cost(Target, Prediction) = \sum_{i=1}^{n} \frac{1}{2} * (Target_i - Prediction_i)^2 \tag{4.11}$$

Where Target is the expected output vector and Prediction is the predicted output vector, where n is equal to the size of the output layer.

Calculating the partial derivative

To update a weight we need to know in which direction to alter the weight, to be going downhill. This is done by calculating the partial derivative of the cost function with respect to a specific weight.

To calculate the partial derivative to all weights going into any layer l in the network, the following equation is used:

$$\frac{dCost}{dW^{[l]}} = N^{[l-1]} * E^{[l]} \tag{4.12}$$

The size of the $\frac{dCost}{dW^{[l]}}$ matrix will then be equal to the weight matrix $W^{[l]}$ for the same layer.

The definition of $E^{[l]}$ haven't yet been covered. Its definition is the "effect vector" for layer l, and it defines the "effect" that the input of all nodes in the layer has on the overall error. It's mathematical definition is:

$$E^{[l]} = \frac{dCost}{dN^{[l]}} = g'^{[l+1]}(N^{[l+1]}) \odot (E^{[l+1]} * W^{[l+1]}) \tag{4.13}$$

Where $\odot$ denotes component-wise multiplication.
The definition of the effect vector given in 4.13 is only true for any layer which is not the final output layer, since the effect of this layer is calculated based on the cost function.

The cost function defined in 4.11 tells us that the overall prediction rating error is equal to the sum of errors over each node. The Effect of any output node is then equal to it's individual error, multiplied by the activation prime function of the output layer.

$$E^{[l^*]} = g'^{[l^*]}(Prediciton) \odot (Prediction - Target) \tag{4.14}$$

Where l^* denotes the final output layer.
Prediction is equal to the output prediction vector of the final layer $N^{[l^*]}$, and target the expected output vector.
The reason behind this is that if we change the output of one output node, the other nodes will remain constant, and therefore have a derivative equal to 0. Since the derivative of a sum, is equal to the sum of derivatives, the effect of any output node will be equal to it's own error.

4.5 Collaborate filtering - k-Nearest Neighbors

4.5.1 Euclidean distance

As previously described in section 3.2.1, the main property of the k-NN algorithm is that it can map features of a user and find the similarity between these users. One way to map this is to make a vector that describes how well one user likes different genres or a vector consisting of all the ratings a user has made. Two users could thereby be described as: $\bar{a} = \{a_1 ... a_n\}$ and $\bar{b} = \{b_1 ... b_n\}$.

The main problem with creating these user vectors based solely on a user's ratings is that users often have only rated a small part of the possible base of movies. This results in the user vectors being sparse and thereby not necessarily being indicative of what a user likes. Just because a user has not rated a movie it doesn't mean that he or she doesn't like it.

Another way of creating these user vectors is to make use of the fact that each movie has a number of genome tag score between 0 and 1. By utilizing these to generate user profiles as described in section 4.1.2, these can make up the user vectors and thereby represent ones taste in movies.
The distance between two user vectors can be calculated with the following formula [43]:

$$d(\bar{a}, \bar{b}) = ||\bar{a} - \bar{b}|| = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 ... (a_n - b_n)^2} \tag{4.15}$$

which outputs the euclidean distance between the two vectors and could also be described as the difference between the two users a and b.

4.5.2 Similarity score

Intuitively it is more convenient to describe the distance between users as a similarity between them instead of their difference. To find a similarity score ranging from 0 to 1 with 0 being two opposite users and 1 being identical ones, the distance is inserted into the following formula:

$$\frac{1}{1 + d(\bar{x}, \bar{y})} \tag{4.16}$$

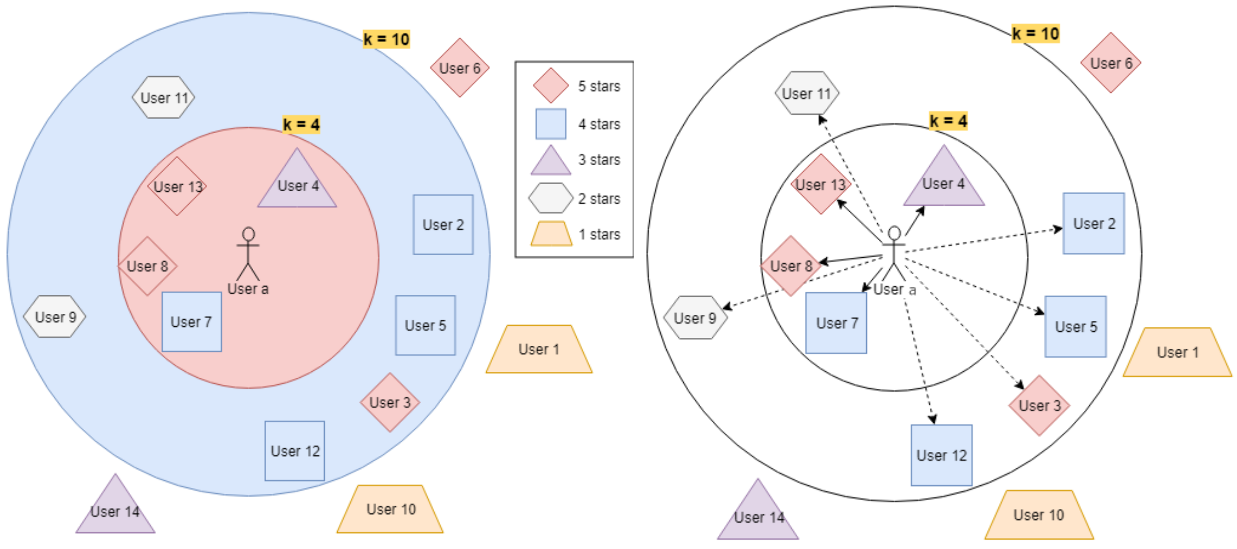
The logic behind it is to find the fraction of the distance and thereby get a value between 0 and 1. 1 is added to ensure that when the distance is 0 the formula returns 1. When these similarity scores have been calculated from one user a to all other users b, these can be sorted and the k most similar users be chosen. When evaluating the k nearest neighbors one needs to decide whether the output rating should be a classification or a regression.

4.5.3 Classification

The basic idea of k-NN classification is that it divides the neighbors into classes based on what rating they gave the movie that user a needs a prediction on. The output rating is decided by the class with the most members within the k-nearest neighbors. This is illustrated in Figure 4.7a which shows that looking at the 4 nearest neighbors - 2 of those neighbors rated 5 stars whereas 1 rated 4 stars and 1 rated 3 stars which results in an output of 5 stars. If we look at the 10 nearest neighbors on the other hand we now see that there are 4 people that rated 4 stars and only 3 people that rated 5 stars which results in and output of 4. This shows that it is important to choose an indicative k-value to ensure that it is not just the class with the most members overall to decide the rating which means that k is often small and sometimes just 1.

4.5.4 Regression

When looking at regression on the other hand, the output rating is not just decided by the one class with most neighbors close to user a, but instead all k-nearest neighbors have a say. This is illustrated in Figure 4.7b. One way of making a predicting to user a for a movie m based on all k-nearest neighbors could be to find the average of their ratings as in Formula 6.3 where a and b are users, $r_{b,m}$ is a movie rating made by user b on movie m and N_m is the set of all the k-nearest neighbors having rated the movie m.



(a) Classification **(b)** Regression

Figure 4.7: Showing the similarities and differences between k-NN classification and regression

$$pred(a,m) = \frac{\sum_{b \in N_m} r_{b,m}}{|N_m|} \tag{4.17}$$

This approach can be improved by using the neighbors' similarity scores to user a as weights. The similarity scores to user a would serve as weights of how much their ratings count towards the predicted rating. This is expressed in Formula 4.18.

$$pred(a,m) = \sum_{b \in N_m} w(a,b) \cdot r_{b,m} \tag{4.18}$$

The weight importance of a neighbor is calculated by normalizing their similarity as it is shown in formula 4.19.

$$w(a,b) = \frac{sim(a,b)}{\sum_{b \in N_m} sim(a,b)} \tag{4.19}$$

By using this method the predictions made are based on a normalized weighted similarity which allows for a more indicative result than the average of all ratings.

4.5.5 Rating bias

The previously presented collaborative methods are all used under the presumption that people's patterns are predictable which is often not the case. This causes a bias of some sort that is not taken into account. One of the main biases is difference in rating behaviours which is illustrated in Figure 4.8

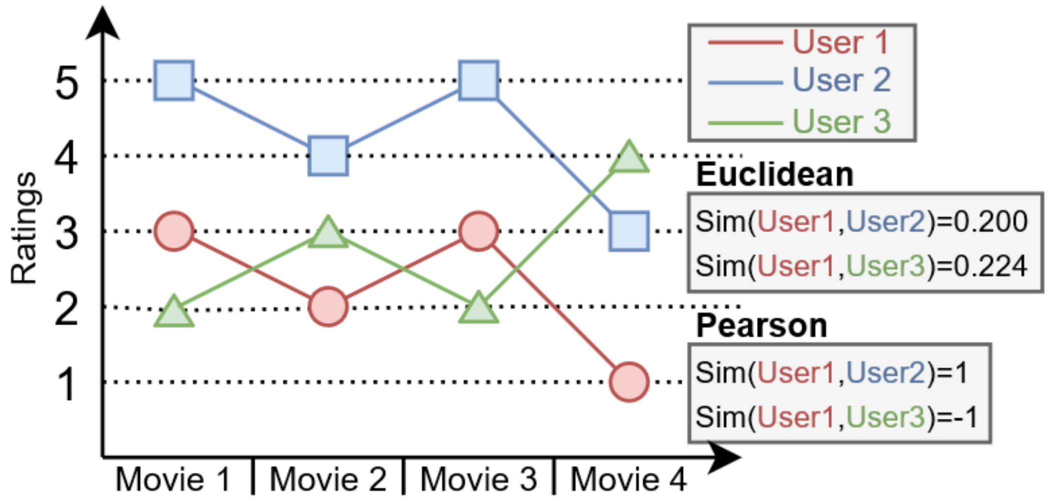


Figure 4.8: Example of a rating bias

As is illustrated the rating patterns of user 1 and 2 are identical but either user 2 has a tendency to rate movies highly or user 1 may have a tendency to use lower ratings. In the previously presented methods user 1 and 2 would achieve a similarity score lower than user 1 and 3 because the euclidean distance between users 1 and 3 is shorter but the behaviour patterns are not taken into account. If they were taken into account users 1 and 2 would be more similar than user 1 and 3 which will be examined in the following section on a new method of calculating similarities.

4.5.6 Pearson correlation coefficient

The Pearson correlation coefficient is a measure of the linear correlation between two variables. It returns a similarity-score between -1 and 1 respectably describing two completely opposite variables and two identical variables. Comparing it to previously presented methods this does not find the distance between two user vectors but rather the difference in angle between the two user vectors. This allows one to take the rating bias into account when calculating similarities. The formula for calculating the Pearson correlation between users a and b is given in Formula 4.20:
$M = M_a \cap M_b$, which is the set of all movies both user a and b have rated.
$r_{a,m}$ is a rating made by user a on movie m.
$\bar{r}_a$ is user a's average rating of movies.

$$sim(a,b) = \frac{\sum_{m \in M}(r_{a,m} - \bar{r}_a)(r_{b,m} - \bar{r}_b)}{\sqrt{\sum_{m \in M}(r_{a,m} - \bar{r}_a)^2}\sqrt{\sum_{m \in M}(r_{b,m} - \bar{r}_b)^2}} \tag{4.20}$$

Looking at Figure 4.8 again and recalculating the similarities with the Pearson correlation the similarity score between users 1 and 2 is 1 and users 1 and 3 is -1. By using this method users 1 and 2 are now more similar than users 1 and 3 because this method allows for rating biases in the data.

In addition to the fact that the Pearson correlation allows for rating biases it is also fairly easily implemented as is shown in Algorithm 1. It uses lists of users having rated each movie as data, as well as lists of ratings made by user *a* and *b*. The algorithm returns the Pearson correlation between user *a* and *b*.

Algorithm 1 k-NN Pearson Correlation - User based collaborate filtering

```
Data:   Ratings made by User $a$, ratingsVector_a
        Ratings made by User $b$, ratingsVector$_b$
        Lists of users having rated each movie, movies
Result: Pearson correlation coefficient between user $a$ and $b$

1: MeanCenteredProduct          // Product of mean-centered ratings
2: LengthUser1                  // Length of mean-centered ratingsVector_a
3: LengthUser2                  // Length of mean-centered ratingsVector_b

4: foreach movie $\in$ movies do
5:   if both users have rated the movie then
6:     $MeanCenteredProduct \mathrel{+}= (r_{a,movie} - \bar{r}_a)(r_{b,movie} - \bar{r}_b)$
7:     $LengthUser\_a \mathrel{+}= (r_{a,movie} - \bar{r}_a)^2$
8:     $LengthUser\_a \mathrel{+}= (r_{b,movie} - \bar{r}_b)^2$
9:   end
10: end

11: $LengthUser\_a := \sqrt{LengthUser\_a}$
12: $LengthUser\_b := \sqrt{LengthUser\_b}$
13: return $\frac{MeanCenteredProduct}{LengthUser\_a * LengthUser\_b}$
```

4.5.7 Summary

When comparing Euclidean distance and Pearson correlation it seems the latter is favorable as it takes rating bias into account and looks for rating patterns instead of just distance. The comparison between making predictions with classification or regression is often case-dependant but as regression can take far more neighbors into account and weigh them it seems like the most favorable method. In conclusion the Pearson correlation coefficient will prospectively be used as the collaborative method and to make predictions, regression will be used.

Chapter 5

User Interface

This chapter seeks to describe the process of connecting the graphical user interface to the algorithms, thus building the movie recommender platform. The chapter will feature the thought process of the implementation as well as the methods used to connect the two parts of the platform.

5.1 Choice of Framework

The programming language of choice for this project is Java, which also applies when programming the user interface. When designing a graphical user interface in Java, it is helpful to use a framework. Luckily Java provides a few to choose from, the ones being most used are JavaFX and Swing. We have chosen to use JavaFX, as Swing is no longer being updated by Oracle [49], as well as JavaFX being a part of the curriculum for this semester's programming course.

To build the user interface, the JavaFX scene-builder will be used. A scene-builder is a plug-in which is linked to the project's source files. The scene-builder makes it possible to build the user interface through drag and drop programming. The scene-builder will then generate the code for the visual part of the user interface, and thus writing the .fxml file for the project. The .fxml file contains all the elements of the interface, that are visible to the user, as well as the properties of those elements. Properties meaning the visual features and positioning of the elements. When the user interacts with the interface by pressing a button, an event will occur. These events are programmed in a different file, called controller. The controller handles the back-end part of the interface. The flow goes like this: The user interacts with the interface, this triggers an event, which is handled in the controller. The controller handles the event, and changes the user interface accordingly.

5.2 Data imports

To allow the user interface to show which movies are recommended, and to who, it is necessary to import some data, from the previously described comma separated data files. Below each method used to import data to the user interface, will be described. All the methods are static methods, located in the DataImporter class. This class holds various methods, used to import data from .csv files.

5.2.1 ImportTitles

The method *importTitles()* utilizes a CSV-reader to read the contents of the file *movies.csv*. This file contains two one column. The file contains one row for each movie ID, and two columns. The first column holds a movie ID, and the second column holds the movie title corresponding to that movie ID. The method uses the CSV-reader to read each line in *movies.csv*, and transfers the contents to a HashMap. The HashMap holds movie ID's as keys, and movie titles as values.

5.2.2 ImportRatings

The method *importRatings*(int userID) also utilizes the CSV-reader. This method reads the file *<userID>.csv*, the user ID being the method's input parameter. This means, that there is one file for each user in the dataset. Each .csv file contains only one row. The files contains one column for each movie ID. Each column holds a movie ID and that movie's predicted rating for the given user. The method uses the CSV-reader to read the file, and transfers the contents to a LinkedHashMap. The LinkedHashMap holds movie ID's as keys and predicted ratings as values. The map is then sorted by values, to make the movies with the highest predicted ratings appear first in the map.

5.2.3 ImportSeenBy

The method *importSeenBy* reads the file *seenBy.csv*. This file contains all the movies the user has watched, for each user as well as the rating he has given it afterwards. The file has one row for each user, where the first column is a user ID. The subsequent columns holds a movie ID, and the rating the user has given to that movie. The method uses the CSV-reader to read the file, and transfer the contents to a HashMap, the keys being user ID's. The values of the HashMap, is another HashMap, containing movie ID's and the ratings made by the user.

5.2.4 ImportCorrelations

The method *importCorrelations* read the file *pearsonCorrelations_100.csv*. This file contains information about all user, and their correlations to the 100 most similar users. The file has one row for each user, where the first column is that user's user ID, and the subsequent columns containing the user ID of a similar user and the correlation between the two. The correlation being a number between 0 and 1. The CSV-reader transfers the content of the file to a HashMap, holding the user ID's as keys, and the values being another HashMap, holding the user ID's of the similar users as keys, and their correlation to the main user as values

5.3 Displaying the user interface

Figure 5.1 shows when each method is called, and shows the structure of the code, used by the user interface. Next, the methods used to display the contents of the interface, will be explained.

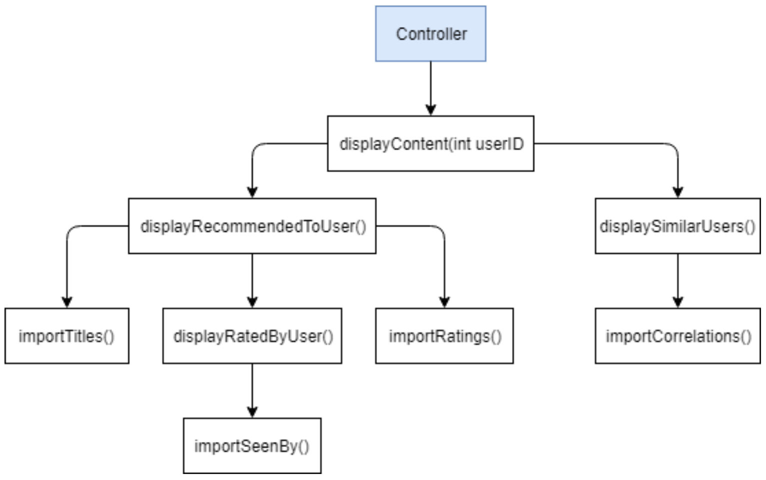


Figure 5.1: Program structure of classes and methods in the GUI code

The first method to be called, is the method *displayContent*(int userID). This method is called when the log in button on the front page is clicked. It reads the content of the text field on the log in page, and uses this integer as an input parameter. The method calls the methods *displayRecommendedToUser*(int userID) and *displaySimilarUsers*(int userID), which will be explained further.

5.3.1 Displaying the titles of recommended movies

As mentioned in section 4.3, the front page will feature a list of the ten movies, with the highest predicted rating for a given user. To achieve this, the method *displayRecommendedToUser()*(int userID) is written. To find and display the highest predicted movies, two imports are necessary. Firstly - a list of all movie ID's and their titles are imported from the data file *movies.csv*, by calling the static method *importTitles()*. This returns a HashMap of movies with movie ID's as keys and their titles as values. This HashMap will be labelled *titleMap*. The second import is a list of all movie ID's and the their predicted rating for a given user. The method *importRatings*(int user ID) is called with the user ID from the text field on the log in page as the input parameter. This returns a sorted LinkedHashMap with movie ID's as keys and their corresponding predicted ratings as values.This map is labelled *predictedRatings* Now it is possible to iterate through the first ten entries of the map to get the movie ID's of the ten highest predicted movies for that user. These are stored in an ArrayList of integers, which are used to get the titles of the movies from the *titleMap*. These titles are then displayed through the labels on the front page.

As mentioned earlier, interface should not display a movie the user has already rated, on the list of recommendations. To make sure this does not happen, the method *displayRatedByUser*(int userID) will be called. This method displays the titles the user has already seen, and returns a list of movie ID's corresponding to the movies, the user has already rated. Now when the map of predicted ratings is iterated through, a check is made to ensure that none of the recommended movies is present in the list of already watched movies.

5.3.2 Displaying titles on the profile page

As mentioned in section 4.3, a profile page is accessible from the front page. This page displays the top ten rated movies a user has watched, and the respective ratings he/she has given them. To display these, the method *displayRatedByUser()*(int userID) is written. Again, two imports are required, the first being the *importTitles()*, which returns a map of movies. The second import is a list of all movies, the user has already seen. The method *importSeenBy()* is called. This returns a HashMap with user ID's as keys and a HashMap containing the movies the user has watched and the rating he has given them, as values. The user ID given in the log in page, is used to get the map of his history of watched movies, and the ratings given. This is converted to a LinkedHashMap, which is sorted by its values, to make the movies the user has rated highest, appear first. Now it is possible to iterate through the map, and display the titles of the movies, and the rating given, and display the titles and ratings through the labels on the profile page.

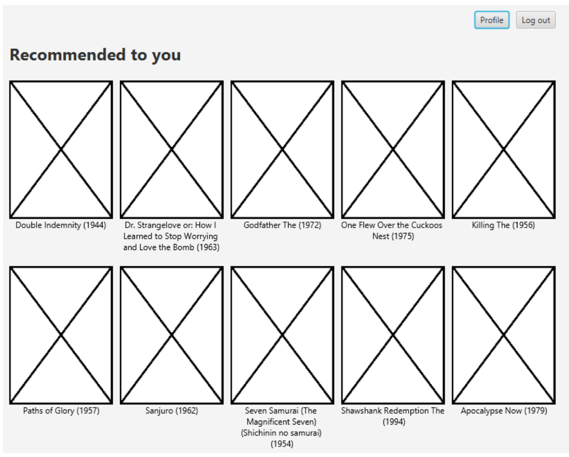
5.3.3 Displaying similar users on the profile page

The profile page serves two purposes. To show which movies a user has watched already, and the rating the user has given them. The second purpose is to show the collaborate part of the recommender system, by providing a list of the 10 most similar users. To create such a list, the method *displaySimilarUsers*(int userID) is written. To display a list of users with the best correlations, it is necessary to import data, by calling the method *importCorrelations*, which returns a HashMap of all users and their correlations with 100 of their most similar users. The inner HashMap is retrieved, converted to a LinkedHashMap, and sorted by values. Now we have a map of the users with the best correlation to the main user, we have logged in with. This map is iterated through, and the 10 most similar users are displayed on the profile page, as well as their correlations to the main user.

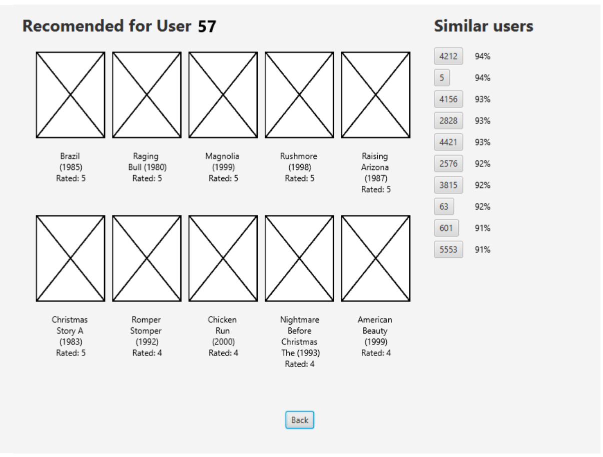
5.3.4 Demonstration

This section will demonstrate how to move around in the user interface, to show how the interface looks like in it's final form.

When the program is started, the user is presented to the log in page. As an example, we input the user ID 57, to arrive at the front page, as shown in figure 5.2a. Here we are presented to the 10 movies with the highest predicted rating, for user 57. In the top right of the page, we can click the profile button, and be redirected to the profile page, shown in figure 5.2b. On this page the can see the 10 movies user 57 has watched and rated highest. On the right side of the page is a list of the 10 users, most similar to user 57. Each similar user is represented with a button and a number, corresponding to that user's correlation to user 57. Each button can be clicked, and the method *displayContent* is called with that user's user ID.


Profile
Log out
Recommended to you
Double Indemnity (1944)
Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1963)
Godfather The (1972)
One Flew Over the Cuckoos Nest (1975)
Killing The (1956)
Paths of Glory (1957)
Sanjuro (1962)
Seven Samurai (The Magnificent Seven) (Shichinin no samurai) (1954)
Shawshank Redemption The (1994)
Apocalypse Now (1979)

(a) Front page


Recomended for User 57
Brazil (1985) Rated: 5
Raging Bull (1980) Rated: 5
Magnolia (1999) Rated: 5
Rushmore (1998) Rated: 5
Raising Arizona (1987) Rated: 5
Christmas Story A (1983) Rated: 5
Romper Stomper (1992) Rated: 4
Chicken Run (2000) Rated: 4
Nightmare Before Christmas The (1993) Rated: 4
American Beauty (1999) Rated: 4
Back
Similar users

4212	94%
5	94%
4156	93%
2828	93%
4421	93%
2576	92%
3815	92%
63	92%
601	91%
5553	91%

(b) Profile page

Figure 5.2: Screenshots of the front page, and the profile page

Chapter 6

Implementation of Algorithms

6.1 Artificial Neural Network

The math mentioned earlier for neural networks, might suggest the design of our neural network would be programmed as a series of matrix multiplications, but this is not true. We chose to design our neural network in an objected oriented manner, where a neural network is designed as a series of layer, node and weight objects.

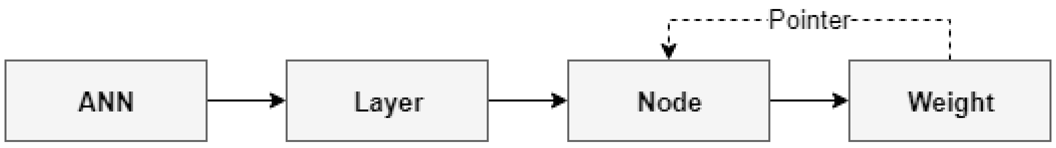


Figure 6.1: Simple representation of the ANN architecture.

The neural network is designed as a single object called "ANN" which contains a bias node (We will cover this in a later section) and a list of layer objects. The layer object contains a list of node objects, which in return contains a list of weights connecting into that specific node.

A more in-depth diagram can be seen at Figure 6.2, which illustrates a neural network with 3 inputs, 2 hidden nodes and 1 output. The architecture is simple, the ANN object contains a list of layer objects, which contains a list of nodes, which contains a list of weights. A weight contains two values, a weight value, and a pointer to it's starting node, which is a node in the previous layer. This is also illustrated in Figure 6.3

So whenever we desire to update a layer by feed forward, the process is simple,

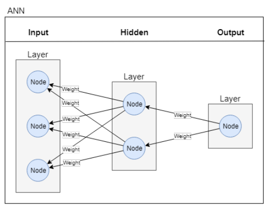


Figure 6.2: A more in depth representation of the ANN architecture.

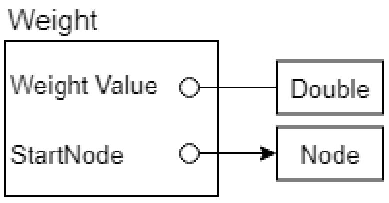


Figure 6.3: A representation of the Weight object.

we go though every node in a layer and accumulate the value of the starting node multiplied by the weight value for each weight in the node, and then apply the activation function, to get the output of that node. This means that the calculations are moving forward in the network, but the weights are connected backwards.

6.1.1 Feed forward implementation

As mentioned in 4.4.2, to perform feed forward, one must update each layer and to do this, one must update each node in every layer, starting from the first non input layer and all the way towards the final output layer.

To update a node, its value has to be changed to be equal to the sum of all its inputs multiplied by their respective weights, and then applying the layer activation function.

Our implementation of feed forward is of course very similar, but since our network is designed as a structure of objects, we have outlined the problem a little different.

In the object *Layer* we have the method *updateLayer()* which does all the feed forward calculations on that layer. This is done by calling a method named *updateNode()* on every node in the layer, which in turn sums the product of each weight and the respective input nodes, that connects into it, and then applies the activation function.
Calling this *updateLayer()* method on all layers in the network will result in a full

feed forward through the network, and the output results will then be the values of the final output layer.

To update our network / evaluate some inputs, the following pseudo-code is used:

Algorithm 2 ANN.evaluateInputs

Data: Input data, inputVector
Result: Prediction vector

```
Network                                   // The ANN instance

$Network.InputLayer := inputVector$
foreach Layer ∈ Network.Layers do
    Layer.updateLayer()
end

return ANN.OutputLayer
```

Algorithm 3 Layer.updateLayer

Result: Calls updateNode() on every node in the layer

```
Layer                                     // The Layer instance

$Network.InputLayer := inputVector$
foreach Node ∈ Layer.Nodes do
    Node.updateNode()
end
```

6.1.2 Backpropagation

Our backpropagation algorithm is alike our feed-forward algorithm, and not based on matrix multiplication, but instead a more iterative method.

To perform backpropagation we use an object named *Trainer* which is tasked to perform any training activity on a given network.
A diagram of the *Trainer* object and its dependant objects can be seen at Figure 6.4, the dotted lines represent that an object uses another object, while the fully-drawn

Algorithm 4 Node.updateNode

Result: Updates the node, by performing feed forward.

```
1: Node                                        // The Node instance
2: Sum                       // Value used to sum the input of a node.

3: $Network.InputLayer := inputVector$
4: $Sum := 0$
5: foreach Weight $\in$ Node.Weights do
6:     Sum += Weight.value * Weight.startNode.Value
7: end
8: Node.value = Sigmoid(Sum);
```

lines represent implements/extends.

The *Trainer* uses a number of different objects to perform it's task. The *Trainer* object uses a *CostFunction*, *TrainingMethod*, *ANN*, *L_Regulization*, learning rate and a dropout rate.
Describing every object the trainer uses is outside the scope of this section. We will only briefly cover the most crucial objects: *CostFunction* & *TrainingMethod*.

The objects required by the trainer is often from an interface or abstract class, which makes the implementation extremely modular and changeable.

CostFunction

The *CostFunction* is an interface that implements two methods: *Cost(Target, Prediction)* and *CostPrime(Target, Prediction)* where *Cost()* calculates the error (cost of a prediction), by applying any costFunction method. The *CostPrime()* method calculates the partial derivative to the error (cost of a prediction), with respect to the prediction of that method.

TrainingMethod

The *TraingMethod* is an abstract class which takes care of the actual training w - which consists of weight calculations. The abstract class implements some general methods as *updateLearnedWeights()* which takes the calculated temporary new weights, and saves them as the real weights - See 6.1.2.

Beyond implementing some default methods, the abstract class also implements

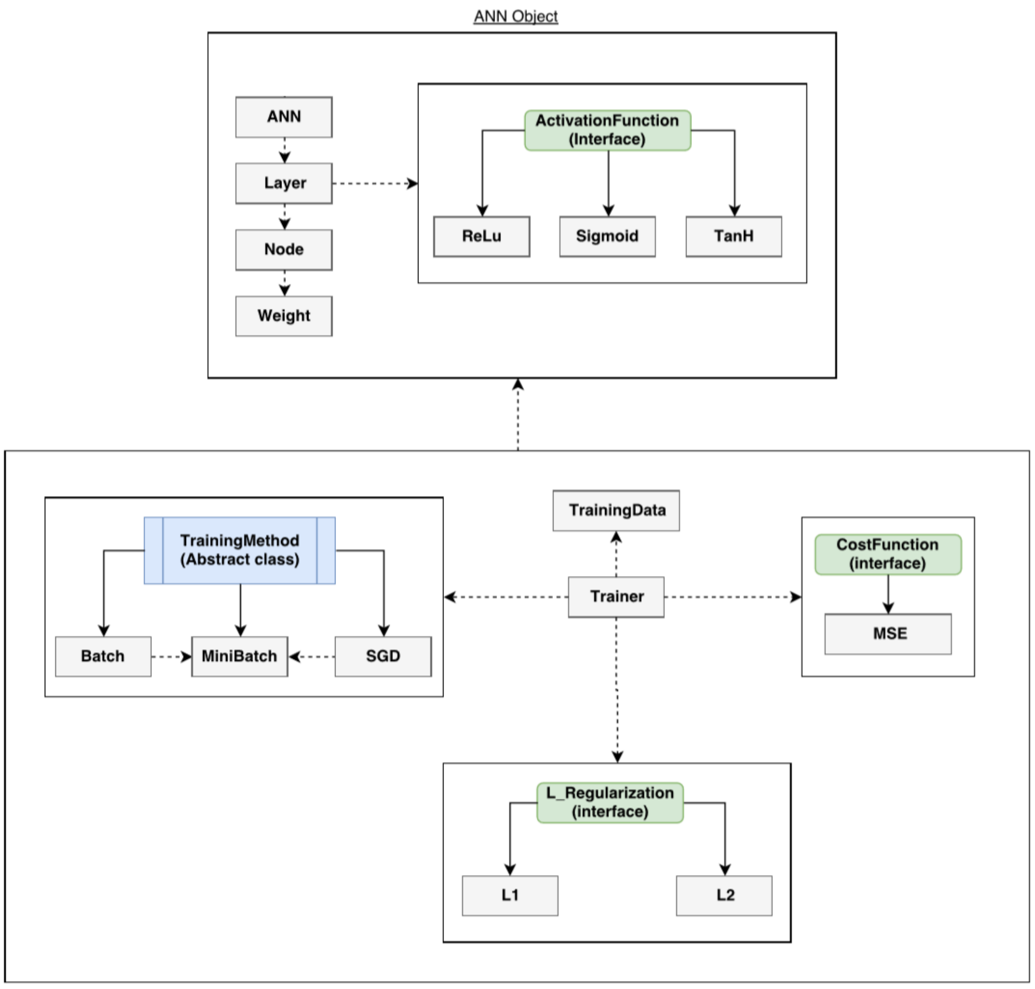


Figure 6.4: Implementation of the Trainer + ANN object.

the method *Train()*, which is the method that gets called whenever a training iteration is needed.

Implementation

As is described in Algorithm 5 the backpropagation algorithm begins with calculating the effect of every node in a layer, then calculates the derivative of every weight connecting into that layer, and save the updated weight value in a temporary value. Move one layer backwards, and perform the same calculations until every weight has been calculated.
When every layer has been iterated over, the calculated temporary weight values are moved into the original weight values.

Algorithm 5 Backpropagation

Data: List of input vectors, inputVectorList
List of output vectors, outputVectorList
A LearningRate, learningRate
Result: Performs one iteration of backpropagation

```
1: Network                                              // The ANN instance

2: $Network.InputLayer := inputVector$
3: foreach Layer ∈ Network.Layers do
4:    foreach Node ∈ Layer.nodes do
5:       Calculate effect of Node
6:
7:       foreach Weight ∈ Node.Weights do
8:          Calculate new weight and Save in temp values.
9:       end
10:   end
11: end
12: Move calculated weight values from temp into the real network
```

6.1.3 Input/Output design

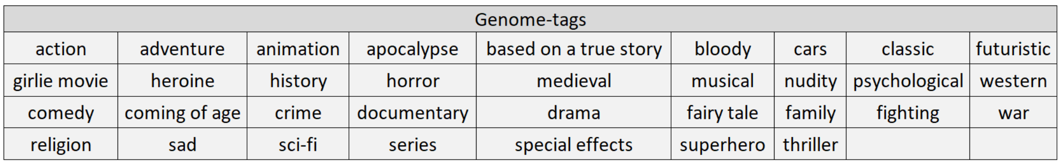
Once the feed forward and backpropagation algorithm had been designed and implemented, the input/output design pattern of our neural network, also had to be designed.
In the end we chose that the neural network needed two inputs: A user vector and a movie vector, also known as a user profile and movie profile. Only one output was needed, a single rating value between 0 and 1, which maps into the scores from 0 to 5.

The user and movie profile structure are very alike, and each contains 52 values. The first 34 values each describe a different genome-tag value, and the last 18 values define genres.

Genome Tags

As described in 4.2 the MovieLens dataset has a long list of 1128 individual genome-tags. Since using all 1128 genome-tags would immensely increase the size of our neural network, beyond the group's current capabilities in both technical knowledge and hardware access. The list therefore had to be drastically shortened.

The genome-tags were filtered using a **manual subjective method**, where the group went though all 1124 genome-tags and removed all genome-tags assessed to be non-relevant or low in quality, which resulted in a total of 34 different genome-tags. The final genome-tags can be seen below at Figure 6.5

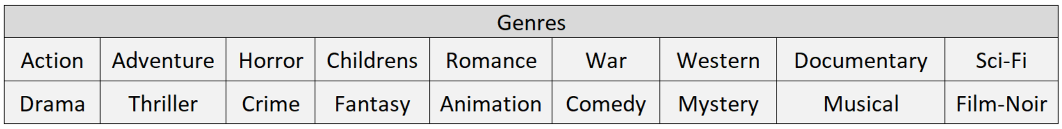


Genome-tags								
action	adventure	animation	apocalypse	based on a true story	bloody	cars	classic	futuristic
girlie movie	heroine	history	horror	medieval	musical	nudity	psychological	western
comedy	coming of age	crime	documentary	drama	fairy tale	family	fighting	war
religion	sad	sci-fi	series	special effects	superhero	thriller		

Figure 6.5: The remaining 34 genome-tags.

Genres

The dataset included a total of 18 different genres, which can be seen below on Figure 6.6.



Genres								
Action	Adventure	Horror	Childrens	Romance	War	Western	Documentary	Sci-Fi
Drama	Thriller	Crime	Fantasy	Animation	Comedy	Mystery	Musical	Film-Noir

Figure 6.6: List of all 18 genres.

Movie profiles

A movie profile is more or less static, and only changes if the genome scores or genre of the movie is changed, which probably won't happen very often.

The generation of a movie profile is very straight forward. First we extract the genome-tag scores from the remaining 34 genome-tags, and save them inside the profile.
The 18 genres is of Boolean type, which means that it is 0 (false) if the genre does not apply to the movie and 1 (true) if it does. As mentioned these values are the last 18 values in a movies profile.

The movie profile will now contain two list, one with the genome scores and one with the genres.

User profile

The user profile is less static, and updated every time the user sees a movie, and makes a rating.

Every value in the user profile is a value between -1 and 1. These values are generated and updated depending on which movies the user sees, and which rating they give the movie.

In short, the values change depending on the movie profile. If a user sees a movie and gives it a high rating (5), the user profile will make small adjustments to better match the movie profile, so the "similarity" between the user profile and the movie profile will become bigger. And opposite if the user assign a very bad score (1), the user profile will make a small adjustment to move away from the movie profile, and therefore decrease the similarity. If a neutral score (3) is given, the user profile wont change.

These adjustments are made using the following formula:

$$Value_{New} = Value_{Old} + AdaptRate * (MovieValue - Value_{Old}) \tag{6.1}$$

Where *AdaptRate* is calculated as:

$$AdaptRate = 0.15 * map(score,\ 1,\ 5,\ -1,\ 1); \tag{6.2}$$

Where 0.15 is subjectively chosen value that moves the user profile 15% closer to the movie profile if a score of 1 is given. *map* is a method that seeks to convert an input-value in the range between -1 and 1, to a value in the range between 1 and 5, to make it match the range of ratings:

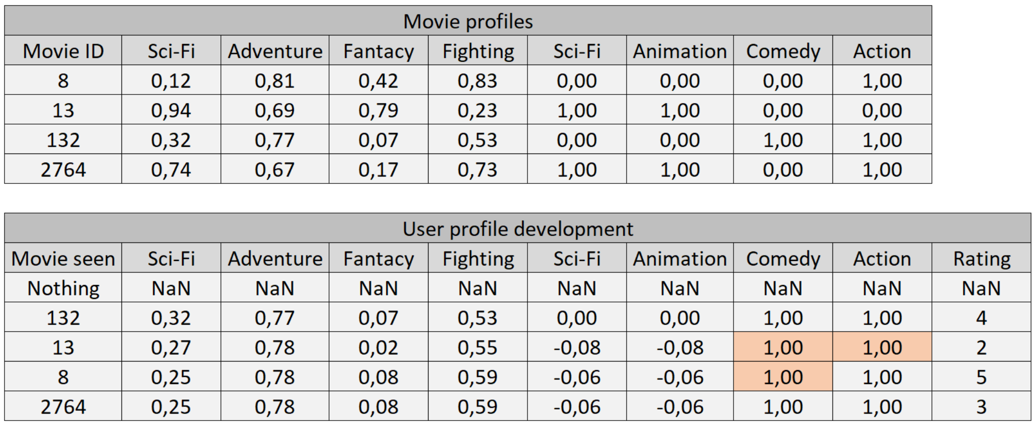
$$map(value,\quad in_{min},\quad in_{max},\quad out_{min},\quad out_{max}); \tag{6.3}$$

Which maps a value between in_{min} and in_{max} to a value between out_{min} and out_{max}.

This calculation is then done to each of the 52 different values describing both genome-tag and genres, and thereby updating the user profile for every movie seen and rated.

An example of these calculations can be seen at Figure 6.7. As seen on the figure, when a user haven't seen or rated any movies, then user profile does not contain any real values, but when the first movie (132) is seen, the user profile becomes identical to that movie.

Then as the user rates movies, their user profile will adapt, to better match their preferences, by either moving towards a movie profile, or away from the movie profile, depending on which rating was given.
A rating above 3 will change the user profile towards the movie profile, and a rating less than 3 will change the user profile away from the movie profile. While a rating of 3 will make the user profile remain constant.



Movie profiles								
Movie ID	Sci-Fi	Adventure	Fantacy	Fighting	Sci-Fi	Animation	Comedy	Action
8	0,12	0,81	0,42	0,83	0,00	0,00	0,00	1,00
13	0,94	0,69	0,79	0,23	1,00	1,00	0,00	0,00
132	0,32	0,77	0,07	0,53	0,00	0,00	1,00	1,00
2764	0,74	0,67	0,17	0,73	1,00	1,00	0,00	1,00

User profile development									
Movie seen	Sci-Fi	Adventure	Fantacy	Fighting	Sci-Fi	Animation	Comedy	Action	Rating
Nothing	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
132	0,32	0,77	0,07	0,53	0,00	0,00	1,00	1,00	4
13	0,27	0,78	0,02	0,55	-0,08	-0,08	1,00	1,00	2
8	0,25	0,78	0,08	0,59	-0,06	-0,06	1,00	1,00	5
2764	0,25	0,78	0,08	0,59	-0,06	-0,06	1,00	1,00	3

Figure 6.7: User profile development example.

The cells marked as orange, are values which have been calculated to a value outside the range of -1 to 1, and therefore have been limited.

6.2 k-Nearest Neighbors

The implementation of the k-NN algorithm is based on the pseudo-code in algorithm 1 which only describes a simplified version. The implemented code needs methods of importing and storing data as well as optimizations. These are all topics that will be described in further detail in this section.

6.2.1 Code structure

The overall structure of the k-NN code is visualized in Figure 6.8. The top class of the hierarchy is an abstract class called HandleNeighbors which contains methods to sort neighbors by similarity and select the k most similar ones. This abstract class is implemented by both the EuclideanDistance and PearsonCorrelation classes. The EuclideanDistance class and its methods will not be described further because of the choice of implementing the Pearson correlation.

The PearsonCorrelation class is responsible for importing data on users and calculating correlations based on this after which they are exported. In the constructor for the PearsonCorrelation class the static method importSeenBy() is called to read the required data into memory. The only publicly available method inside the class is exportCorrelations() which is responsible for both calculating correlations and printing them to a file as is also clear in Figure 6.8.

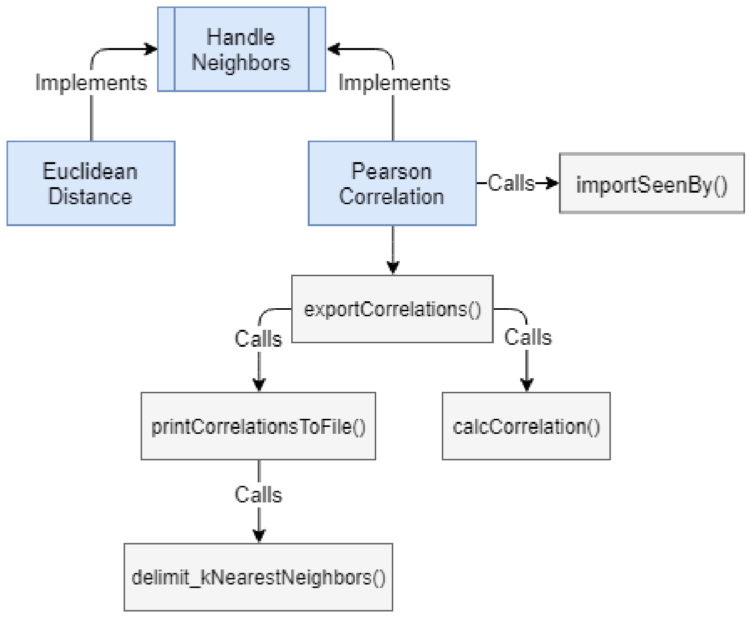


Figure 6.8: Program structure of classes and methods in the k-NN code

6.2.2 Generating data

As illustrated in Figure 6.8 the seenBy.csv is needed in the Pearson Correlation class but is not originally part of the data set. The data set consists of a file called ratings.csv which in turn consists of all the necessary data needed in seenBy.csv. It contains pieces of data such as time stamps which is not necessary and is formatted in such a way that would worsen the accessibility of the relevant pieces of information to kNN. It is therefore reformatted and inserted to the seenBy file.

6.2.3 Receiving data

The k-NN algorithm depends on getting data about all users' ratings, and using this to calculate correlations between all users. The calculated correlations can be used to predict ratings for users which will be described further in the following section. The data needed to calculate correlations is in the generated file: seenBy.csv. Each line represents a user with a user-ID and all movies rated by this

user as well as his ratings on the respective movies. This is stored in a HashMap of HashMaps. The logic behind this is that HashMaps store a value for each key and since the dataset skips some user IDs it will stay consistent because of the use of keys as opposed to indices in ArrayLists. The reasoning for storing HashMaps inside of HashMaps is that each user should store a HashMap of all movies and ratings he has seen. This brings it on the following form:
$seenBy = HashMap < userID, HashMap < movieID, rating >>$

6.2.4 Calculating correlations

To find the k most relevant neighbors it is necessary to first find correlations to all users and then sort out the k most relevant ones. This means that one must iterate from all users to all users and find their correlations. The correlations are also stored in a HashMap inside a HashMap on the following form:
$correlations = HashMap < fromUserID, HashMap < toUserID, correlation >>$

The java code in Listing 6.1 is part of the PearsonCorrelation class and the method used for exporting the correlations between users. The previously mentioned seenBy.csv consisting of all users is used to iterate from every user to every user. This is shown in lines 6 and 9 where seenBy is iterated over. At each iteration of the inner for-loop the method for calculating correlations between two users is called with the fromUser ID and the toUser ID.

```
1  public class PearsonCorrelation {
2      public void exportCorrelations() {
3          HashMap<Integer, HashMap<Integer, Double>> correlations = new HashMap<>();
4          Double correlation;
5
6          for (Map.Entry<Integer, HashMap<Integer, Double>> fromUser : this.seenBy.entrySet()) {
7              // Creating a new HashMap to store all correlations in
8              correlations.put(fromUser.getKey(), new HashMap<>());
9
10             for (Map.Entry<Integer, HashMap<Integer, Double>> toUser : this.seenBy.entrySet()) {
11                 correlation = calcCorrelation(fromUser.getKey(), toUser.getKey());
12                 correlations.get(fromUser.getKey()).put(toUser.getKey(), correlation);
13             }
14         }
15
16         printCorrelationsToFile(correlationAllUsers);
17     }
18 }
```

Listing 6.1: exportCorrelations method as code

As the calcCorrelation method is called we wish to take a further look at its code which is shown in Listing 6.2. A few pieces of information is needed before being able to calculate the correlation. Equation 4.20 shows that the average of all ratings made by each user is required and is calculated accordingly in lines 5 and 6. Another piece of information needed is all movies rated by both user A and B, which is found by making a set of watched movies for each user and finding the intersection of them between lines 11 and 18. The last thing to check before calculating the correlation is that users A and B have rated more than 7 of the same movies. This is in place to ensure that users who have only rated one common movie won't get a correlation of 1 solely based on that one movie.

On line 27 an iterator starts going through all the movies that users A and B have in common. In each iteration the three variables *meanCenteredProduct*, *lengthUser_A* and *lengthUser_B* are accumulated based on each user's rating on the current movie based on the implementation in Algorithm 1. Upon having gone through each movie the square root of the variables *lengthUser_A* and *lengthUser_B* are found to finally get the length of each user vector. Before returning the resulting correlation a check is made to ensure that the numerator - consisting of the product of the two lengths as in Equation 4.20 - does not equal 0 which would throw an error. Lastly the calculated correlation between users A and B is returned.

```
1  public class PearsonCorrelation {
2      private double calcCorrelation(int userID_A, int userID_B) {
3          double meanCenteredProduct = 0, lengthUser_A = 0, lengthUser_B = 0, movieRatingA,
               movieRatingB;
4
5          double avgRatingA = averageOverallRating(userID_A);
6          double avgRatingB = averageOverallRating(userID_B);
7
8          HashMap<Integer, Double> userA = this.seenBy.get(userID_A);
9          HashMap<Integer, Double> userB = this.seenBy.get(userID_B);
10
11         // Make set of all movies user A has seen
12         Set<Integer> userMoviesA = new HashSet<>(this.seenBy.get(userID_A).keySet());
13
14         // Make set of all movies user b has seen
15         Set<Integer> userMoviesB = new HashSet<>(this.seenBy.get(userID_B).keySet());
16
17         // Find intersection between the two sets to find the movies they have in common
18         userMoviesB.retainAll(userMoviesA);
19
20         // Return correlation 0 if user A and B have not seen more than 7 of the same movies
21         if (userMoviesB.size() < 7) {
22             return 0;
23         }
24
25         // Loop through all movies user A and B have in common
26         // Calculate their correlation
27         for (Integer movieEntry : userMoviesB) {
28             movieRatingA = userA.get(movieEntry);
29             movieRatingB = userB.get(movieEntry);
30
31             meanCenteredProduct += (movieRatingA - avgRatingA) * (movieRatingB - avgRatingB);
32             lengthUser_A += (movieRatingA - avgRatingA) * (movieRatingA - avgRatingA);
33             lengthUser_B += (movieRatingB - avgRatingB) * (movieRatingB - avgRatingB);
```

```
            }

            lengthUser_A = Math.sqrt(lengthUser_A);
            lengthUser_B = Math.sqrt(lengthUser_B);

            if (lengthUser_A * lengthUser_B == 0)
                return 0;

            return meanCenteredProduct / (lengthUser_A * lengthUser_B);
        }
    }
```

Listing 6.2: calcCorrelation method as code

6.2.5 Exporting correlations

The code in Listing 6.1 calculates correlations from all users to all users, but this results in every correlation being calculated twice. The enhanced for-loops might look from User 1 to User 2, but the next iteration of the outer for-loop it looks from User 2 to User 1 which will result in the same correlation being both calculated and stored twice. This can be solved by adding a conditional statement to only calculate and store correlations if the fromUser ID is smaller than the toUser ID as is shown in Listing 6.3.

```
if (fromUser.getKey() >= toUser.getKey())
    continue;
else {
    correlation = calcCorrelation(fromUser.getKey(), toUser.getKey());
    correlations.get(fromUser.getKey()).put(toUser.getKey(), correlation);
}
```

Listing 6.3: Optimization of number of correlation calculations

This optimization only skips the duplicate correlation calculations which is not enough. All correlations are still required to be exported as they are used for predicting ratings and we want to be able to make predictions for both User 1 and User 2. This gets handled when printing the correlations to a file which is done in the last line of code in Listing 6.1, where the printCorrelationsToFile method is called. A similar iterator is constructed that goes from all users to all users and in the inner for-loop a conditional statement checks whether this was one of the correlations that were skipped, and if so, then go get the equivalent one in correlationAllUsers. From each user a LinkedHashMap is made to store his correlations after which the k most similar users are found and lastly these are printed to a file.

6.3 Predictions

The use of neural networks and k-NN algorithms is all made to ensure the best possible predictions for each user. This is implemented in the code described in this section where the methods for predicting ratings are described, and how the link between two different prediction algorithms can make a combined prediction.

6.3.1 Structure

The structure of the prediction code was written to ensure the ability to make predictions using all three methods to allow for continuous validation of each model, even though the one that will be primarily used on the platform will be the better one.

The overall design of the prediction code is shown in Figure 6.9. At the top of the figure is the PredictionMethod interface which is implemented by each of the prediction classes since it ensures that they all have a method for predicting a rating which is their primary purpose. They each have a different method for calculating predictions which will be described further in the coming sections, the only thing they have in common is that they receive a userID and a movieID for which a prediction is needed. The Predictor class uses each of these prediction method objects to either validate the model and calculate an error, or export a csv file of predictions for all users.

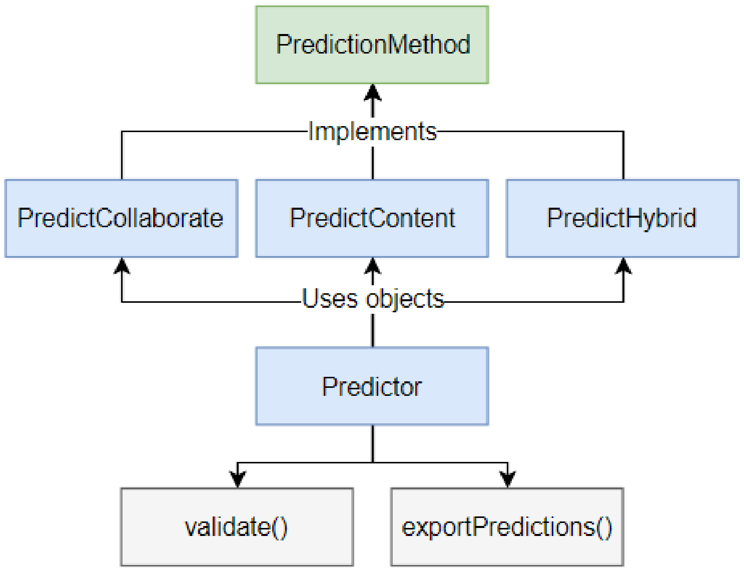


Figure 6.9: Program structure of classes and methods in the predictor code

6.3.2 Collaborative

The predictions made in the collaborative part are made from a formula based on the previously presented equations 4.18 and 4.19. Building upon these by adding the average rating of user *a* and mean centering the correlations as is shown in Equation 6.4.

$$pred(a,m) = \bar{r}_a + \sum\nolimits_{b \in N_m} w(a,b) \cdot (r_{b,m} - \bar{r}_b) \tag{6.4}$$

The weight of user *b*'s rating is defined by a normalized average of the correlation between users *a* and *b*. The method for calculating a prediction is shown in Listing 6.4 and receives a user ID and movie ID for which a rating is needed. The correlations between the user for which a rating is needed and all other users are imported in line 3 and iterated through in line 5. In each iteration it is tested whether the neighbor has rated the movie that a rating is being predicted for. If this is the case then the fitting variables are accumulated based on the rating and correlation to this neighbor. After having iterated through all neighbors it is checked whether more than 3 neighbors have rated the given movie or the denominator is 0 - if one of these is true then return the average overall rating, otherwise just return the predicted rating.

```
1  public class PredictCollaborate implements PredictionMethod {
2      public double predictRating(int userID, int movieID) {
3          LinkedHashMap<Integer, Double> userCorrelations = correlations.get(userID);
4
5          for (Integer neighborID : userCorrelations.keySet()) {
6              HashMap<Integer, Double> neighborRatings = seenBy.get(neighborID);
7
8              // Make sure that the user has rated this movie and a correlation between the users is stored
9              if (neighborRatings.containsKey(movieID)) {
10                 /*
11                 Accumulate results based on the current neighbor
12                 */
13             }
14         }
15
16         // If no more than 3 people have rated the movie or the denominator is 0 return average rating
17         if (numberOfUsersSeenMovie < 3 || weightSum == 0)
18             return avgOverallRatings.get(userID);
19
20         return predictedRating;
21     }
22 }
```

Listing 6.4: Predicting ratings using the the kNN method

6.3.3 Content-based

The predictions made by the content-based neural network are given directly from its output. It is trained to receive data about a user and movie, and from this output a rating as is illustrated in Figure 6.10. This means that as in the prediction using collaborate filtering, a userID and movieID is given to the predictContent method to get the correct userProfile and movieProfile for which a recommendation is needed. These act as inputs to the neural network which feeds this information forward through the trained network and a prediction is output.

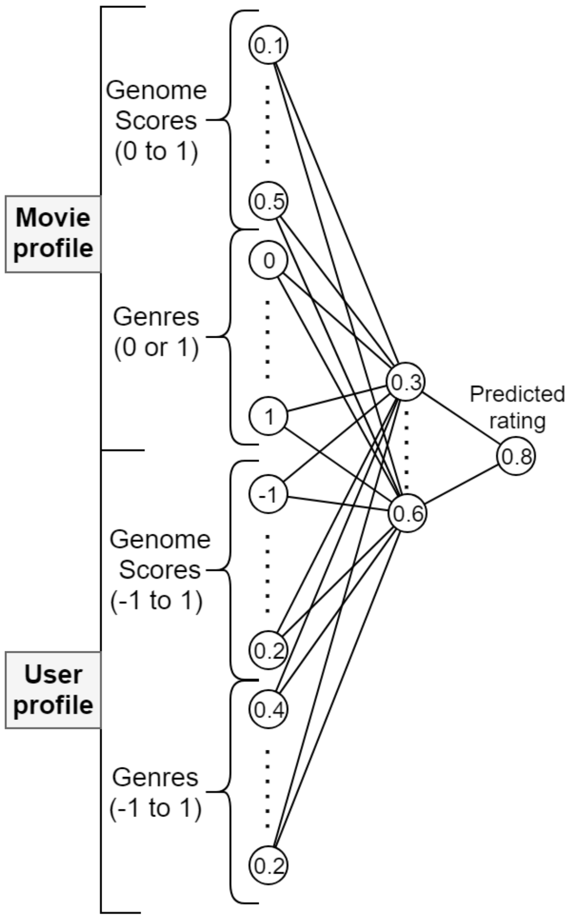


Figure 6.10: Illustration of how trained neural network can make predictions

6.3.4 Hybrid

When trying to connect the predictions made by the kNN and the ANN a hybrid between the two is made. The hybrid is able to weigh different solutions based on how good the individual prediction is. As we only need to combine two predictions the following formula is used:

$$pred_{hybrid} = pred_{ANN} * weight + pred_{kNN} * (1 - weight) \tag{6.5}$$

The formula weighs the prediction of the ANN with 'weight', and the rest is weighed towards the kNN.

Chapter 7

Testing

7.1 Validation

This section on validation is made for the purpose of arguing for the different implementation choices and to validate the results.

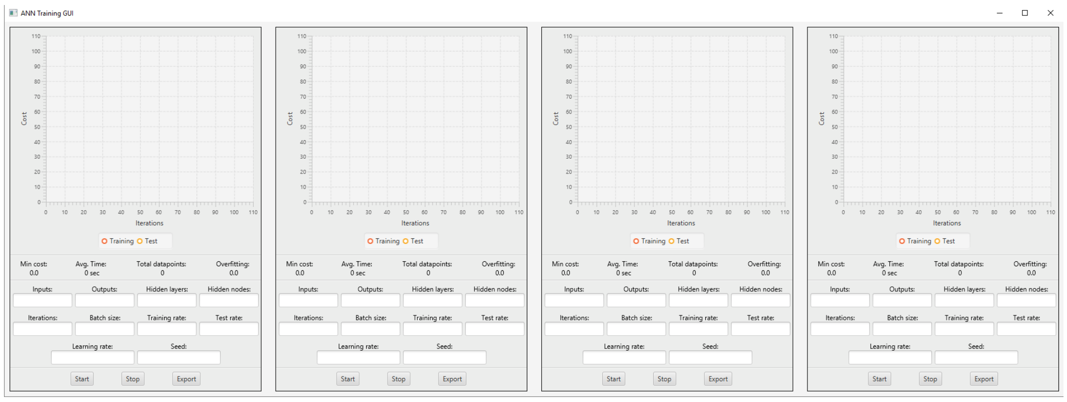
To verify our results, and make them comparable to other results found on MyMediaLite.net, we used the 5 fold cross validation technique to find more representative values for our final error-value.
This was done by randomly splitting the training data into 5 different equally sized sections, and then using each section as a validation set, and the other 4 sections as a single training set. This creates 5 different sets of both training and validation data with respective proportions of 80% and 20%.

All tests in this section was run on the same piece of hardware, therefore the computational time itself is not comparable to state of the art server hardware, but they are comparable relative to each other.

7.1.1 ANN Architecture

To help us define the free parameters of our model, we needed a method for fast and easy testing of our model after each change. The free parameters are parameters that must be estimated experimentally. Since editing the code for each change will often makes the process more confusing and sporadic, we chose to develop a "Test-Center" platform, which can be seen at Figure 7.1.

This platform allow us to train up to four different models simultaneously in parallel by utilizing multithreading within the platform, and make the task of changing the free parameters of the model quickly and easily. The platform even supports



Min cost: 0.0 | Avg. Time: 0 sec | Total datapoints: 0 | Overfitting: 0.0

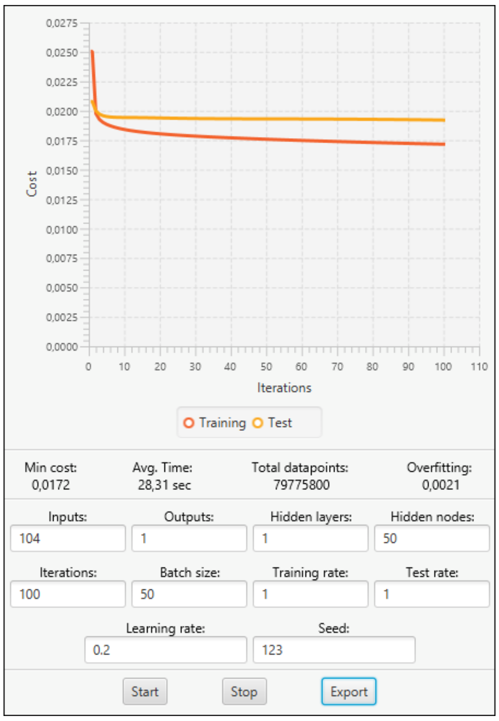
Inputs: | Outputs: | Hidden layers: | Hidden nodes:

Iterations: | Batch size: | Training rate: | Test rate:

Learning rate: | Seed:

Figure 7.1: The ANN Test-Center Platform.

exporting the currently trained model, along with the graph data and a snapshot of the training environment, which shows the settings, stats and the graph of the desired model. An example of this snapshot can be seen at Figure 7.2.



Min cost:	Avg. Time:	Total datapoints:	Overfitting:
0,0172	28,31 sec	79775800	0,0021

Inputs:	Outputs:	Hidden layers:	Hidden nodes:
104	1	1	50

Iterations:	Batch size:	Training rate:	Test rate:
100	50	1	1

Learning rate:	Seed:
0.2	123

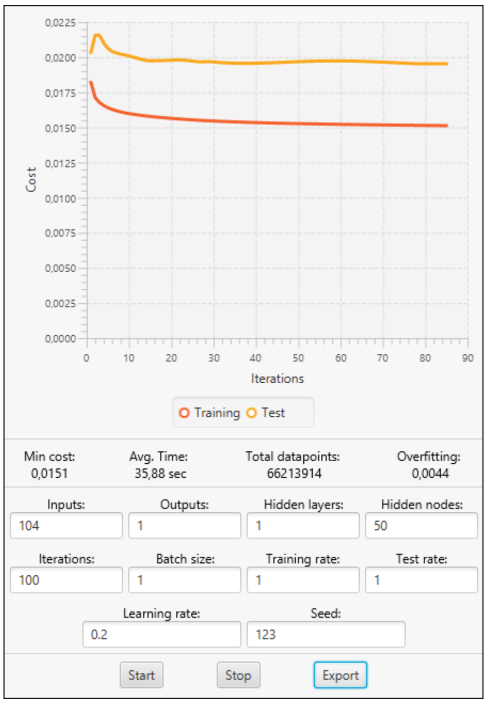
Figure 7.2: An example of the snapshot being exported by the Test-Center Platform.

Using this platform we tested the effect of changing different parameters of the model, such as the batch size.

As described earlier in section 3.1.2, when training a neural network, one of the challenges faced when using a neural network is non-convex problems. As mentioned using stochastic gradient descent or mini-batch might help overcome this problem, by making more but smaller adjustment for each iteration, by only back-propagating over a smaller set of data, instead of a complete iteration of the dataset.

On Figure 7.3a a snapshot of a network using stochastic gradient descent (mini-batch with size 1), can be seen. When looking at both the testing and training cost, it can be seen that they vary a lot in performance, which means that our platform is over-fitting. This is a result of the network in a larger degree learning to remember the training data, instead of learning the underlying model, which of course is a problem.

By increasing the batch size, we found that a batch size of 300 seems to more or less remove the over-fitting in the first 100 iterations, which can be seen on Figure 7.3b.

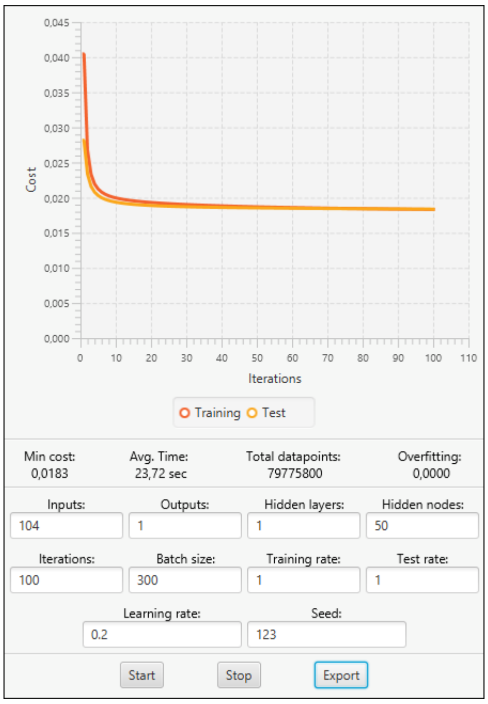


Min cost:	Avg. Time:	Total datapoints:	Overfitting:
0,0151	35,88 sec	66213914	0,0044

Inputs:	Outputs:	Hidden layers:	Hidden nodes:
104	1	1	50
Iterations:	**Batch size:**	**Training rate:**	**Test rate:**
100	1	1	1

Learning rate: 0.2
Seed: 123

(a) A snapshot of a model using a batch size of 1.



Min cost:	Avg. Time:	Total datapoints:	Overfitting:
0,0183	23,72 sec	79775800	0,0000

Inputs:	Outputs:	Hidden layers:	Hidden nodes:
104	1	1	50
Iterations:	**Batch size:**	**Training rate:**	**Test rate:**
100	300	1	1

Learning rate: 0.2
Seed: 123

(b) A snapshot of a model using a batch size of 300.

Figure 7.3: Showing snapshots of neural networks with different batch sizes

By using the same technique as above, we found that a neural network with the size 104-40-1 seems to have a good structure, 104 inputs, 40 hidden nodes and 1 output - and a learning-rate of 0.2 seems to be a good fit to also keep the training time relatively low.

7.1.2 ANN - Training over time

In this section a neural network with the size 104-40-1, and the same settings as described in section 7.1.1, will be used.

Training a neural network can be quite a time consuming task, so it often helps to know when to stop the training. This can be achieved by either using early stopping, which is a technique that stops the training when no progression have been made for a specified amount of iterations, or by simply stopping the training manually when no future improvement is expected.

To study the precision of our content based filtering method throughout its training, we tested the network every 10th iteration over 4000 iterations, which resulted in Figure 7.4

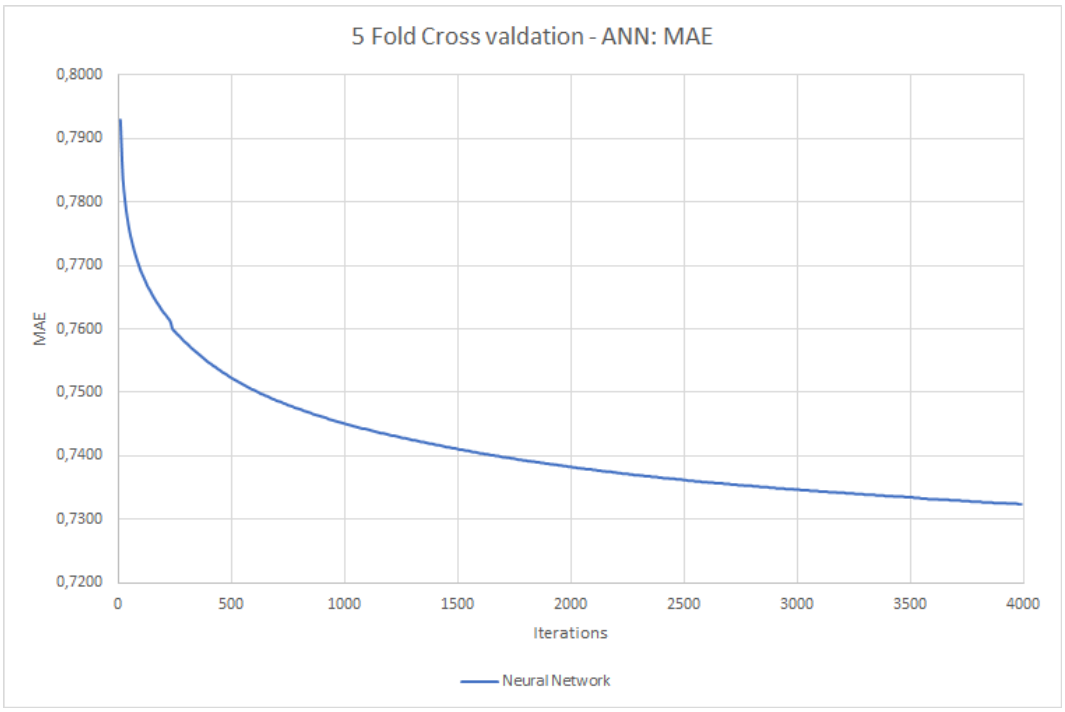


Figure 7.4: A 5 fold cross validation illustration of the prediction error of the ANN, as a development of the training iterations

The initial error at zero iterations were removed from Figure 7.4 since the value was too large (1.417) compared to every other value, and skewed the graph.

As Figure 7.4 shows, the model continues to learn even after 4.000 iterations of about 800.000 data points each, which is intriguing for our method, but as can be seen on both Figure 7.4 and Figure 7.5 the development is quickly slowing.

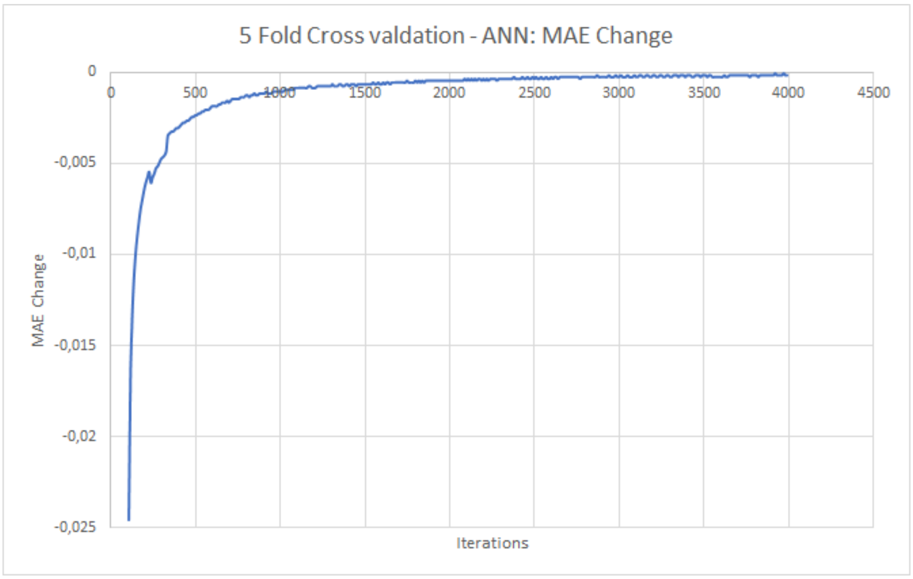


Figure 7.5: Shows the training development in the last 100 iterations.

As shown in Figure 7.5, the development after 4.000 iterations is very small, and after 4.000 iterations, the development in the last 100 iterations were only a mere -0.0002, which indicates that no large future development should be expected. While future improvement certainly is achievable, the time required to achieve a meaningful improvement might be too long, since 4.000 iterations in our tests require about 14 hours of training. According to our prognosis: More than 10.000 additional iterations, which equals about 35 hours of additional training, would be required to reach an error of about 0.72, which is only an improvement of about 1.7%.

If time is essential one might stop the training after about 2.500 iterations which would result in cutting the training time almost in half, and this only results in a precision-loss of about 0.5%.

7.1.3 kNN - Number of neighbors

The implemented kNN algorithm uses regression, which means that it looks at the specified k neighbors and allows their rating to have a weighed influence on the predicted rating - based on the correlation between them. Based on this it might seem that an increase in k thereby would also increase accuracy of the predictions. This would be the case because each neighbor is weighed based on their similarity as opposed to classification where only 1 or a few nearest neighbors is optimal which was previously described in Section 4.5.3. It would thereby seem like the optimal number of neighbors would be to look at all other users, but as will soon become apparent this is not the case. This all means that to create an optimal kNN algorithm in the given situation, a fitting k should be chosen by looking at both accuracy of predictions and computational time with different values of k.

To find the most fitting number of neighbors, different values of k will be tested and based on both prediction error and computational time of predictions, the most optimal one will be discussed and chosen.

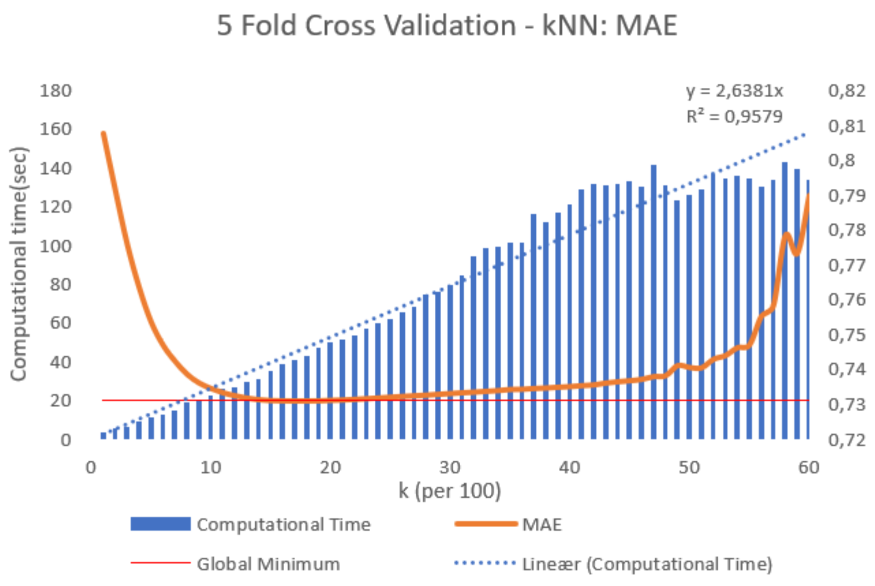


Figure 7.6: Shows how the MAE and computational time develops as we look at more neighbors

Figure 7.6 illustrates that as the number of neighbors increase by 100 the computational time of predicting ratings with pre-calculated correlations increase by 2.6 seconds - as it is linear. The MAE decreases at first, but quickly starts increasing again. This might be caused by the fact that at that point we start to look at neighbors with negative correlations. This means that we look at neighbors with both negative and positive correlations in the same calculation which might cancel out the effect and not achieve the wanted effect. To optimize this, one might have made some predictions based on only those with positive correlations and only those with negative correlations, and afterwards combined these predictions.

Figure 7.6 show that the computational time of prediction ratings with 1700 neighbors is around 40 seconds, but as this time grows linearly with the number of neighbors that we look at, it is not very costly to increase the value of k. The single most telling piece of information in the figure is that the MAE is 0,7309 at its lowest point at 1700 neighbors. Due to the fact that the MAE is nowhere near as low as that with a k-value lower than 1700 neighbors it seems obvious that this is the optimal number of neighbors in the given situation.

7.1.4 Hybrid - Weights

In previous sections it has been described how ANN and kNN individually can get the highest prediction accuracy, but as previously mentioned it is expected that a combination of the two can make even more accurate predictions. We assume that the best individual configurations also will work the best in a hybrid method. The formula for calculating this was described in Section 6.3.4 which also specifies that a weight for each of the two methods has to be chosen. To do this we use the network size, number of iterations of the network and value of k that was chosen in the previous section. By using these the best prediction accuracy can be found by depicting it as a graph.

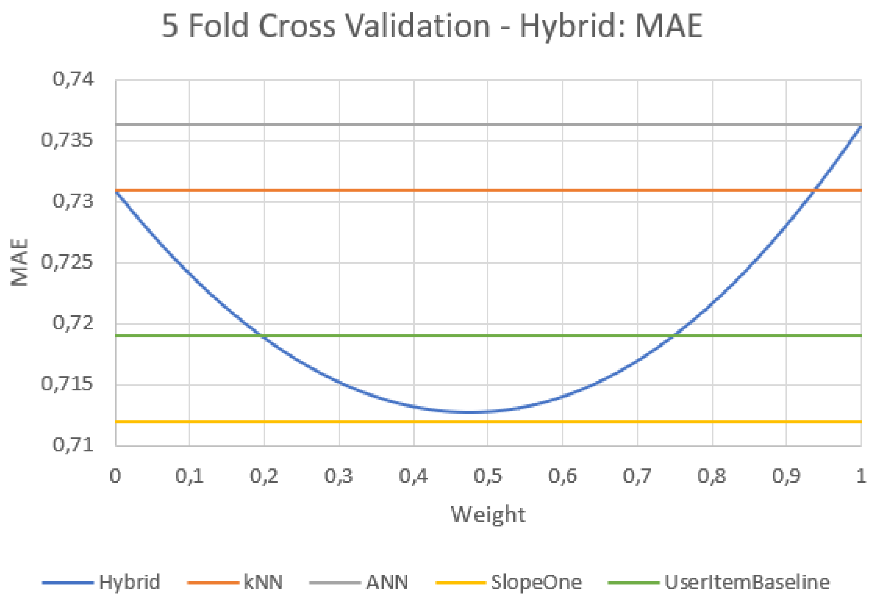


Figure 7.7: Shows how the ANN(2500 iterations) and kNN(1700 neighbors) complements each other in a hybrid solution, and compares the result to other methods using the same dataset

In Figure 7.7 the mean average error at weights between 0 and 1 has been depicted as the blue line described as 'hybrid'. At a weight of exactly zero the ANN is weighed 0 and the kNN is weighed 1 which means that that is its best MAE score at 1700 neighbors. On the opposite side at a weight of 1 the ANN is now weighed 1 and is therefore its best MAE score. As the figure illustrates, these methods complement each other and achieves a better MAE score when combining the predictions of the two. When using a weight of 0.48 on the ANN and 0.52 on the kNN a MAE score of 0.713 is achieved which is the lowest point and thereby the best weight.

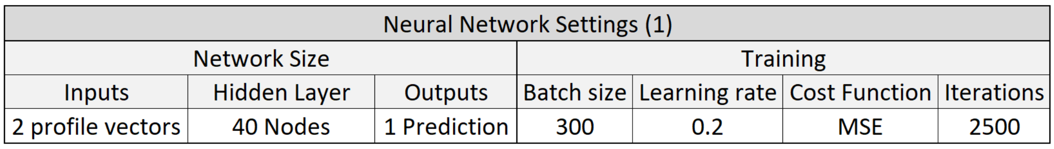
The hybrid, ANN and kNN methods are shown in reference to some of the methods with similar MAE scores from MyMediaLite [50] - also in Figure 7.7. Both the Slope One method and User-Item Baseline method are more simple approaches than both ANN and kNN, but this still shows that the described implementation is comparable to other implemented methods on the same data.

7.1.5 Recap

In this section, a recap of our previous results and findings will be represented and shortly discussed.

Recommender method settings

On Figures 7.8, 7.9 and 7.10 the settings used in our final methods can be seen. These settings were developed through our previous sections, by experimantally trying different ones.



Neural Network Settings (1)						
Network Size			Training			
Inputs	Hidden Layer	Outputs	Batch size	Learning rate	Cost Function	Iterations
2 profile vectors	40 Nodes	1 Prediction	300	0.2	MSE	2500

Figure 7.8: The settings used by our content based method, implementing an ANN.

The inputs of our neural network are 2 profile vectors, which each equals to 52 different attributes defining both genome tags and genres, which means our input layer has a total of 104 input nodes.

Our output layer has a single output node, which returns the prediction, but in a range from 0 to 1, where 0.2 is 1, and 1 is 5.

In our kNN method, when calculating the correlations, we use a setting "Min. Common Movies" which requires that two users need at least 7 common movies, which means that they need to both have rated 7 of the same movies, before a

k-Nearest Neighbor settings (2)				
Correlation Settings			Prediction	
Method	Type	Min. Common movies	Neighboors	Min. Relevant Neighboors
Pearson	Regression	7	1700	3

Figure 7.9: The settings used by our collaborate based method, implementing kNN.

correlation could be calculated - since a very limited amount of common movies could result in misleading correlations.

When using our kNN method to predict ratings, we use another setting "Min. Relevant Neighbours", to avoid making predictions on too sparse information. This setting requires that you have at least 3 other neighbours who have seen the movie, before a prediction can be made. If this requirement is not met, the average rating of that user is returned instead.

Hybrid settings (3)		
Content	Collaborate	Weight
Same as (1)	Same as (2)	0.48

Figure 7.10: The settings used by our hybrid based method, implementing both an ANN and kNN.

Our hybrid method is a combination of both content and collaborative recommendations. Each of them have a vote, and by applying a weight to these votes, a combined prediction is made.
The formula used to weigh the methods can be seen at Equation 6.5.

Recommendation method performance

A quick recap of the performance of both error and speed can be seen at Figure 7.11.

Method	Predictions / Sec	Relative	MAE	Relative
ANN	105.000	17,2	0,73627	1,033
kNN	6.700	1,1	0,73091	1,026
Hybrid	6.100	1,0	0,7127	1,000

Figure 7.11: A comparison of all our recommendation methods, on both error and speed.

As seen on Figure 7.11, the hybrid method is superior in predicting the ratings, which is shown by having the least error (MAE), but on the other hand is far inferior in prediction speed, which is over 17 times as slow as the ANN alone.

Depending on how much time is available to predict ratings, one might choose a lesser prediction method, which in return has a higher prediction rate, and therefore can more quickly calculate all the necessary predictions.

7.2 Unit testing

To ensure the reliability and accuracy of our results a form of testing has to be made on the different parts of the program. Unit tests on some of the most essential parts of the recommender system will be described further in this section. The reliability that the program itself runs is important but these kinds of errors will often quickly become apparent as opposed to small accuracy errors that would rarely be found. These accuracy errors are the most essential errors to catch and are therefore the ones that will be put into focus and described further in this section.

```java
@Test
public void predictionTest(){
    /*
    *   Initialize  content based prediction method - Hidden from example
    */

    // Load the user and move vector, for user 1 and movie 1.
    double[] userVector = content.getUserProfileList().get(1).convertToArray();
    double[] movieVector = content.getMovieProfileList().get(1).convertToArray();

    /*
    *   Load input data from user and movie vector - Hidden from example
    */

    // Sum all input values
    // Will be used when calculating the expected prediction.
    double sumOfInputs = 0;

    for (Double value : input) {
        sumOfInputs += value;
    }

    // Calculate expected prediction.
    ActivationFunction sigmoid = new SigmoidActivation();
    double expected = sigmoid.activation(sigmoid.activation(sumOfInputs) * 2) * 5;

    // Calculate the actual prediction.
    double prediction = content.predictRating(1, 1);

    // Assert that the expected and actual prediction are equal.
    Assertions.assertEquals(expected, prediction, 0.001);
}
```

Listing 7.1: Predicting test using the content based method, ANN

The java code example shown in Listing 7.1, shows one of our unit tests which tests that our content based method returns the correct prediction given some static parameters. First a very simple ANN is imported which makes it easier to calculate the expected prediction. Next we import the necessary information to calculate both the expected and actual prediction to check if they are equal. If they are equal, then this part of the implementation of our content based method is correct.

This test can be rerun every time we make an alternation or addition to the algorithm, and if it still returns the expected rating, then the changed version should still be correct.

An important side note is that unit tests can never confirm the absence of errors, only prove their existence. This means that just because every test returns true, it does not confirm that the program is correct, only that it most probably is.

Chapter 8

Conclusion and reflections

8.1 Conclusion

In the task of making a recommendation system that relatively accurately can recommend movies based on a set of data points, the results are in favor of using a hybrid solution that uses several different recommendation methods to weigh the prediction and thereby use the strengths from each method. In our case only two different methods were used but the results point towards the fact that if more methods with similar or better prediction accuracies were used, the hybrid predictions might have been even better.

Our implementation of the kNN algorithm based on Pearson correlations is known for being easily implemented but no longer used as state of the art - primarily because of scalability and prediction accuracy. This clearly manifested itself as the calculation of Pearson correlations from all users to all users has a time-complexity of $\Theta(n^2)$. The calculation of predictions has linear complexity, $\Theta(n)$, which means that an increase in the number of users also will increase the computational time of predictions. More users will also allow for the ability to look at even more neighbors, and because collaborative filtering is dependant on knowing the preferences of as many relevant users as possible it might even allow for increased prediction accuracy.

The use of neural networks to recommend movies is not yet state of the art, and kNN is no longer state of the art. The comparison of the two reveals that artificial neural networks might have potential to gain some ground in the industry as its rating results were comparable to our implementation of kNN and a few other methods that used the same dataset and validation technique. It also makes recommendations far quicker than our implementation of kNN and is highly scalable which is often an important feature on larger user bases.

8.2 Reflections

8.2.1 Expanding on the social aspect

One of the main focuses when first defining the problem was that the social aspect of movie interests was not harnessed by other services. Due to time constraints as described in Section 2.6 this was de-emphasized. In further development on the project this would be one of the main points of interest as the project only really becomes unique when adding this feature. The way of implementing this would be to use the collaborative filtering method to find the users with the most similar movie interests and recommend them as friends, while allowing to add friends and recommend movies based on them.

8.2.2 Other recommendation methods

A very popular and currently state of the art method is SVD, which is very interesting since it draws many similarities to our implementation of ANN. It's important to know that SVD in the context of recommendations and machine learning rarely is the original definition of Singular Value Decomposition, where a matrix is decomposed into 3 other matrices, which multiplied results in the original matrix. Instead a method where the matrix is decomposed into two different matrices which by applying stochastic gradient descent learns the two matrices which approximates the original matrix by minimizing a cost function.

This method draws many similarities to the ANN method. While the SVD compared to our ANN implementation is superior, certain drawbacks also apply to the SVD compared to the ANN.
Whenever a new user or movie is added to the system, the two learned matrices must be retrained, which in comparison is not the case when implementing our ANN method. This advantage of the ANN also implies one of its drawbacks, which is that whenever a user rates a movie, the profile vector of that user should be recalculated, which is not necessary with the SVD.

The recalculation of the profile vector is not necessarily required, since using the same profile vector for a longer period of time is not forbidden. Equivalently updating the SVD matrices is not required after each addition to the matrix they approximate.
The approximation quality might decrease for each addition, but it will most probably still remain an okay approximation. By applying this policy, the mentioned drawback of the ANN will more or less vanish.

8.2.3 Cold starts

Both our recommendation methods, ANN and kNN, are dependent on previous user ratings to create new predictions. This means that our system is vulnerable to the cold start problem, which is when a new user sign up on our platform. At first the new users will have zero previous data to get recommendations, which means that the kNN cant find any neighbors, and the ANN does not yet have a user vector profile as input, and therefore cant reliably predict anything.

To address this problem we require that at sign up, new users provide some data about their movie preferences - such as rating different genres. This data can help our us create an initial user profile vector, and help us locate neighbors before suggesting movies to watch.

Another method to collect data, is using an implicit method. By analyzing the users behaviour, it's possible to derive certain traits. If a user only watches the first half hour of a movie, and then never watches the rest, it's fair to assume that he did not like that particular type of movie, even though he did not rate it himself. Likewise if a user watches a movie to the end, or watches the movie multiple times, then it's fair to assume that the user liked that movie, even though he did not rate the movie himself.

Besides having a cold start for individual new users, the platform itself will have a cold start at the beginning of its lifespan, when only few users are registered on the platform. This will decrease the performance of the kNN, as it requires neighbors and only few are available. A suggestion to address this problem, is to completely ignore the kNN or give it a very low "voting weight" in the hybrid prediction. The content based method will not have the same drawback, since it does not, in the same way, rely on other users to generate predictions.

Bibliography

[1] Barry Schwartz. *The paradox of choice.* https://www.ted.com/talks/barry_schwartz_on_the_paradox_of_choice?language=da#t-833280. July, 2015.

[2] Marcos Suliveres. *Social Media: The Death of Real World Interaction?* https://medium.com/musings-of-a-writer/social-media-the-death-of-real-world-interaction-5e2f33cfd8ee. June 5, 2014.

[3] Wikipedia. *Social Media.* https://en.wikipedia.org/wiki/Social_media#cite_note-SMDefinition-1. February 28, 2018.

[4] Pew Research Center. *Demographics of Key Social Networking Platforms.* URL:http://www.pewinternet.org/2015/01/09/demographics-of-key-social-networking-platforms-2/. January, 2015.

[5] Victor Luckerson. *This Is How Much Netflix We're All Watching Every Day.* http://time.com/4186137/netflix-hours-per-day/. January 19, 2016.

[6] Katie Young. *Social Media Captures Over 30% of Online Time.* https://blog.globalwebindex.net/chart-of-the-day/social-media-captures-30-of-online-time/. September 11, 2017.

[7] Daniel Danker. *Introducing Watch, a New Platform For Shows On Facebook.* https://newsroom.fb.com/news/2017/08/introducing-watch-a-new-platform-for-shows-on-facebook/. August 9, 2017.

[8] weforum.org. *How does digital media really affect us?* https://www.weforum.org/agenda/2016/01/how-does-digital-media-really-affect-us/. January 28, 2016.

[9] Richard Alleyne. *Shared interests key to friendship.* http://www.telegraph.co.uk/news/science/science-news/8171136/Shared-interests-key-to-friendship.html. December 01, 2010.

[10] Finder. *Netflix USA vs The World: Content libraries compared.* https://www.finder.com/netflix-usa-vs-world-content. February 27, 2016.

[11] Stuart Jeffries. *Why too much choice is stressing us out.* https://www.theguardian.com/lifeandstyle/2015/oct/21/choice-stressing-us-out-dating-partners-monopolies. October 21, 2015.

[12] sigmoidal.io. *Recommendation Systems – How Companies are Making Money*. https://sigmoidal.io/recommender-systems-recommendation-engine/.

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

[14] Kurt Nørmark. *P2 Projektet*. http://people.cs.aau.dk/~normark/p2-f18/p2-katalog.pdf. January, 2018.

[15] Jochen Eckert Oliver Hinz. *The Impact of Search and Recommendation Systems on Sales in Electronic Commerce*. https://link.springer.com/article/10.1007/s12599-010-0092-x. April, 2010.

[16] Miller McPherson, Lynn Smith-Lovin, and James M Cook. "Birds of a Feather: Homophily in Social Networks". In: *Annual Review of Sociology* 27.1 (2001), pp. 415–444. DOI: 10.1146/annurev.soc.27.1.415. eprint: http://arjournals.annualreviews.org/doi/pdf/10.1146/annurev.soc.27.1.415. URL: http://arjournals.annualreviews.org/doi/abs/10.1146/annurev.soc.27.1.415.

[17] Richard M. Ryan and Edward L. Deci. "Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being". In: *American Psychologist* (2000), pp. 68–78.

[18] Steven Mazie. *Do You Have Too Many Facebook Friends?* http://bigthink.com/praxis/do-you-have-too-many-facebook-friends. 2016.

[19] Dave Chaffey. *Global social media research summary 2018*. https://www.smartinsights.com/social-media-marketing/social-media-strategy/new-global-social-media-research/. 8 February, 2018.

[20] KC Ifeanyi. *Will Facebook Watch Be Must-See TV In 2018?* https://www.fastcompany.com/40510962/will-facebook-watch-be-must-see-tv-in-2018. March, 2018.

[21] Francesco Ricci et al. *Recommender System Handbook*. Springer, 2011, pp. 1–17. ISBN: 978-0-387-85819-7.

[22] Wikipedia. *Recommender system*. https://en.wikipedia.org/wiki/Recommender_system. February 27, 2018.

[23] Libby Plummer. "This is how Netflix's top-secret recommendation systems works". In: *Wired* (22 August 2017).

[24] Subana Shanmuganathan and Sandhya Samarasinghe. *Artificial Neural Network Modelling*. 1st. Springer Publishing Company, Incorporated, 2016. ISBN: 3319284932, 9783319284934.

[25] Frontiers in Computational Neuroscience. *Artificial Neural Networks as Models of Neural Information Processing*. https://www.frontiersin.org/research-topics/4817/artificial-neural-networks-as-models-of-neural-information-processing. 19 December, 2017.

[26] *The Evolution and Core Concepts of Deep Learning & Neural Networks*. https://www.analyticsvidhya.com/blog/2016/08/evolution-core-concepts-deep-learning-neural-networks/. 3 August, 2016.

[27] Hernane Spatti D. Andrade Flauzino R. Liboni L.H.B. dos Reis Alves S.F da Silva I.N. *Artificial Neural Networks A Practical Course*. 1st. Springer Publishing Company, Incorporated, 2017. ISBN: 9783319431628.

[28] AnimatLab. *Neuron Basics*. http://animatlab.com/Help/Documentation/Neural-Network-Editor/Neural-Simulation-Plug-ins/Firing-Rate-Neural-Plug-in/Neuron-Basics.

[29] Towards Data Science. *Activation Functions: Neural Networks*. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6. 6 September, 2017.

[30] Wikipedia. *Genetic algorithm*. https://en.wikipedia.org/wiki/Genetic_algorithm.

[31] Moustafa Alzantot. *Episode 1 — Genetic Algorithm for Reinforcement Learning*. https://becominghuman.ai/genetic-algorithm-for-reinforcement-learning-a38a5612c4dc. 8 June, 2017.

[32] LEE JACOBSON. *Creating a genetic algorithm for beginners*. http://www.theprojectspot.com/tutorial-post/creating-a-genetic-algorithm-for-beginners/3. 12 February, 2012.

[33] Tony B. *genetic algorithms in PHP code*. http://www.abrandao.com/2015/01/simple-php-genetic-algorithm/. 21 January, 2015.

[34] Vijini Mallawaarachchi. *Introduction to Genetic Algorithms—Including Example Code*. https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3. 8 July, 2017.

[35] "cyborg". *Is a genetic algorithm a form of unsupervised learning? [closed]*. https://stackoverflow.com/questions/20752232/is-a-genetic-algorithm-a-form-of-unsupervised-learning. 23 December, 2013.

[36] tutorialspoint.com. *Genetic Algorithms - Introduction*. https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_quick_guide.htm.

[37] Dale Thomas. *What are the disadvantage of genetic algorithm?* https://www.quora.com/What-are-the-disadvantage-of-genetic-algorithm. 25 February, 2016.

[38] Michael Nielsen. *How the backpropagation algorithm works*. `http://neuralnetworksanddeeplearning.com/chap2.html`. December, 2017.

[39] Daniel Sabinasz. *Deep Learning From Scratch IV: Gradient Descent and Backpropagation*. `http://www.deepideas.net/deep-learning-from-scratch-iv-gradient-descent-and-backpropagation/`. October 21, 2015.

[40] Wikipedia. *How the backpropagation algorithm works*. `https://en.wikipedia.org/wiki/Backpropagation`. 5 March, 2018.

[41] Matt Mazur. *A Step by Step Backpropagation Example*. `https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/`.

[42] cs.cornell.edu. *Non-Convex Optimization*. `http://www.cs.cornell.edu/courses/cs6787/2017fa/Lecture7.pdf`. Fall, 2017.

[43] MathOnline. *The Distance Between Two Vectors*. URL:`http://mathonline.wikidot.com/the-distance-between-two-vectors`.

[44] Charu C. Aggarwal. *Recommender Systems: The Textbook*. 1st. Springer Publishing Company, Incorporated, 2016. ISBN: 3319296574, 9783319296579.

[45] Herbert J Bernstein and Lawrence C. Andrews. "Acceleratingk-nearest-neighbor searches". In: *Journal of applied crystallography* 49.5 (), pp. 1471–1477.

[46] GroupLens. *MovieLens Datasets*. URL:`https://grouplens.org/datasets/movielens/1m/`. February, 2003.

[47] GroupLens. *What is GroupLens?* `https://grouplens.org/about/what-is-grouplens/`. Accessed: 2018-02-27.

[48] Jesse Vig, Shilad Sen, and John Riedl. "The Tag Genome: Encoding Community Knowledge to Support Novel Interaction". In: *ACM Trans. Interact. Intell. Syst.* 2.3 (2012), 13:1–13:44. ISSN: 2160-6455. DOI: `10.1145/2362394.2362395`. URL: `http://doi.acm.org/10.1145/2362394.2362395`.

[49] Oracle. *JavaFX Frequently Asked Questions*. `http://www.oracle.com/technetwork/java/javafx/overview/faq-1446554.html`. May, 2018.

[50] MyMediaLite. *MyMediaLite: Example Experiments*. URL:`http://www.mymedialite.net/examples/datasets.html`.