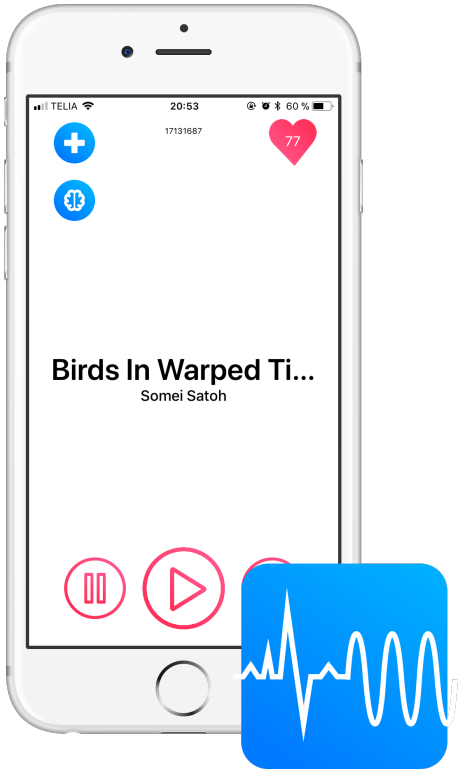
CHALMERS
UNIVERSITY OF TECHNOLOGY



Music Recommendations Based on Real-Time Data

Bachelor of Science Thesis in Computer Science and Engineering

MARCUS AURÉN
ALBIN BÅÅW
TOBIAS KARLSSON
LINNEA NILSSON
DAVID HAGERMAN OLZON
PEDRAM SHIRMOHAMMAD

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden, May 2018

Music Recommendations Based on Real-Time Data

Bachelor of Science Thesis in Computer Science and Engineering

MARCUS AURÉN
ALBIN BÅÅW
TOBIAS KARLSSON
LINNEA NILSSON
DAVID HAGERMAN OLZON
PEDRAM SHIRMOHAMMAD

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden, May 2018

Abstract

This thesis describes the development, implementation and results of a music recommender system that utilizes real time data, namely time and heart rate, for the recommendations. The recommender system was made by combining two systems, the recommender system which predicts a number of song features for a specific user and a ranking system which finds the best matching tracks for these features. Three implementations of the recommender system were implemented for comparison, namely Deep Neural Network, Contextual Bandit and Linear Regression. These implementations were tested with offline evaluation which showed that for our problem, a contextual bandit model had the best accuracy.

Keywords: Recommender system, music recommendations, neural network, deep learning, reinforcement learning, contextual bandit, linear regression, deep neural network

Acknowledgements

First of all, we want to thanks our supervisor K V S Prasad for his guidance and pep talks. We also want to thank Mikael Kågebäck for his insightful suggestions regarding machine learning, Niklas Broberg for taking the time to help us get back on track and Lars Norén for assisting with hardware. Finally, we want to thank Fackspråk for helping us with insight in the art of technical writing.

Marcus Aurén
Albin Bååw
Tobias Karlsson
Linnea Nilsson
David Hagerman Olzon
Pedram Shirmohammad

Gothenburg, Sweden, May 2018

Sammanfattning

Den här rapporten beskriver utvecklingen, implementeringen och evalueringen av ett musikrekommendationssystem som använder sig av realtidsdata så som tid och puls som en faktor i sina rekommendationer. Rekommendationssystemet består av två separata delsystem, ett som rekommenderar ett antal sångattribut och ett rankingsystem som sorterar alla låtar i databasen baserat på de rekommenderade attributen. Tre olika typer av rekommendationssystem har implementerats för utvärdering. Ett baserat på ett djupt neuralt nätverk, en kontextuell bandit och en linjär regression. Alla tre implementationer utvärderades sedan med hjälp av offline-evaluering som visade på att den kontextuella banditen gav oss bäst träffsäkerhet.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With today's music streaming services, enjoying a wide array of music is more accessible than ever. You don't have to buy your own copies of the records, but can explore whole libraries of music from your computer, tablet or smartphone, and at least in theory this makes it easier to find new favourites. The vast number of songs may be hard to survey, which is why most streaming services offer recommender systems that aim to figure out what a specific user might be interested in listening to. Our main thought when starting this project was that these recommendations are seldom very accurate, therefore we started thinking about ways of making better predictions of songs that the user actually wants to hear.

In the project, we explore if we can get better recommendations by using real-time data, specifically a user's heart rate and the time of day, when making recommendations. The system making the recommendations will be implemented using several machine learning techniques and will be presented through a mobile application. The system uses a smart watch to recognize the user's heart rate in order to give recommendations of songs according to what kind of music is usually associated with that heart rate and time of day for that specific user. For instance, if a user is out running, the user's heart rate is probably higher than normal. The intention is that the system will learn that this user often takes runs in the morning, and that they prefer to listen to e.g. fast and loud music while exercising. To get this recommendation the user presses the play or next button in the mobile application which captures the heart rate and current time and sends it to the system. The recommender system has been trained to recognize patterns in the user's listening behaviour. It takes in these parameters and matches them with other metrics such as the audio features of a song in order to make a recommendation to the user. The best matching song is then sent back to the mobile application which plays it to the user.

This thesis covers how we have implemented three versions of this type of recommender system and the results of the different approaches. The thesis starts with a brief background based in current research and moves on to explaining further why this project is relevant. The following sections in the introduction consists of the purpose and scope of the project, explaining the goals and delimitations when developing the system. Chapter 2 consists of the theory behind the ideas, concepts and algorithms applied in this project. In chapter 3 the implementation is described with detailed descriptions of all parts of the system as well as how they are all connected. In chapter 4 the results of the project is presented. In chapter 5 the results are discussed and analyzed, and the project as a whole is discussed. In chapter 6 the discussion is narrowed down to form a conclusion encompassing the entire project.

1.1 Background

The volume of data on the web is growing at an increasing rate and filtering through the data can be overwhelming, thus the need for personalized recommendations is greater than ever. With the rise of the Internet of Things

and because a growing amount of people are connected to the internet, a large amount of information can be gathered. For instance, the GPSes in our cars and phones gathers information about our location and movement patterns. Another example is that phones together with pulse monitoring devices, such as smart watches, or connected medical devices gather medical data such as heart rate or blood pressure. Even data of eating habits can be gathered by smart fridges and kitchen appliances. This continuously changing information can infer activity and mood which affect our preferences on a real-time basis.

Recommender systems today are usually based on a user's feedback history regarding a specific group of items. Such a service presents its users with a number of recommendations but will give the same recommendations regardless of the users' current activity, mood or location. Attempts at developing recommender systems that take contextual information into consideration have been made. Two examples of this includes systems that considers the day of the week, location and the user's companions, as well as systems that are aware of the user's activity. Both approaches have shown success compared to systems that do not consider these aspects [38] [36]. Therefore it is likely that taking the aforementioned parameters into account which suggests the user's mood and activity gives more accurate recommendations.

Real-time data changes constantly but data connected to a person based on real-time parameters often follow certain patterns. Humankind is a creature of habit and as such we do not tend to deviate from these patterns randomly. As real-time data can change quickly and an algorithm based on real-time data must be efficient. We want recommendations that are appropriate for the current state and not previous states. Machine learning techniques such as neural networks are currently the focus of a lot of research, and they are also becoming increasingly popular in the field of recommender systems [33]. They can not only handle the ever growing amount of data, but due to the learning algorithms, actually improve in quality in correlation to the amount of data considered. As the amount of data has grown and computational power of computers has increased, machine learning has become a far more viable field than it has been historically. The algorithms are designed to find patterns and therefore a feasible choice for a recommender system that incorporate real-time data parameters. As the interest in the field of machine learning has increased developer tools and libraries have been created which has led to machine learning becoming a more approachable subject for developers without expertise in machine learning.

Real-time sources of data are especially relevant for music recommendations because the correlation between our music preferences and our current emotional state is high. Certain songs or types of music can alter our moods in different ways and our preferences regarding music are therefore often connected to our mood [20][3]. Music choices also relate to the listeners current activity. Although we might have a certain taste in music, our preferences will change according to what we are doing. For instance a person will probably listen to different songs when they are working out at the gym compared to when they are trying to fall asleep at night [8].

Our heart rate is connected to both our emotional state and our activity level and studies have shown that listening to different types of music affects a person's heart rate [35] [12]. This, together with the fact that heart rate is an easy to access real-time parameter makes it an optimal source of real-time data to incorporate into this project.

1.2 Heart Rate and Music

There are several studies on how heart rate is affected when listening to music with different tempos which indicate that heart rate changes according the tempo of the music your are listening to [35] [22] [20]. The fact that a user's heart rate is increased when listening to high tempo music is not an indication of that the users music preferences is changing. However, there is additional research that indicate that there is a relationship between a person's heart rate and preferred tempo. In 1995, Iwanaga M found an harmonic relationship between the heart rate and preferred tempo that showed that the preferred tempo increased as the heart rate increased [25]. Several studies has since then been made on preferred music tempo while exercising and the results are

quite conclusive and support the previously mentioned relationship between heart rate and preferred tempo [8] [21]. The intent of our system is not to change a user's heart rate but instead use it as one of the points of data we base our recommendations on.

1.3 Purpose

The purpose of this project is to create a service which recommends music to a user. The system gives user-specific recommendations using machine learning and real-time data, mainly the users current heart rate but also the time of day. The project also serves to examine a few machine learning techniques to find which gives the most accurate prediction for this specific problem. To be able to examine the machine learning techniques in a meaningful way, extensive research on the application of machine learning in recommender systems has been done. In an effort to examine different machine learning techniques, three techniques has been implemented and evaluated in this project.

1.4 Scope

We have used an already existing streaming service in order to retrieve and play music as it is not in the scope of the project to develop our own music streaming service. We decided to use Spotify because of their extensive API:s and SDK:s which are easy to use. Spotify also offers access to a large amount of audio feature data connected to the music which we retrieve and use in the recommender system.

On the other hand, by using Spotify we affect how our application functions. For instance, we use the song ids associated with the music which we retrieve from Spotify. The song ids are used to identify the songs, this leads to problems in case we would want to switch to another streaming service in the future. Other services most likely generates their song ids in a different way which would mean that we would have to rewrite the part of the code that connect a song id to a specific song. As we also use the audio features provided by Spotify, using another service would require us to change how we get audio features. Exactly what changes would be necessary depends on how, and if, that service provides audio features. Another downside of using Spotify is that they are utilizing a paywall, i.e. the user needs to own a Spotify "Premium"-account to be able to play the music in our application. Lastly, to be able to perform experiments we have decided to restrict the amount of songs played by the app to a single playlist containing 1124 tracks. The songs are from a wide range of genres and types of music so that it mimics the entire Spotify library. A smaller library of songs allows us to easier see whether the recommender system is learning and in what direction which would be difficult with the huge library of music that Spotify has.

When it comes to the mobile application we decided to only develop a mobile application for the iOS-platform. The reason for this is because of availability and because it's not currently in the scope of the project to develop and maintain two separate mobile applications. On the other hand, the backend will in no way be limited to iOS only which makes it possible to develop similar mobile application to other platforms in the future. To measure heart rate we are using the Apple Watch due to both availability and because of how the functionality of the watch connects with an iPhone as it allows us to easily get the current heart rate. Even though the Apple Watch does not normally measure heart rate continuously it is possible to force the clock to do so.

As we do not have an existing data set of user data that is applicable for this problem and only limited resources for creating our own user data, the amount of data will be limited. Machine learning algorithms often grow more accurate when there is more data to process and our lack of data might affect the performance of our algorithms. In an effort to mitigate this problem, the recommender system will only be predicting three audio features namely mode (major or minor), loudness and tempo. Training the recommender system for fewer audio features will require less data and make it possible to get more accurate results despite our lack of data. The main focus of the project is to implement the recommender system and evaluate how different machine learning

techniques would perform in a larger system.

There are several other real-time parameters that could be taken into account apart from heart rate and time in order to possibly achieve better music recommendations, for instance the users location. However, we have decided to not implement more than specifically heart rate and time based on the assumption using more parameters would not increase the accuracy enough to warrant the time needed for such an implementation. As one of the goals of the project is to test a few different machine learning techniques there will be three implementations of the recommender system which are contextual bandit, deep neural network and linear regression. We decided to use these three algorithms because of their wide use, both in general and in recommender systems in particular, and because they are rather different from each other.

Chapter 2

Theory

To give a deeper understanding of the subject, this chapter will present recommender systems in general as well as the most common filtering approaches they use. Methods of evaluating recommender systems will also be presented. The recommender systems in this project builds upon content-based filtering, but collaborative filtering is also explained for further depth and because it is relevant for our discussion. Due to our lack of users, collaborative filtering can not be implemented in a meaningful way although we would have liked to implement this alongside content-based filtering.

We have used three different machine learning techniques to implement our recommender system, namely a deep neural network, a contextual bandit and linear regression. This section will introduce these techniques and explain the theory behind them as well as give an understanding as to why we chose to implement them into our system. This is done by presenting examples of successful uses of the three techniques in recommender systems and by highlighting their advantages and disadvantages in such systems. Both contextual bandit and linear regression are based on content-based filtering while the deep neural network utilizes both content-based filtering as well as collaborative filtering. However the collaborative filtering plays a small role even in the deep neural network and the system is mostly built around content-based filtering as previously mentioned.

2.1 Recommender Systems

A recommender systems might gather a user's history and other information such as gender, location, age and other user specific data to be able to give personalized item recommendations to each user. The items could be anything that a user could buy, use, read, watch or listen to. Recommender systems are used by numerous companies featured on the web today that aim to sell products or services, including retailers, media services and social media companies.

Recommendations can be given based on information collected via indirectly monitoring a user and its browsing behaviour, for instance personalized ads based on previously visited sites. Other more direct ways to give recommendations are, for instance, built upon a user's explicitly stated review of a product. For instance, if a user gives a movie a high rating the recommender system would recommend the sequel. Echo Nest, which is a company that works with music recommendation, looks at both analysis of the actual music as well as text analysis where they look at how artists are mentioned around the internet [40]. The recommender system in this project is built around the analysis from Echo Nest, as Spotify uses Echo Nest for their music analysis, as well as real-time data which is used to personalize the recommendations further. Web-based services and companies can increase their revenue if the recommender system works well by presenting other relevant items to the user so that the user will buy more, stay longer or in other ways keep utilizing the service provided. If recommender systems are well constructed they benefit not only the companies behind them but studies show that users also

find accurate recommender systems helpful [28].

There are two major classes of filtering items for recommendations namely collaborative filtering, which takes into account the users' histories and their similarities, and content-based filtering, which as the name implies, recommends items based on the contents of the item such at attached labels and features of the item [15].

2.1.1 Content-based Filtering

Content-based filtering draws from the idea that if a user is interested in an item, the user is also likely to be interested in similar items. As such content-based filtering uses labels and attributes to filter and group items together. Depending on the type of the item, different attributes can be extracted and items with similar attributes are grouped together. This way, only the history of the target user is needed. The content of the items in the user's history is matched with the content of the other items in the domain and the items in the domain with highest similarity to the ones in the user's history are recommended. One instance where content-based filtering is commonly used is when recommending news article, web sites or other text based items. This is because it is easy to extract words from textual data and apply labels for filtering [15].

The main challenge of content-based filtering is extracting relevant information about the items in such a way that it can provide good recommendations. New techniques to extract information are constantly being created which is makes content-based filtering a viable choice for recommender systems. As an example, Spotify has looked into extracting the frequency ranges of parts of songs and comparing them to find similar sounding songs [10]. Another problem with content-based filtering is to determine what features are relevant and how they are weighted against each other for a specific set of items or users. When recommending a movie to a user based on a previously watched movie, the genre and movie director might be more relevant than the release year or the length of the movie.

Content-based filtering does not have the same issues with new items that collaborative filtering has which will be discussed in the next section. However a downside of content-based filtering is that the system will recommend songs that are similar to what you already listen to and as such will not recommend songs that are different from those tracks. For a new user that has only listened to a few songs, the recommendations will be influenced solely on these few listenings and as such might be very predictable and unsatisfying for the user. Although if the extracted content is diverse enough and based on a lot of different parameters the system might find connections between songs that do not sound very alike to a human ear but might still attract the same listeners.

To complement user id and real-time parameters the project utilizes content-based filtering to find similar tracks. The challenge of extracting data does not apply to this project since Spotify provides a vast number of features connected to a track which will be used as the content used to find similarities between songs in our system. However, we will only be able to use three of the features, namely loudness, mode and tempo, because we do not have enough user data. It is not possible to know if these features are the optimal ones to describe a track although tempo and loudness have been shown to have a connection to heart rate [13][35]. Furthermore, tempo and mode have been shown to infer how we perceive a song as "happy" or "sad" [16]. Because of the aforementioned reasons, we made the assumption that these features are well suited for the approach of this project.

2.1.2 Collaborative Filtering

A system that uses collaborative filtering operates on a user-item matrix to find correlations between users based on their preferences regarding the items in the system. There are different ways to gather preferences, either explicitly by feedback given from the users such as "like/dislike-buttons" or numerical ratings, or implicitly gathered information like number of views, listenings or clicks. The system builds on the notion that if two

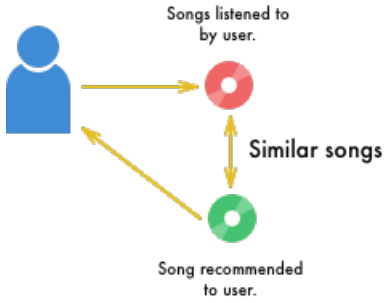


Figure 2.1: An example of how content-based filtering works.

users have similar ranking histories they are likely to have similar preferences in the future as well. For instance, if user X and user Y have ranked a lot of items similarly the connection will be high between them and when user X gives a new high rating on an item that user Y has not yet evaluated the system will recommend that item to user Y. A group of users with similar taste is usually called a neighbourhood and by observing the neighbours rankings and user history, predictions and recommendations can be made. The similarity of two users could be calculated using *Pearson's correlation coefficient* [39]:

$$\rho_{X,Y} = \frac{\sum_i (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_i (X_i - \bar{X})^2}\sqrt{\sum_i (Y_i - \bar{Y})^2}}$$

Where X_i and Y_i are the ratings of user X and user Y regarding item i and $\bar{X}$ and $\bar{Y}$ is the mean value of their ratings. A prediction whether user X will like or dislike an item is based on the weighted average of the neighbours' recommendations of this item.

Collaborative filtering methods can also be used to find correlations between different items as opposed to different users. User X gets recommended items which have a high correlation to the higher ranked items in user X's history. Amazon's patented recommender system which has been in place since 1998[24] use item-based collaborative filtering to group items which are often bought together. This way multiple items can be recommended to new customers.

A problem with collaborative filtering approaches is the so called *cold start problem* where not enough data is available for a user or an item. When a new user is added to the system it will not initially have enough ratings for the system to find sufficiently similar users and thus the accuracy of the predictions will be limited. Another example of cold start is when a new item is added to the system. The item will not have any ratings so it won't be recommended to any users. There are many ways so solve this problem, asking new user for initial ratings or recommending the most popular items while gathering more information are two approaches to overcome the new user cold start problem. An approach to solve the new item problem is to implement content-based filtering in a hybrid approach which will be discussed in the coming section. However the cold start problem for new users exists for content-based filtering as well because a new user will not have any songs from which to filter and recommend music.

As mentioned in the scope, our system does not rely on collaborative filtering as we do not have the user base to support using this way of filtering. The only one of the three implementations that uses collaborative filtering is deep neural network and even then the usage is limited. Preferably we would have used collaborative filtering together with content-based filtering and the real-time data however this proved not to be viable.

2.1.3 Hybrid and Other Approaches

Collaborative Filtering and Content-based Filtering each have their own advantages and disadvantages as previously discussed. A widely used approach is to use these two filtering approaches together in a hybrid approach

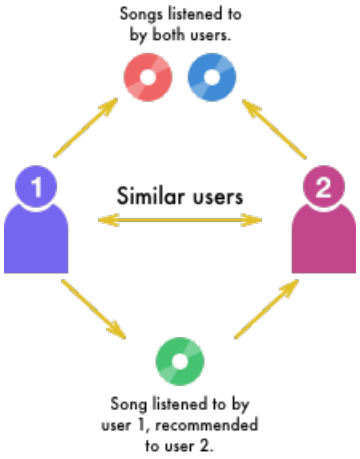


Figure 2.2: An example of how collaborative filtering works.

to gain the advantages of both approaches and mitigate their disadvantages [15]. As previously mentioned a hybrid approach mitigates the problem in Collaborative Filtering when a new item is introduced by utilizing content-based filtering for that item. An advantage over content-based filtering is that the collaborative aspects of the systems can provide a wider range of recommendations as it is not limited to the features of a specific item.

A solution to the cold start problem for a new item in collaborative filtering is to apply a hybrid recommender system that utilizes a content-based filtering approach on the new items. This way the system can group the new items together with other items with existing ratings based on other properties than user interactions and ratings. This grouping can then be used to recommend the new items to users who are likely to be interested in them.

A hybrid approach also allows for more parameters to be taken into consideration for each recommendation. Other filtering methods such as demographic filtering, which is a type of collaborative filtering that groups users in the same demographic together, can also be implemented into such a system. Our recommender system filters both content-based as well as using real-time data and therefore would qualify as a hybrid approach. As previously mentioned we would have liked to use a collaborative approach as well to try to mitigate some of the mentioned issues with content-based filtering. Using real-time data does not eliminate the problems with content-based filtering but is intended to refine the recommendations further by personalizing them based on the current state of the user.

2.2 Machine learning

In this section the three implementations of our feature recommendation system will be introduced and discussed. The theory behind the implementations as well as why we chose to implement those specific machine learning techniques will be covered.

2.2.1 Deep Neural Networks

A neural network is a set of algorithms, designed to find numerical patterns in large sets of data. The patterns are used to classify and cluster the data which in turn can be used to classify new incoming data based on patterns from previous data. The network is trained by considering large amount of data, called training data. The more data you can feed into a neural network, the more accurate that network will become. A neural network consists of an input layer, an output layer as well as one or more hidden layers. The hidden layers are layers of nodes,

called artificial neurons, where the computations are made. The programmer does not interact with these layers which is why they are explicitly called hidden. Instead the neural network tweak the nodes and connections between said nodes based on training and it is this principle that allows the network to learn on its own. A deep neural network [17] is a neural network with more than three total layers meaning that it has at least two hidden layers. See figure 2.3 for an overview of the layers in a deep neural network.

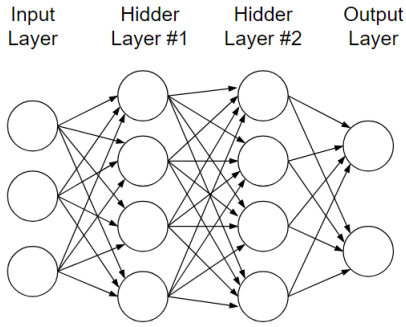


Figure 2.3: A deep neural network with two hidden layers

The nodes perform all computations in a neural network. They receive input either directly from the data set, if the node is in the input layer, or from other nodes in the network. The inputs are then weighted and the product of all inputs and their corresponding weights are summed. This sum is then used in an activation function that is what ultimately decides how much the node will be activated. There are several different types of activation functions and which one that is the most optimal depends on what kind of data you have, but almost all of them are non-linear. The output from the node is then either passed to another layer of nodes or used as part of the output from the neural network if the node is in the output layer. See figure 2.4 for an overview of how a node works.

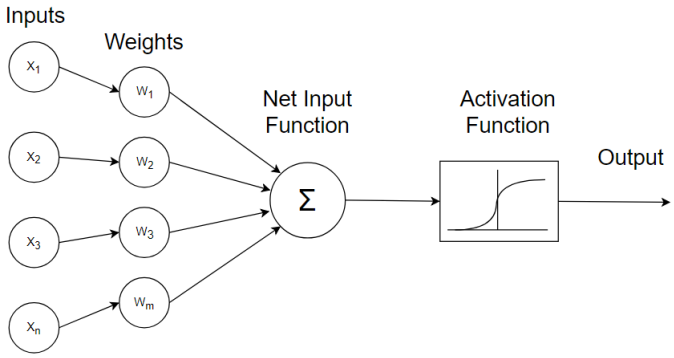


Figure 2.4: A diagram over a Neural Network Node

A neural network trained on labeled data, which is called supervised learning, can be used to classify unlabeled data. As an example, a neural network trained on a data set with pictures of human faces, labeled with either 'smiling' or 'not smiling', could be used to give an accurate prediction on whether an unlabeled picture of a human face is smiling or not. Supervised learning works by giving the network an input and asking for a prediction. The labels are then used to determine whether the network was correct or not. If the prediction was correct the weights that allowed the network to make a correct guess are increased and if the prediction was incorrect the weights are lowered to make that guess less likely.

Neural networks can also be trained on unlabeled data, also known as unsupervised learning. Since the neural network has no labels to work with, it clusters the data instead of classifying it. This can then effectively be used to detect similarities or anomalies in large data sets. Using the same example as above where an unsupervised neural network is trained on a data set of human faces, this time with no labels. The neural network would then create multiple clusters for the faces, each face belonging to several clusters at the same time. By inspecting those clusters you could possibly identify labels by inspecting at the data in each group. Groups could be anything from 'smiling' or 'brown-haired' to things much harder to classify for a human researcher. The advantage of supervised learning over unsupervised learning is that the training for the network is predictable. The network will be able to classify exactly what you trained it to classify as opposed to unsupervised learning which makes its own groupings that can be unpredictable. However the largest downside of supervised learning is that it requires a large amount of labeled data which is often hard to get and time-consuming to make as it requires a person to look through the data and add appropriate labels.

Due to their ability to classify and make predictions as well as clustering items, deep neural networks are becoming increasingly popular to incorporate into recommender systems [33]. For instance, the recommendation service on YouTube has improved since utilizing neural networks, which consider both user history and content of the videos to give personal recommendations [27]. In music recommendations deep neural networks have been successfully used to cluster songs together due to audio features[1] to improve content-based recommendations as well as incorporate music content in collaborative filtering approaches to avoid the cold-start problem for new tracks [9][14].

There are several different way to implement a neural network such as multilayer perceptron, autoencoder, convolutional neural network, recurrent neural network and several more [17]. How each of these implementations work will not be discussed in this paper. Each of these implementations are viable implementations to be used within a recommender system, there is not one single method to use for all problems as each have their own advantages and disadvantages [33]. From our research, we could not find anything indicating that a specific network would perform better for our problem. Therefore a basic feed forward deep neural network was chosen for evaluation.

In our project, we have evaluated DNNs trained on labeled data. The networks' features are the user data, and its labels are the values of a song feature such as tempo, loudness or mode. The intention is that with enough data our DNNs will be able to accurately predict what tempo or mode a user prefers at a specific heart rate and time of day.

2.2.2 Reinforcement Learning and Contextual Bandits

Reinforcement learning (RL) is a type of machine learning in which an agent is trained to maximize a reward by observing the environment, its state and then taking some kind of action. This can be modelled as a Markov decision process (MDP) where we have a set of agents and environment states called S, a set of possible actions A, $P_a(x,y)$ which is the probability that you go from state x to state y with action a and $R_a(x,y)$ which is the immediate reward gained from going from state x to state y with action a.

The goal of a RL agent is to maximize the cumulative reward for a set of MDPs in which the reward is not known before the action is taken. A RL agent is therefore using a combination of both exploratory random actions combined with learned and estimated reward probabilities. The balance between these random exploratory actions and using the current knowledge is the key to having a high performing RL agent. The exploratory random actions help prevent the agent from getting stuck on a single action if said action gives a good reward. For instance, if the agent takes an action and a good reward is given it might not try another action in the action space that would give an even better reward. This is what makes RL agents well-suited for recommender systems [4], their exploratory actions enables the agent to explore.

RL differs from supervised learning in that it takes a reward which can be a number in a range. In supervised learning a prediction is either correct or incorrect while in RL the reward is a number signifying how good the prediction was. One type of RL is the multi-armed bandit where the goal is for the network to learn which of the arms give the best reward or payout. The multi-armed bandit problem can be explained as k one-armed bandit slot machines that all have a different hidden chance of winning. The task is to find the slot machine that has the highest chance of winning. A RL agent is a viable choice for solving such a problem by testing each of the arms in a random manner and then trying the arm that has given the best reward. However this problem does not take the environment into consideration and instead focuses solely on maximizing the profit of a static set of rewards. Real world problems rarely have a static set of rewards and the environment or context plays a role in what the reward of an action is. An example of an agent that takes actions based on the environment is the *contextual bandit*, most commonly used to recommend news and articles on websites [23]. A contextual bandit problem can be stated as: we have k one-armed bandits with a reward chance that changes depending on the time, these one-armed bandits represent the actions we can take and the time represents the context and state of our environment. From reading the state the bandit can then make predictions for the most suitable action to take for that specific state i.e. which bandit the agent should pull based on the rewards received for previous actions taken in the same context. This is accomplished similarly to the multi-armed bandit problem by testing actions for each state and then learning which action gives the highest average cumulative reward for each state.

The math behind a contextual bandit is as follow:
A contextual bandit runs $t = 1, 2, ..., T$ rounds, the context for our environment is $x_t \in X$. Based on the context the bandit selects an action $a_t \in A$ and the environment returns with a reward $r_t \in R$.
The goal of the bandit is to maximize the cumulative reward $\sum_{t=1}^{T} r_t$. To accomplish that the bandit has a set of policies $\Pi \subseteq \{X \to A\}$ and attempts to find the policy $\pi \in \Pi$ that grants a the largest reward.

Figure 2.5: Contextual bandit process

We decided to implement a contextual bandit as the contextual bandit problem is similar to the one in our project. The problem involves that we have a state which is determined by the user's current heart rate and the time of the day, and we want the system to determine the best audio features for this state. Another advantage of the contextual bandit is that it mitigate the issue of using content-based filtering to some extent, as it is capable of taking random action which increases it chances of finding an optimal set of features.

2.2.3 Linear Regression

Linear regression is used to model the relationship between an independent variable x, and one or more dependent variables y. This is done by fitting a straight line to observed data in a way that minimizes the errors, as can be seen in figure 2.6. For instance if we want to find the relationship between a person's height and weight we can consider observed data and plot the weights depending on the heights and find a linear equation with the aim that the distance from the data points to the line is as small as possible. The standard approach to find the best fitted line $\hat{y} = b_0 + b_1 x$ is the least square method which minimizes the sum of the squared offsets from the data points to the line. To use linear regression for predictions the fitted line is used to look up corresponding $\hat{y}$-values to an input value where no previous data exists. In recommender systems linear regression models have been used to find relationships between different users in collaborative filtering approaches and use these relationships to efficiently make predictions [5] [37]. We chose to implement a linear regression algorithm into

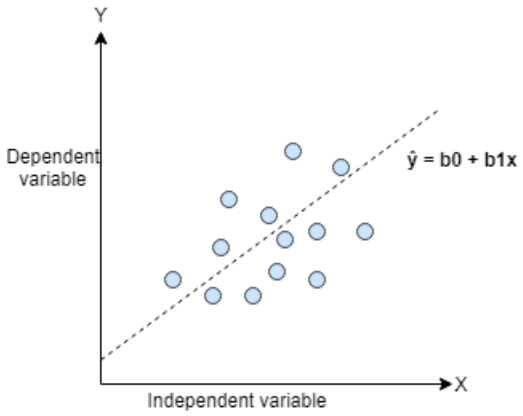


Figure 2.6: Example of linear regression

our recommender system because it is widely used and it would be interesting to compare a linear approach to the other non-linear ones.

2.3 Evaluation Methods

There are many approaches to evaluate recommender systems such as user studies, offline evaluation and online evaluation. In this section the two evaluation methods online and offline evaluation will be described, including their advantages and disadvantages, as we chose to attempt conducting these method to evaluate our system. However due to the low amount of user data accessible in this project online evaluation did not prove viable, which will be discussed in the section about online evaluation.

2.3.1 Offline Evaluation

An offline evaluation is conducted by testing a system on a number of simulated or manually created users. By simulating a user you are able to evaluate your system on a specific hypothesis. To create and simulate real user behaviour the offline user is often created on data gathered from real users. Thus the evaluation relies heavily on data gathered from real users to be able to simulate a behaviour as close to real user behaviour as possible [41]. Offline evaluation is cheap and easy to conduct since it doesn't require real users and is thus often conducted in the earlier stages of development. Offline evaluation of recommender systems can for instance be conducted in an early stage to compare two algorithms and find the most appropriate one. [31].

When conducting an offline evaluation there are three common steps [31]:

Hypothesis:
Similar to other evaluation systems it is important to create a clear hypothesis that presents the test with a clear purpose in the first step. This could be comparing algorithm A and B to determine which algorithm gives the best recommendations.

Controlling Variables:
It is important that only the variables being evaluated are changed and all other variables remain static. So for our previous example of comparing two algorithms, if the algorithms were trained on different data it would be difficult to prove that the result was not affected by the data the network was trained on.

Generalization Power:

Because we evaluate our system in a closed environment we are only answering a narrow set of questions and thus it is important to make various evaluations to cover all possible scenarios.

A disadvantage of offline evaluation is, as mentioned, that it can only answer a narrow and predefined set of questions. Another disadvantage is that it is not possible to ask a specific question and monitor why a user chooses to make a specific choice or give a certain answer, because a simulated user can not be asked about its intention or reasons for making a specific choice. The advantage of this evaluation system is that it is much cheaper and faster than other evaluation methods [31].

2.3.2 Online Evaluation

A common method for measuring a recommender system's impact on user behaviour is through using a method called *online evaluation*. When conducting an online evaluation the user is put in contact with a functioning version of the recommender system to evaluate, and can therefore receive actual recommendations from the system. During the experiment, the user's actions are recorded and processed in order to evaluate the system. According to some researchers the only true way of measuring user satisfaction is through online evaluation [41][7].

User behaviour can sometimes be hard to measure, as the effect of a recommender system is dependant on plenty of different variables, such as the user's intent (i.e. what their needs are), the user's context (i.e. what their past experiences look like), and the looks of the system (i.e. the user interface) [32]. For this reason, when you perform an online evaluation of a system you often run multiple versions of the same systems in order to see how the different systems perform compared to each others, as it can be hard to interpret numbers alone. This type of evaluation method is often referred to as A/B-testing [32]. When conducting an A/B-test it is of importance that the users are assigned at random to a recommender system to test as you otherwise risk getting a biased result [32].

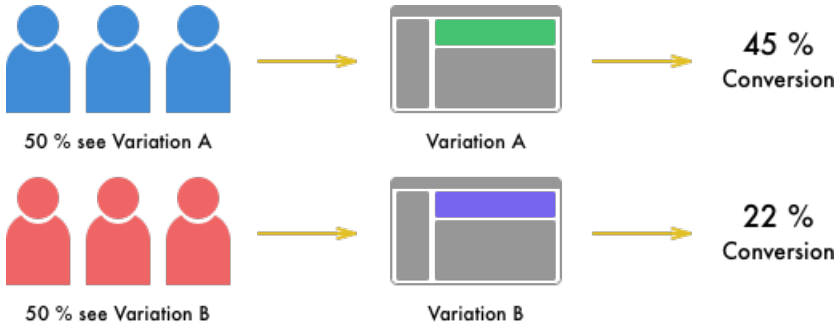


Figure 2.7: Example of an A/B-test

Advantages of online evaluation in comparison to other evaluations methods include that it is capable of directly measuring overall system goals, such as user retention or click-through-rate. This allows the person conducting the experiment to understand how system goals are affected by properties like the accuracy and diversity of recommendations. On the other hand, a disadvantage of using online evaluation is that it can be hard to gain a complete understanding of these relationships as properties are varying independently. Online evaluations are also relatively expensive in contrast to other methods. [32]

In this project online evaluation was supposed to be used in order to evaluate how well the system recommends songs to the users, but due to a lack of data a proper online evaluation could not be conducted even though it would have been the best method for evaluating the system.

Chapter 3

Implementation

In this chapter the implementation of our entire system is described. Each subsystem and all components is described in detail as well as a system overview which explains the main flow and how all parts of the system is connected.

3.1 System Overview

The system consists of a server and one or more clients connected over internet. The server keeps a database of user data and server statistics, a web server that manages client communication, and the recommender system that is used to give song recommendations to the connected clients. The recommender system contains two significant parts: a set of feature recommenders and a ranking system. A feature recommender returns a value for a specific audio feature based on user data. While the ranking system recommends a set of songs based on these features. Each client in the system is a mobile application that is used to play the recommended music and measure the user's heart rate. In figure 3.1 an overview of the system and the data flow directions can be found.

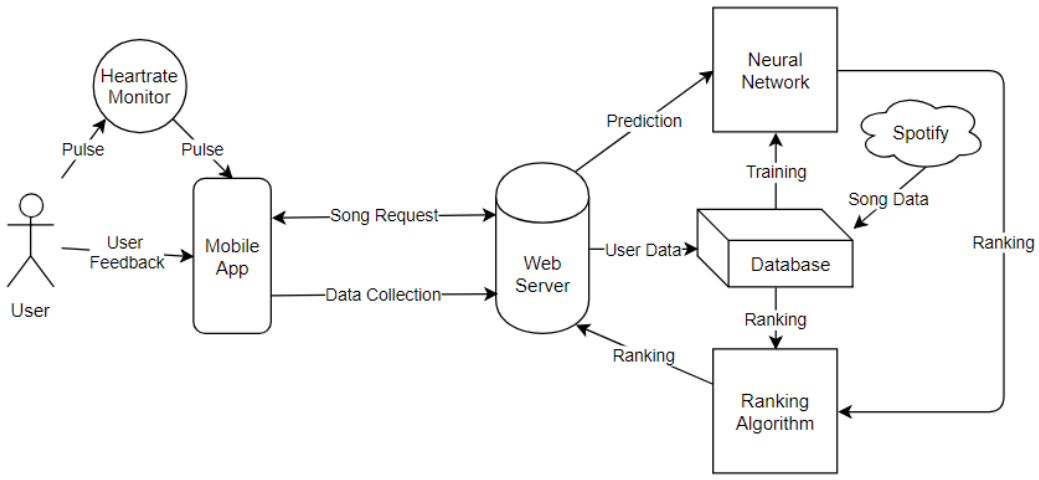


Figure 3.1: An overview of the different parts in the recommender system

3.2 Data

We have two large data sets that our recommender system uses as its base for its recommendations. The first is a set of songs, each song is paired with its respective tempo, mode and loudness, i.e. the features that we selected as variables for our content-based filtering. At the initial stage of the project, a set of 1124 songs was selected. The songs were chosen from a variety of different playlists to get a broad mix of artists and genres in order to mimic the Spotify library, although in a much more compact form. Spotify uses a unique song id to identify each song. We also use this id to represent the songs in our system. The audio features for all of these songs was downloaded from Spotify on the 17th April 2018 using their developer API [34] and saved to the database on our server.

- **Tempo** - The average estimated tempo of a song measured in beats per minute.
- **Mode** - The mode of a song which can be major or minor, represented by 1 and 0 internally.
- **Loudness** - The loudness of a song measured in decibels averaged over the entire song, ranges between -60 and 0.

The second data set is our user data. We have 233 data points from two real life users, that has been gathered over several months. There is also three automatically generated data sets with 500 data points each that has been used to evaluate the different recommender systems. Each data point represents a song being played or skipped. They are created when a user has finished listening to a song and consists of the variables below.

- **User id** - A unique user name
- **Heart rate** - Average heart rate for the user during the song
- **Time value** - The number of minutes from midnight. Used to identify what part of the day it was when the song was listened to.
- **Song id** - A unique song identifier
- **Rating** - The time the user listened to that song in percentage of that songs total time. Full skip is a '0', half the song is '0.5' and the entire song is '1.0'

3.3 Mobile Application

The mobile application is the part of the system which the user interacts with. The application mainly has two functions: gathering information from the user and to act as a music player. In order to achieve this the application has been divided into three parts: data collection, server communication, and music playing.

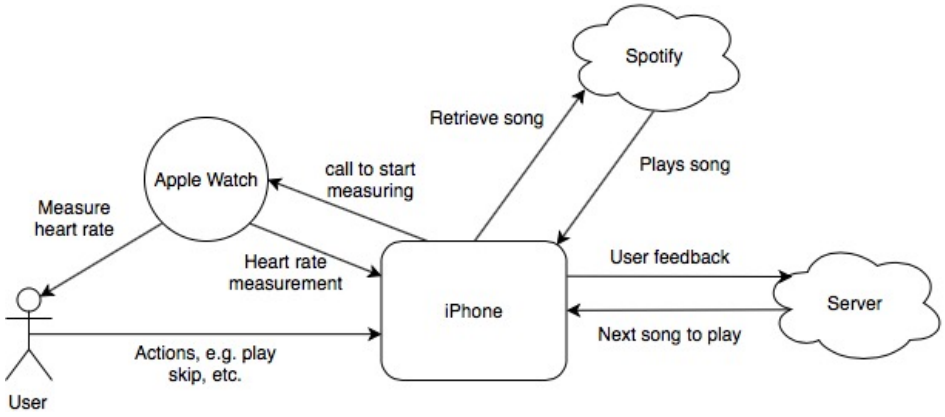


Figure 3.2: Flow chart of the mobile application

The application gathers both direct and indirect feedback from the user. Direct feedback comes from user actions such as starting a session or skipping a song. Indirect feedback on the other hand comes from active measuring the user's behaviour in the application, which includes continuous measuring of the user's heart rate and the time the user listens to each song. In order to measure the heart rate of the user the mobile application is in need of a smart watch and a corresponding application for the watch. In the current implementation the mobile application is built for iOS 11 (the latest version of the iPhone- and iPad operative system) and the type of smart watch used is an Apple Watch with a watchOS-application (built for watchOS version 4) installed.

There are two scenarios where the mobile application and the server communicate. The first scenario is when the mobile application requests a song recommendation from the server in order to play it to the user. This scenario plays out every time the user starts a new listening session and when the currently playing song has ended. The second scenario is when the mobile application sends feedback on a song to the server, this happens when has song has ended, either because the song has played through, the user decided to skip the song, or the user stopped playing music all together.

Measuring the heart rate continuously isn't a pre-built feature for iOS, so in order to accomplish this the iOS-application needs to implement a few things. First, the application needs to establish means of communication between the phone and the watch, this can be achieved by using the in-built messaging service for iPhone and Apple Watch, which enables us to send simple messages between the two devices. Using the service we then send a message to the watch to start measuring the heart rate of the user. The actual measuring of the heart rate is accomplished by starting a *exercise* on the watch, which starts a continuous measurement of the user's heart rate. Then every time we receive a new measurement (approx. every 5 sec) we send back the measurement to the phone, using the messaging service, which process this in order to send it back as feedback to the server.

The music player part of the application is heavily dependant on Spotify's iOS framework. The framework is used both to retrieve the songs and their data, as well as for playing the songs. On top of the framework we have built extra functionality to be able to gather information and in order to build a user interface where the user can control the music player.

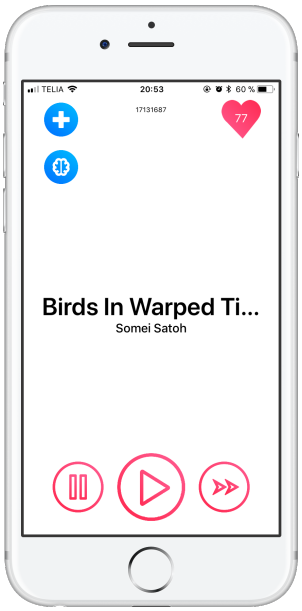


Figure 3.3: A screenshot of the mobile application's user interface.

	CPU	RAM	HDD
Server	i5-4570S @ 2.90GHz	16 GB DDR3 1600 MHz	1TB 64MB Cache SATA 6.0Gb/s

Table 3.1: The hardware specification for the system server

3.4 Web Server

The web server's task is to handle client requests, forwarding them to any relevant back-end system and sending a response back to the client once it's been calculated. The server is built with the web framework *Django* version 2.0.2 [11] with Python 3.5.2 and it runs on the hardware found in table 3.1.

There are two types of client requests that the web server is configured to accept. Data collection and song recommendations. A data collection request is initiated from a client when a song has finished or has been stopped. It is sent as a http POST request with a JSON-encoded data package containing user data as described in section 3.2. When the server receives the request, the packaged data is parsed with a serializer and saved to our database if the contained data is valid. The client then receives a standard http 201 response if the data was valid, otherwise a http 400 response is sent to inform the client that no data has been saved.

A song recommendation request is initiated from a client when a new song needs to be played. The request comes in the format of a http GET request with a JSON-encoded data package containing the user id, heart rate and time value.

Once the data is deemed valid the web server then:

1. Checks if a new trained feature recommender is available
   (a) If that is the case it then switches to most recently trained model
2. Checks if the song cache for that user is empty
   (a) If the cache is empty, it asks the feature recommenders for the recommended song features
   (b) Then use the features to populate the cache with the top songs from our ranking system that match

those features, adjusted for user history

3. Pop a song from the users cache and returns it to the client.

A sequence graph for how a song request is handled with an empty cache and no new trained model can be found in figure 3.4.

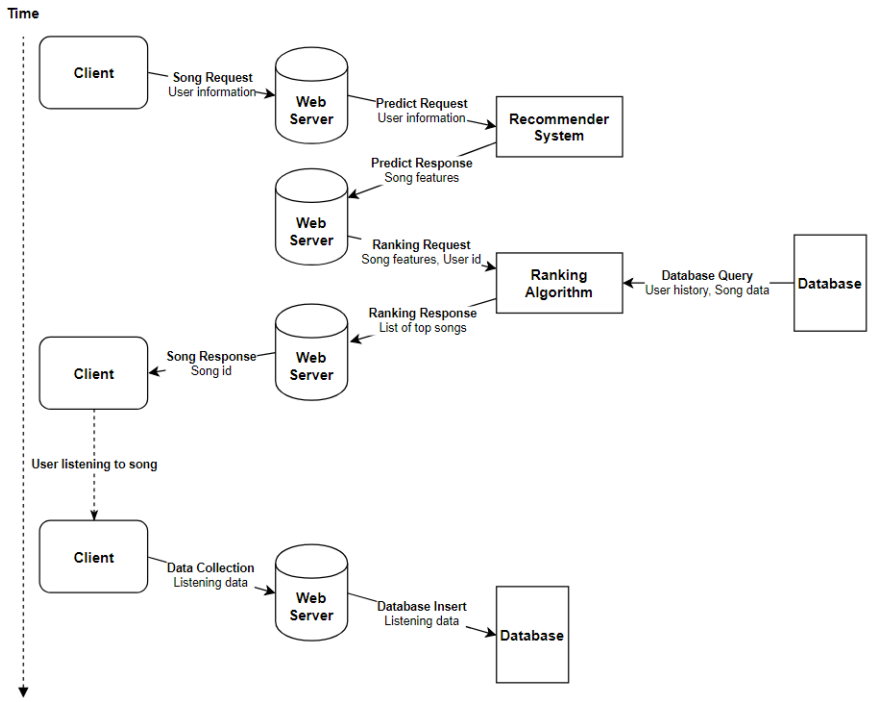


Figure 3.4: A sequence graph for a song recommendation with an empty server cache

3.5 Database

All data that requires persistency is saved in a postgreSQL 9.5.12 [29] database that runs on the same server as the web server and the recommender system.

The database has tables for:

- *User listening data* - User data as described in section 3.2. Used by feature recommenders for training and by the ranking system to calculate user bias.
- *Song audio analysis data* - Song feature data as described in section 3.2. Used by the ranking system.
- *User information* - How many songs a user has played and a unique index for each user. Used by the web server and the ranking system.
- *Django* - Data required by Django. Used by the web server.
- *Last played* - Table to track the last time a user played a specific song. Used by the ranking system.
- *Action ids* - Table to track the actions taken by the contextual bandit so they can be correctly paired with the corresponding reward at a later time.

3.6 Recommender System

The recommender system consists of two subsystems where the output from the first system becomes the second systems input. The first subsystem is our three feature recommenders. Each feature recommender takes user information as the input and returns with the recommended song feature that it's been configured for. We have three different sets of feature recommenders, each based on a different machine learning technique and implemented with Tensorflow which is a machine learning library developed by Google. All three has been evaluated and trained on the same set of data. Our second subsystem is our ranking system. The ranking system takes the output from our feature recommenders and combine it with the users song ratings and recent listens to rank all our songs in our database based on that input. The top ranking songs are returned to the web server.

3.6.1 Deep Neural Network

The deep neural network (DNN) that was evaluted is a set of three almost identically configured DNNs, each trained on the same user data but with different labels. The model was implemented using a *DNNClassifier estimator* in Tensorflow and configured to use a gradient descent optimizer with a static learning rate of 0.01. The feature columns are identical for all three DNNs and can be found in table 3.2. The label is either mode, tempo or loudness for the song that was played. Tempo and loudness are bucketized in to the buckets in table 3.3 while mode was represented by either 0 or 1e. Two hidden layers, the first with 5 neurons and the second with 1 neuron, was used for all three DNNs, which was an estimated middle ground based on the formula 3.1.

When a song recommendation request is sent to the web server, only data for user id, heart rate and time value are sent with that request. To generate the predictions, the web server then bundles the user data with rating 1.0 and sends it to our trained models. In order to generate song features that are paired with as high rating as possible. Once predictions for each of our three song features has been calculated, they are returned to our web server that forwards them to our ranking system.

Training is done either manually by calling our training script or automatically through a scheduled function which is configured to run the training every 2 minutes on the server. The training script extracts all user data 3.2 from our database. The song id is used to lookup the mode, tempo and loudness for each entry from our song data table. All that data is then organized in a format that our DNN model could read and the user features was sent together with the respective labels to each DNN for training. After the training the models are saved to a checkpoint, so that the web server can use them to instantiate new trained models to generate predictions from.

To determine the size of the hidden layers we used the following equation as a rule of thumb. It is mainly a recommendation as there is no single way of determining the optimal size of hidden layers.

$$Nh = \frac{Ns}{(\alpha \cdot (Ni + No))} \tag{3.1}$$

Nh = upper bound of hidden neurons
Ni = number of input neurons.
No = number of output neurons.
Ns = number of samples in training data set.
α = an arbitrary scaling factor usually 2-10.

Name	Type
User id	Categorical
Heart rate	Bucketized numerical
Time value	Bucketized numerical
Rating	Bucketized numerical

Table 3.2: Feature column types used for our deep neural network and linear regression models

3.6.2 Linear Regression

A linear regression (LR) model was implemented with Tensorflow as a *LinearClassifier estimator* which is then instantiated once for every song feature in our scope 1.4. The feature columns are identical for all three LR objects and can be found in table 3.2. The label is either mode, tempo or loudness for the song that was played. Tempo and loudness are sorted in to the buckets in table 3.3, while mode only can take a value of 0 or 1.

Predictions are made in a similar way as with our DNN model. The server receives user data, bundles that data with a rating of 1.0 and sends that to every LR object to get the predicted song features. To train the LR model we use the same method as mentioned in the previous section about DNN due to the similarity in implementation of the two.

3.6.3 Contextual Bandit

The contextual bandit setup that was evaluated was a set of three contextual bandits for each user. Each bandit was then used to predict an action that corresponds to a specific value for the song feature that the contextual bandit was configured for.

The set of actions that was used by each bandit was the same as the buckets used by our DNN and LR configuration 3.3. As an example, action 6 for the tempo bandit corresponded to a tempo of 131 to 150 BPM. The set of states is the same for each bandit and each state a unique integer that matched to the unique combination of bucketized heart rate and time value 3.4. The reward received is calculated from the rating of each song with a basic reward formula:

$$y(x) = \begin{cases} 1 & \text{if rating } \geq 0.8 \\ -1 & \text{else} \end{cases}$$

Each contextual bandit was configured as a feed-forward neural agent with a gradient descent optimizer. A prediction is done by either by selecting an exploratory random action or the action with the currently maximum weight for the state received as input. The ratio of selected actions between the exploratory and value maximizing actions was set to 1:9. Once an action had been chosen, an action id was created and sent with the request. The id, chosen action and state is saved to a lookup table and then used in training to backtrack the rating received to the bandits choices. Training was done either manually by calling our training script or automatically through a scheduler that was configured to run the training every 2 minutes on the server. The training script extracted all user data 3.2 that was not already flagged as data has been trained on from our database. The action id's for all the extracted data was then used to lookup the chosen action and state for each rating. All that data was then organized in a format that our DNN model could read and the action, state and rating was then sent to the model for training. The models was then saved to a checkpoint after the training. The contextual bandits can also be trained on data that has no action id attached, generated by the DNN or linear regression models. To train the bandit on that data, the state was directly calculated from the heart rate and time value in the data point. The song id was then used to lookup the song features and the actions for each bandit was calculated by bucketizing all the features. The bandits can then be trained as normal with tuples of action, state and rating.

Label/Action	Tempo (BPM)	Loudness(db)
Bucket 0	$0 \leftrightarrow 30$	$-60 \leftrightarrow -20$
Bucket 1	$31 \leftrightarrow 50$	$-19 \leftrightarrow -18$
Bucket 2	$51 \leftrightarrow 70$	$-17 \leftrightarrow -16$
Bucket 3	$71 \leftrightarrow 90$	$-15 \leftrightarrow -14$
Bucket 4	$91 \leftrightarrow 110$	$-13 \leftrightarrow -12$
Bucket 5	$111 \leftrightarrow 130$	$-11 \leftrightarrow -10$
Bucket 6	$131 \leftrightarrow 150$	$-9 \leftrightarrow -8$
Bucket 7	$151 \leftrightarrow 170$	$-7 \leftrightarrow -6$
Bucket 8	$171 \leftrightarrow 190$	$-5 \leftrightarrow -4$
Bucket 9	$191 \leftrightarrow \infty$	$-3 \leftrightarrow -2$
Bucket 10	-	$-1 \leftrightarrow 0$

Table 3.3: Label and action buckets for tempo and loudness

Bucket	Heart rate	Time value	Rating
Bucket 0	$0 \leftrightarrow 40$	$300 < t < 660$	$0 \leftrightarrow 0.2$
Bucket 1	$41 \leftrightarrow 60$	$659 < t < 960$	$0.21 \leftrightarrow 0.4$
Bucket 2	$61 \leftrightarrow 80$	$959 < t < 1320$	$0.41 \leftrightarrow 0.6$
Bucket 3	$81 \leftrightarrow 100$	$1319 < t$ *or* $t < 300$	$0.61 \leftrightarrow 0.8$
Bucket 4	$101 \leftrightarrow 120$	-	$0.81 \leftrightarrow 1.0$
Bucket 5	$121 \leftrightarrow 150$	-	-
Bucket 6	$151 \leftrightarrow 180$	-	-
Bucket 7	$181 \leftrightarrow \infty$	-	-

Table 3.4: Heart rate, time value and rating buckets

3.6.4 Linear Regression

A linear regression (LR) model was implemented with Tensorflow as a *LinearClassifier estimator* which is then instantiated once for every song feature in our scope 1.4. The feature columns are identical for all three LR objects and can be found in table 3.2. The label is either mode, tempo or loudness for the song that was played. Tempo and loudness are sorted in to the buckets in table 3.3, while mode only can take a value of 0 or 1.

Predictions are made in a similar way as with our DNN model. The server receives user data, bundles that data with a rating of 1.0 and sends that to every LR object to get the predicted song features. To train the LR model we use the same method as mentioned in the previous section about DNN due to the similarity in implementation of the two.

3.6.5 Ranking System

In order to complete the recommender system, a ranking system is introduced with the task of sorting all songs in our database. This sorting is based on the recommended song features from the feature recommenders together with the user id for the user making the request. With these as inputs, the algorithm then proceeds to iterate through all songs in the database, placing all of them in a dictionary and giving each song a weight value between 0 and 1. The weight values are calculated based on how well certain features of the songs correspond to the desired feature values output by the recommender system. Each feature gets a value between 0 and 1, and the total weight is then calculated by adding all the feature weights together and dividing them by the number of features used. The individual feature weights are calculated in different ways. Since mode only can assume a value of either 0 or 1 (minor or major), the mode weight can also only be either 0 or 1, depending on if the mode is correct or not. Loudness and BPM can assume many different values and we want the weight for them to be

higher based on how close to the desired value they are. This is done using the following quadratic equation:

$$y(x) = \begin{cases} -0.01x^2 + 1 & \text{if } -10 \leq x \leq 10 \\ 0 & \text{else} \end{cases}$$

Where y represents the feature weight and x represents how close to the desired value the songs feature value is. This means that the feature weight for these features will be 0 if the feature values of a song is more than 10 BPM or decibel away from the wanted feature value. Figure 3.5 shows a plot of this function.

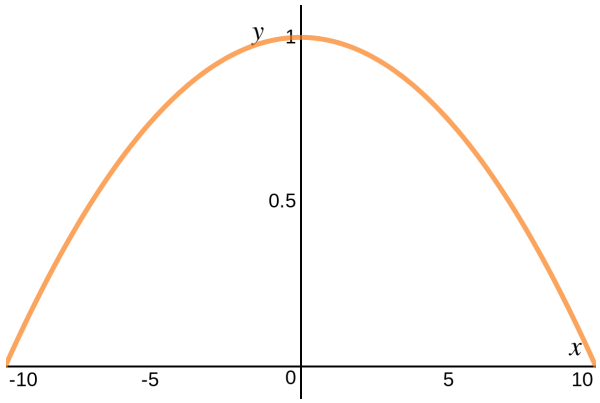


Figure 3.5: The feature weight function for loudness and BPM plotted in a graph

The weighting algorithm also takes into account how many other songs that has played since the song that is currently being weighted was last played. If the song was recently played this is converted to a variable close to 0. The variable then increases linearly until it reaches 1 when it was more than 40 songs since last played, after that it stays at 1 until it is played again. The total weight of all other parts of the song is then multiplied with this variable to get the final song weight.

$$\text{Weight multiplier} = \begin{cases} \text{Number of songs since last played}/40 & \text{if Number of songs since last played} < 40 \\ 1 & \text{else} \end{cases}$$

To get the songs weights adjusted for individual preference, a user bias is set by reading all times a specific song has been played by the current user and calculating an average rating. For songs that have never been played by the user, user bias is exempted and the song is only weighted on how similar it is to the requested song features.

$$\text{Song weight} = \frac{\text{Loudness weight} + \text{Mode weight} + \text{Tempo weight} + \text{User bias}}{\text{Number of Features}} * \text{Weight multiplier}$$

If the song has never been played previously by the user, the number of features is set to 3 and user bias is set to 0. Otherwise the number of features is set to 4 and user bias is included. This is because no user bias exists before the song has been played and to avoid that this non-existent bias influences the weight we omit the user bias. Once all songs have been weighted the ranking system finishes by sorting them all and returning a list of the 10 song id:s with the highest weights.

Chapter 4

Results

4.1 Evaluation System

To conduct an evaluation on our networks we must train them first with large amount of user data. Since we do not obtain that amount of data and are not able to produce it by ourselves, we chose to make an offline evaluation. As described in section 2.3.1 an offline evaluation should be conducted on real user data and one should avoid a biased evaluation. However due to our scarce user data we choose to a simulate a user with very specific music preferences. In our first evaluation we created a user that listens to a song with specific attributes at five different times' at day. The attributes used for this user can be seen in table 4.1.

Our hypothesis for this test was to examine if the network would recognize the user's music preferences during different times of the day, hence the very specific music preferences to be able to measure the correctness of the recommendation. So we trained the network on this user 500 and then plot the satisfaction of each recommendation made by the network.

To evaluate how satisfying the recommendations made by the network are we created an algorithm that generates a "score" for each recommendation based on the preferences of the user. Since our user exclusively listens to one song at a specific time with precise attributes we know what we can expect from the network depending on what time it is. We compare the expected attributes with the recommended songs' attributes and divide it by 3 to obtain the mean. However since each attribute have different ranges, e.g. a songs tempo could be between 40 - 210 BPM, we divide each attribute with its range. This results in each attribute being less than or equal to 1 and thus each attribute affects the score equally. However since the attribute Mode can either be 0 or 1 it affects the score greater than the other attributes, we see this fit because of mode's low range it would be a considerable error from the network to get it wrong. Thus we see fit that it would decrease or increase the score greater than the other attributes. Another complication was that the difference in range between tempo and loudness(tempo range: 0 - -60 , BPM range : 0 - 270) was too great. This would mean that tempo would affect the outcome of the algorithm more than tempo, however because we have "bucketized", i.e split the range of an attribute into

Time	Heart rate	Tempo(BPM)	Loudness(db)	Mode
07:00	60	126	-27.5	0
13:00	90	193	-4.9	0
15:30	120	101	-25.8	1
18:00	150	126	-5.8	1
22:00	55	151	-15	0

Table 4.1: Wanted song attributes by user during different times and heart rates

10 intervals, there is an equal chance for the tempo as loudness to result in half of its range.

$$
\begin{aligned}
a_1 &= \text{wanted tempo} & a_2 &= \text{recommended tempo} \\
b_1 &= \text{wanted loudness} & b_2 &= \text{recommended loudness} \\
c_1 &= \text{wanted mode} & c_2 &= \text{recommended mode} \\
x &= 270 = \text{range of tempo} & y &= -60 = \text{range of loudness}
\end{aligned}
$$

$$
z = \left| \frac{x - (a_1 - a_2)}{x} \right| + \left| \frac{y - (b_1 - b_2)}{y} \right| + \left| \frac{1 - (c_1 - c_2)}{1} \right| \tag{4.1}
$$

$$
f(z) = \text{Score} = \frac{z}{3} \tag{4.2}
$$

The contextual bandit will be trained using a clean database, i.e. when there is no entries in the user data tables. While the LR- and DNN-model need to have at least one entry in the user data table in order to fulfill their first training. As we evaluated the contextual bandit first we decided to keep its first entry for the other models, which is the reason to that all the graphs start off the same.

4.2 Deep Neural Network

Graph 4.1 represent the score of each recommendation made by the deep neural network. The Y-axis represents the score where 1 is the highest score a recommendation could receive and the X-axis represents the number of iterations. The graph also highlights the trend of every 10th score. The table 4.2 represents a sample of the recommended songs by the neural network.

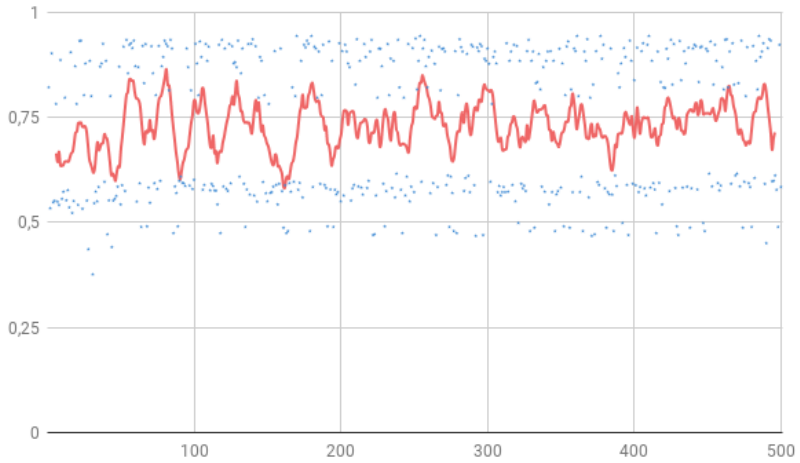


Figure 4.1: The blue dots represent the score of a recommendation at each iteration and the red line highlights the trend of every 10th score by the deep neural network.

Due to the large amount of songs recommended by the network we chose to highlight a sample of the recommended songs by the network. The order of the table does not have any intent. Score for each song is based on what time they are played and "Times played" displays how many times overall the song was played at a specific time of the day.

Time	Name	Tempo(bpm)	Loudness(db)	Mode	Score	Times played
07:00	Warren G - Regulate	95	-13	0	0.87	5
13:00	2Pac - I Get Around	96	-13.9	0	0.79	6
13:00	Bruce Springsteen - I'm on Fire	88	-14.5	0	0.78	4
15:30	Shallo - Lie	108	-11.6	0	0.57	5
15:30	Johann Bach - Violin Concerto BWV 1042	99	-14	1	0.93	7
18:00	Dan Hartman - I can dream about you	113	-14	1	0.93	9
22:00	Diamond D - Day One	91.5	-11	0	0.88	14

Table 4.2: A sample of the recommended songs by deep neural network.

4.3 Contextual Bandit

Similar to Graph 4.1 the graph below represents the score for each recommendation made by the contextual bandit network and the trend line for every 10th score. The table 4.3 represents a sample of the recommended songs by the contextual bandit.

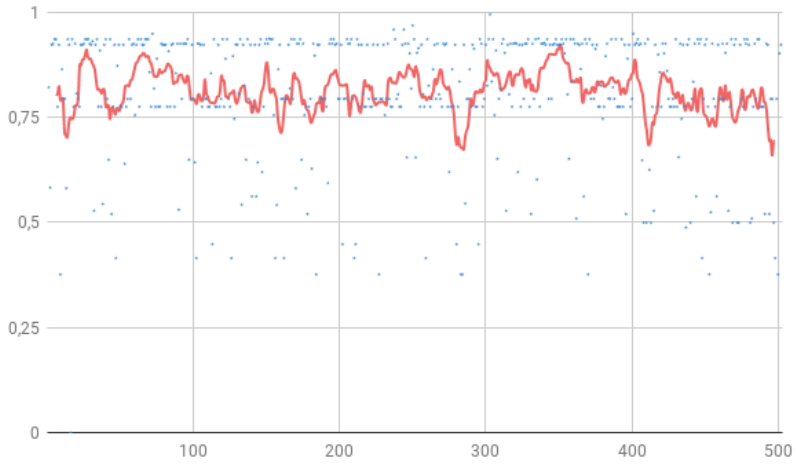


Figure 4.2: The blue dots represent the score of a recommendation at each iteration and the red line highlights the trend of every 10th score by the contextual bandit.

Time	Name	Tempo(bpm)	Loudness(db)	Mode	Score	Times played
07:00	Destiny's Child - Say My Name	67	-3.5	0	0.79	89
13:00	Destiny's Child - Say My Name	67	-3.5	0	0.77	65
15:30	George Frideric - Messiah, HWV 56	107	-16	1	0.93	64
18:00	ASAP Ferg - Plain Jane REMIX	170	-4.3	1	0.92	75
18:00	Jimi Hendrix - Hey Joe	170	-2.8	0	0.58	1
18:00	U2 - With Or Without You	110	-2.2	0	0.86	7
22:00	Anton Bruckner - Symphony No. 4	61	-19.5	1	0.82	9
22:00	D-Block - Promised Land	150	-1.7	0	0.92	60
22:00	CLC - Hobgoblin	110	-2.2	0	0.86	6

Table 4.3: A sample of the recommended songs by contextual bandit.

4.4 Linear Regression

Similar to section 4.2 and 4.3 the graph and table below represent the result for our linear regression system.

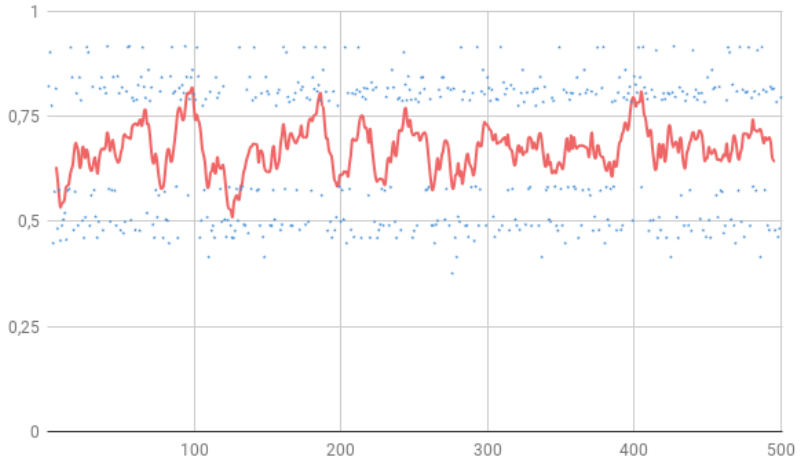


Figure 4.3: The blue dots represent the score of a recommendation at each iteration and the red line highlights the trend of every 10th score by the linear regression.

Time	Name	Tempo(bpm)	Loudness(db)	Mode	Score	Times played
07:00	Sofia Karlberg - Blue Jeans	50	-9.5	1	0.44	6
13:00	Destiny's Child - Say My Name	67	-3.5	0	0.79	7
13:00	Drake - Diplomatic Immunity	75	-5.2	1	0.47	5
15:30	Anton Bruckner - Symphony No. 4	61	-19.5	1	0.9	5
15:30	Destiny's Child - Say My Name	67	-3.5	0	0.48	14
15:30	Luke Bryan - Huntin', Fishin' ...	77	-4.2	1	0.84	9
18:00	Monsta X - SHINE FOREVER	80	-3.5	1	0.91	7
22:00	Destiny's Child - Say My Name	67	-3.5	0	0.8	11
22:00	VIXX - Shangri-La	77	-3.5	0	0.82	11

Table 4.4: A sample of the recommended songs by linear regression.

Chapter 5

Discussion

5.1 Scope

During the project we realized that, even with the use of a limited list of songs, we would run in to problems related to a lack of users to gather data from as well as a lack of usage. A large amount of users and data is needed to train a system like this, which is something we have not had access to. This problem was amplified by the lack of a stable production build at several stages of the project, limiting the amount of data that could be collected. The low amount of real life user data was the primary driving factor behind the usage of offline evaluation.

While we early on suspected that the lack of data could be an issue, we had initially planned to use a machine learning system that could give us a predicted song id directly. This meant that the number of possible combinations for a given rating was affected by both the number of songs in our playlist and the user data. Even with our limited number of songs this meant that we needed

$$Number\,of\,songs \cdot Time\,buckets \cdot Heart\,rate\,buckets = 31\,472$$

31 472 number of plays to explore all combinations of songs together with our buckets for time, heart rate and getting a rating for them for a single user. Not all of them need to be explored to get decent recommendations since many of those combinations will always be left unexplored in real life usage. A user might never play songs in the middle of the night or he might never exercise and thus never reach the higher buckets of heart rate.

By trying to generalize the songs in to features, the problem was somewhat alleviated. Instead of having to play every song, we now needed to explore all combinations of features at every heart rate and time bucket.

$$Tempo\,buckets \cdot Mode\,buckets \cdot Loudness\,buckets \cdot Heart\,rate\,buckets \cdot Time\,buckets = 6160$$

The system can then also be scaled up easier, adding more songs to the database does not affect what tempo a user prefers at a certain heart rate and time. This method of identifying what songs that should be played comes with its own disadvantages. Generalizing songs down to the values of three different attributes is not sufficient to classify songs properly. For instance if a user prefers high tempo music at high heart rate, he might be recommended both speed metal songs together with speedcore songs due to their similarity, even though he might dislike one of those genres. While the user bias we implemented in our ranking system could identify specific songs that a user dislikes, we have no way of excluding entire genres and having to skip all 3731 speedcore songs that exist on Spotify is not a very user friendly system. More features could have been introduced, this would have increased the amount of data required and the time to train and make prediction. But there is also no guarantee that including more features would enable us to identify all different types of music nor give more accurate predictions. Using more types of real time data, such as location, could have

improved recommendation accuracy. As an example, a user might always like high tempo music when located at the gym, regardless of heart rate. Using that as an extra input together with the other user information might been beneficial to the results. We did not use it as we decided that the possible gain was not worth the extra complexity, training and data required. Static data such as age, sex and country could be interesting to introduce but was deemed an unviable option for us due to the limited number and variation of users we had at hand.

5.2 Scaling

Training the models took up towards 12 hours at the end of the project even with our low amount of user data. While better hardware and optimizing the code could make the process significantly faster it is unlikely that it could be made fast enough in a real system with millions of songs and users. A long training time makes the system close to unusable as the feedback from the users need to be implemented in to our recommendations as fast as possible. If a user has skipped a song to indicate that he does not like high tempo music at his current time and heart rate, the system should not continue to recommend high tempo music.

The way our ranking system is built is also a roadblock for scaling up the system. We currently iterate through every song in our database and calculate an individual weight for them. This is already an issue with the current number of songs in our database, the ranking takes close to 3.5 seconds which makes for a very unresponsive application. This could most likely be optimized heavily in our code. For instance with better caching to reduce the number of queries to our database or by excluding large parts of the song data set that is unlikely to be ranked highly.

To make our system more responsive, caching was introduced to avoid users from making too many CPU time expensive requests to our recommender system. This introduced another problem, by the time a user got to the last song in his cached recommendations, his or her heart rate might have changed. This would mean that a recommendation would be inaccurate due to change in heart rate and a very different set of songs should have been recommended for that heart rate.

5.3 Recommendations

By having a recommender system with two different sub systems we introduced a possible weakness. If our system returns with low-rated song recommendations to the user it is hard to pin point which part of the system that is to blame. We could have a good audio feature recommendation but a bad ranking, a good ranking but a poor audio feature recommendation, or both of the parts could be performing poorly. This also leads to a greater issue when training the machine learning systems, as if the ranking system performs poorly the algorithms might receive low rewards for good recommendations which would teach it not to give that recommendation again. Currently, we have no proper solution to this problem as combining the two parts into one would demand extremely large data sets and in the case of the contextual bandit probably would not work at all as it would have too many possible actions in order to be trained properly. Another solution to the problem could involve splitting the ranking system into multiple smaller parts where each part gets a specific task. This sort of solution could make it easier to find malfunctions in the system but could also make it more complex as, for instance, a poor ranking could have multiple different sources.

5.4 Results

The evaluation displays that in our application the contextual bandit preforms best. Unlike linear regression and deep neural network the contextual bandit understands the connection between what time of day the user is listening and preferred tempo at that specific time. The LR and DNN models recommends songs with a good

average score without consideration of the time. Thus leading to many repeated recommendations. Because contextual bandit identifies this connection, we see a much larger variation in recommendations. The fact that our contextual bandit is configured to use random exploratory actions 10% of the time also accounts for a greater variation in song selection. One possible reason for the high average rating the contextual bandit scored might be because our limited amount of songs and our moderate amount of iterations.

Due to using few attributes it was difficult for all network to recommend song similar to each other. The networks were not able to understand the different genres and thus a classical song and a Hip Hop song could receive equally high score if their song features were similar. This is displayed in the recommended song Destiny's Child - Say My Name and VIXX - Shangria-La where these two songs have very similar attributes but are very unlike.

Our evaluation did not evaluate the influence of the heart rate on the recommendation which was one of the main purposes of this project. This might have been possible if conducting an evaluation on real users. However this would require that we would have someone use the application for a very long time enabling the network to have an appropriate training but due to the projects short deadline this was not possible. Another possible solution was that one of the team members would use the application, however as mentioned in 5.1 this would not be viable.

5.5 Workflow

At the initial stage of the project, a lot of time was spent on researching machine learning due to the group members inexperience with the subject. This delayed the actual implementation of the product and our data collection. While a prototype was completed in reasonable time, bugs and the lack of a stable production build also hampered data collection and user testing. Stricbibliotter rules for submissions to the main branch of our git repository should have been discussed and agreed upon at an early stage.

The capacity of our server also limited how fast we could iterate on our recommender system. Since training was slow, group members often had to wait a day to see how their code changes affected the recommendations. This was debilitating to progress when many small changes had to be made.

5.6 Ethical Aspects

There are definitely ethical aspects to consider for this project. The most obvious aspect is the matter of data collection and data storage. Our service will handle some possibly sensitive data, such as the user's heart rate at a given time and the user's music preferences. It is important that people who use the service are made aware of which data is gathered and stored, as well as how this data is handled. Since the data that would be gathered if our service were to be released to the public could be of interest to various third parties, it is important that we as a group discuss our views on matters like data gathering for commercial use and make our views clear to eventual users of the service. Since some sensitive data is to be handled, it is also important to make sure our database and the software connecting to the database are secure and that important data is encrypted, making it not vulnerable to data breaches. As of writing this paper, the intention is not to release the application to the public which is why there is no concrete conclusion on how data storage and collection will be handled. We do however have a privacy policy in use, which is referenced in appendix A.

5.7 Impact on Society

Better music recommendations might not change our world but the use of real time data in automated systems is something that is becoming more and more prevalent. Using more data to give better assessments and predictions is something that can be applied to a multitude of problems. Machine learning is a powerful tool coupled with the cheaper and more powerful processors and GPUs we have available to us due to Moore's law and the enormous amount of data that is currently being generated on the internet. While it is wrong to think that machine learning techniques is a magic solution that can be applied to any problem, with careful planning and consideration it can be used to huge success.

5.8 Experience

As mentioned in the section 5.5, the group lacked experience and knowledge of machine learning as well as recommendation services. This proved to be a hindrance for the project as a lot of time was spent initially researching these subjects. If we had that knowledge before the project, the implementation could have been done earlier and with less issues along the way, specifically for the machine learning part. As we did not know what implementation to initially go with for the feature recommendation system, a lot of time was spent researching specifically what implementation would be viable. When we decided on a implementation, largely due to the insight of Mikael Kågebäck, actually getting the system implemented took longer than expected. This meant that we had less time for experimentation than we would have liked. Therefore, if someone decides to build upon this project we would recommend previous knowledge of both machine learning and recommender systems.

Chapter 6

Conclusion

While basing music recommendations on a combination of real time data and user history is not in itself a bad idea and using machine learning as a way to recommend items is a well-proven technique, our implementation has a plethora of issues. To have a system that could be scaled to realistic levels, generalization between users is needed, perfect personalization is not feasible. While we could see that our recommendations got better over time, one large reason was that our simulated user had completely static music preferences. Machine learning cannot handle quick variations in user preferences due to the long time of training and is better used as a way to give more generalized recommendations. Using offline evaluation also prevented us from exploring if there was a clear link between the selected song features and a users heart rate since the simulated user was created with a set song feature preference at different times and heart rates.

Due to our limited ability of testing the application with online evaluation, our result has limited use. As we could only evaluate the system with simulated users we do not know how an actual user would rate the system. Our result says that the feature recommender system actually improves but we do not know for sure if it improves in such a way that it increases user satisfaction. It would have been interesting to try using other audio features than the ones implemented. However, that would require online evaluation to determine whether these audio features gave a better result. Because offline evaluation can only answer whether the application can find the optimal value for the implemented audio features. Another interesting experiment would be to exchange heart rate for another real-time parameter to see whether heart rate is actually a viable parameter to utilize. But as this would also require online evaluation, it was not possible for us to experiment with this.

If this project was to be repeated we would suggest limiting the scope to one part of the system, either making a good ranking algorithm or making feature suggestions. For the other part we would recommend to use an already existing library or similar work. This would allow the work to be more in depth as both parts of the program ended up being rather time-consuming and broad subjects. We would also recommend using a contextual bandit as it was the one that seemed most suitable for this problem. As mentioned in the discussion the amount of data proved to be a limiting factor and having access of a data set with usable data, or the ability to gather such data, would enable better training and testing.

Chapter 7

Bibliographic Notes

7.1 Recommendation Systems

Recommender systems: Principles, methods and evaluation [15] – Thorough review of different recommendation techniques and their strengths and weaknesses, useful in background to the approach of this project.

Evaluating Recommendation Systems. [31] – Paper on how to test and evaluate recommendations. Very useful for evaluation of result of this project.

A User-Centric Evaluation Framework for Recommender Systems [28] – A paper that evaluates users experiences regarding quality of recommendations.

A Contextual-Bandit Approach to Personalized News Article Recommendation [23] – Paper that presents a contextual bandit algorithm for making personal recommendations. Gives insight in to how contextual bandits can be implemented and how to evaluate them.

Reinforcement Learning based Recommender System using Biclustering Technique [4] – A paper where a reinforcement learning algorithm for giving recommendations are developed. A biclustering technique is used to reduce the state and action space. Relevant as it shows an example of how reinforcement learning can be useful in giving recommendations.

Restricted Boltzmann Machines for Collaborative Filtering [30] – A paper that describes how RMB can be used for giving recommendations, which is applicable on very large data sets. The article is a bit dated because there have been much development in machine learning since 2007 but could still be relevant.

A smartphone-based activity-aware system for music streaming recommendation [38] – Music recommendation based on the users current activity and mood. The study does not use heart rate but still possesses useful information on activity recognition, machine learning techniques and classification.

Clu-PoF-A Novel Post Filtering Approach for Efficient Context Aware Recommendations [36] – A paper on how to utilize contextual information about user and item when giving recommendations.

Two Decades of Recommender Systems at Amazon.com [24] – Article that describes Amazon's item based collaborative filtering recommender system and its development. Useful example.

A User-Centric Evaluation Framework for Recommender Systems [28] – Paper that presents a framework that evaluates different recommender systems based on user experience. Useful in how it narrows down what con-

stitutes a satisfying recommender system.

How music recommendation works — and doesn't work [40] — Article that describes how Echo Nest, which is the company that Spotify uses for recommandations, analyze and label music.

7.1.1 Neural Networks and Deep Learning

Deep Learning based Recommender System: A Survey and New Perspectives. [33] – Extensive survey of recent years research and advancements in recommender systems using deep learning. The paper also points out some open problems in the field and describes the newest trends in deep learning techniques for recommender systems.

Wide and Deep Learning for Recommender Systems [19] – Paper that present a wide and deep learning that utilizes both wide linear models and deep neural networks to give recommendations.

Content-aware collaborative music recommendation using pre-trained neural networks. [9] – Paper that shows that incorporating music content in a collaborative filtering approach can solve the "cold-start problem". Relevant for understanding of music recommender systems.

Deep content-based music recommendation. [1] – Conference paper that shows that a deep neural network based on audio signals can be used to predict latent factors for music recommendations that user data can't show.

Recommending music on Spotify with deep learning [10] – Sander Dieleman interns at Spotify and uses deep learning to cluster songs together based on the model from the paper [1] where he was co-author. Interesting with an industrial perspective and connects to the data (Spotify's song data) used in this thesis.

Deep Neural Networks for YouTube Recommendations [27] – Paper that describes on a high level how YouTube recommendations are being made. Helpful with real world examples of neural network recommender systems.

Hybrid Collaborative Filtering with Neural Networks [14] – Paper that introduce a neural network to perform collaborative filtering with side information to avoid cold start.

DeepPlaylist: Using Recurrent Neural Networks to Predict Song Similarity [6] – A paper that uses a DNN to see if two songs are similar based on lyrics and/or sound. Very relevant in recommender systems to find similar songs so not only the most popular songs get recommended.

7.2 Connecting Music to Heart Rate, Activity or Mood

Relationship Between Exercise Heart Rate and Music Tempo Preference [8] – Article on how music tempo preference correlates to exercise heart rate, supports the choice of heart rate as real time parameter for music recommendation.

Music can make the heart beat faster [12] – Article on how music can affect the heart rate and blood pressure, not very relevant and we will probably not use it.

Stress-relieving music [20] – Study of the development of personalized music recommendations to lower stress levels. It is probably not relevant to this paper.

Cardiovascular, cerebrovascular, and respiratory changes induced by different types of music in musicians

and non-musicians: the importance of silence [22] – Research on how listening to different music affects cardiovascular and respiratory variables. Not very relevant.

Music and the heart. [35] – An article on what, and if, music has an effect on the heart. Is used for showing a connection between exciting/tranquilizing music and heart rate.

Maladaptive and adaptive emotion regulation through music: a behavioural and neuroimaging study of males and females [3] – Study that mesaure neural responses to music to make connections to mental health. Interesting in that it shows very clear correlations between different music and different emotional states.

Harmonic relationship between preferred tempi and heart rate [25] – Research on if there is an harmonic relationship between preferred tempi and a users heart rate.

Mode and tempo relative contributions to "happy-sad" judgements in equitone melodies [16] – A paper that connects the mode and tempo of a song to how "happy" or "sad" the song is perceived.

The effects of music tempo and loudness level on treadmill exercise [13] – A paper that examined the effect of loudness and tempo of background music during exercise.

7.3 Machine Learning and Neural Networks in General

Deep Learning [17] – Book on deep learning and the mechanics behind it, as well as other machine learning techniques. Useful for describing the learning processes of the algorithms.

Real-Time Learning Capability of Neural Networks [18] – Article that highlights the difficulties of real time learning for neural network in some applications. Maybe outside of the scope but could be useful regarding discussion of upscaling of the product presented in this thesis.

Multiworld Testing Decision Service [2] – A whitepaper on a Multiworld Testing which is a machine learning system based on contextual bandits from Microsoft research.

Doubly Robust Policy Evaluation and Learning [26] – Research on how contextual bandits perform when using the doubly robust technique.

A New Prediction Approach Based on Linear [5] – A paper about using linear regression in recommender system. Introduces an approach for an effective linear regression model using the Pearson Correlation Coefficient.

A Linear Regression Approach to Multi-criteria Recommender System [37] – Paper that suggests using multilinear regression and multi-criteria collaborative filtering as a basis for a recommender system.

7.4 Resources

Django [11] – Web framework used for the web server. The most commonly used python web framework in the world.

Postgresql [29] – A advanced open source relational database that uses the SQL database language. *Correlation Coefficient* [39] – A page explaining what a correlation coefficient is and how it is used.

References

[1] Benjamin Schrauwen Aaron van den Oord Sander Dieleman. *Deep content-based music recommendation*. 2013. URL: https://papers.nips.cc/paper/5004-deep-content-based-music-recommendation.pdf.

[2] Alekh Agarwal et al. *Multiworld Testing Decision Service: A System for Experimentation, Learning, And Decision-Making*. June 2015. URL: https://www.researchgate.net/publication/305881203_A_Multiworld_Testing_Decision_Service.

[3] Emily Carlson et al. "Maladaptive and adaptive emotion regulation through music: a behavioral and neuroimaging study of males and females". In: *Frontiers in Human Neuroscience* (Aug. 2015). URL: https://www.frontiersin.org/articles/10.3389/fnhum.2015.00466/full.

[4] Sungwoon Choi et al. *Reinforcement Learning based Recommender System using Biclustering Technique*. Jan. 2018. URL: https://arxiv.org/pdf/1801.05532.pdf.

[5] Xinyang Ge et al. "A New Prediction Approach Based on Linear Regression for Collaborative Filtering". In: *2011 Eighth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)* 4 (July 2011). URL: https://ieeexplore.ieee.org/document/6020007/.

[6] Anusha Balakrishnan and Kalpit Dixit. *DeepPlaylist: Using Recurrent Neural Networks to Predict Song Similarity*. 2016. URL: https://cs224d.stanford.edu/reports/BalakrishnanDixit.pdf.

[7] Jorean Beel and Stefan Langer. *A Comparison of Offline Evaluations, Online Evaluations, and User Studies in the Context of Research-Paper Recommender Systems*. Nov. 2015. URL: https://link.springer.com/chapter/10.1007%2F978-3-319-24592-8_12.

[8] Leighton Jones Costas I. Karageorghis and Daniel C. Low. "Relationship Between Exercise Heart Rate and Music Tempo Preference". In: *Research Quarterly for Exercise and Sport* 77.2 (2006). URL: https://doi.org/10.1080/02701367.2006.10599357.

[9] Minshu Zhan Dawen Liang and Daniel P. W. Ellis. *Content-aware collaborative music recommendation using pre-trained neural networks*. 2015. URL: http://ismir2015.uma.es/articles/290_Paper.pdf.

[10] Sander Dieleman. *Recommending music on Spotify with deep learning*. Aug. 2014. URL: http://benanne.github.io/2014/08/05/spotify-cnns.html.

[11] *Django Web Framework*. Accessed 2018-04-24. URL: https://www.djangoproject.com/start/overview/.

[12] Ed Edelson. *Music can make the heart beat faster*. URL: http://abcnews.go.com/Health/Healthday/story?id=7902380&page=1.

[13] Judy Edworthy and Hannah Waring. "The effects of music tempo and loudness level on treadmill exercise". In: (2007). URL: https://doi.org/10.1080/00140130600899104.

[14] Jeremie Mary Florian Strub and Romaric Gaudel. *Hybrid Collaborative Filtering with Neural Networks*. Mar. 2016. URL: https://pdfs.semanticscholar.org/fcbd/179590c30127cafbd00fd7087b47818406bc.pdf.

[15] Y.O.Folajimi F.O.Isinkaye and B.A.Ojokoh. "Recommendation systems: Principles, methods and evaluation". In: *Egyptian Informatics Journal* 16.3 (Nov. 2015), pp. 261–271. URL: https://www.sciencedirect.com/science/article/pii/S1110866515000341.

[16] Lisa Gagnon and Isabelle Peretz. “Mode and tempo relative contributions to “happy-sad” judgements in equitone melodies”. In: *Cognition and Emotion* 17.1 (2003), pp. 25–40. URL: https://www.tandfonline.com/doi/abs/10.1080/02699930302279.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[18] Qin-Yu Zhu Guang-Bin Huang and Chee-Kheong Siew. “Real-Time Learning Capability of Neural Networks”. In: *IEEE TRANSACTIONS ON NEURAL NETWORKS* 17.4 (2006 7), pp. 863–878. URL: https://pdfs.semanticscholar.org/cbdb/093c51072f696503975c73880a9dd3a44bcf.pdf.

[19] Levent Koc Heng-Tze Cheng and Jeremiah Harmsen. *Wide and Deep Learning for Recommender Systems*. June 2016. URL: https://arxiv.org/pdf/1606.07792.pdf.

[20] Il-hyung Shin Jaepyeong Cha and Gyeong Woo Cheon. *Stress-relieving music*. 2014. URL: http://ieeexplore.ieee.org/document/6945093/metrics?part=1.

[21] C. I et al Karageorghis. “Revisiting the exercise heart rate-music tempo preference relationship”. In: *Research Quarterly for Exercise and Sport* 82 (2011). URL: https://doi.org/10.1080/02701367.2011.10599755.

[22] C Porta L Bernardi and P Sleight. “Cardiovascular, cerebrovascular, and respiratory changes induced by different types of music in musicians and non-musicians: the importance of silence”. In: *Heart* 92.4 (2006). URL: http://heart.bmj.com/content/92/4/445.

[23] John Langford Lihong Li Wei Chu and Robert E. Schapire. *A Contextual-Bandit Approach to Personalized News Article Recommendation*. Mar. 2012. URL: https://arxiv.org/pdf/1003.0146.pdf.

[24] Brent Smith andGreg Linden. “Two Decades of Recommender Systems at Amazon.com”. In: *IEEE Internet Computing* 21.3 (2017). URL: https://www.computer.org/csdl/mags/ic/2017/03/mic2017030012.html.

[25] Iwanaga M. *Harmonic relationship between preferred tempi and heart rate*. 1995. URL: https://www.ncbi.nlm.nih.gov/pubmed/8532483.

[26] John Langford Miroslav Dudík and Lihong Li. *Doubly Robust Policy Evaluation and Learning*. Mar. 2011. URL: https://arxiv.org/pdf/1103.4601.pdf.

[27] Jay Adams Paul Covington and Emre Sargin. *Deep Neural Networks for YouTube Recommendations*. 2016. URL: https://static.googleusercontent.com/media/research.google.com/sv//pubs/archive/45530.pdf.

[28] Li Chen Pearl Pu and Rong Hu. “A User-Centric Evaluation Framework for Recommender Systems”. In: *Proceedings of the fifth ACM conference on Recommender Systems* (Oct. 2011), pp. 157–164. URL: https://dl.acm.org/citation.cfm?id=2043962.

[29] PostgreSQL. *PostgreSQL*. 2018. URL: https://www.postgresql.org/.

[30] Geoffrey Hinton Ruslan Salakhutdinov Andriy Mnih. *Restricted Boltzmann Machines for Collaborative Filtering*. June 2007. URL: http://www.cs.toronto.edu/~rsalakhu/papers/rbmcf.pdf.

[31] Guy Shani and Asela Gunawardana. *Evaluating Recommendation Systems*. Oct. 2010. URL: http://www.bgu.ac.il/~shanigu/Publications/EvaluationMetrics.17.pdf.

[32] Guy shani and Asela Gunawardana. *Evaluating Recommendation Systems*. 2011. URL: https://link.springer.com/content/pdf/10.1007%2F978-0-387-85820-3.pdf.

[33] Lina Yao Shuai Zhang and Aixin Sun. *Deep Learning based Recommender System: A Survey and New Perspectives*. July 2017. URL: https://www.researchgate.net/publication/318671349_Deep_Learning_based_Recommender_System_A_Survey_and_New_Perspectives.

[34] Spotify. *Spotify API*. URL: https://beta.developer.spotify.com/documentation/web-api/.

[35] Lutz Jäncke Stefan Koelsch. *Music and the heart*. Sept. 2015. URL: https://academic.oup.com/eurheartj/article-pdf/36/44/3043/6733702/ehv430.pdf.

[36] Anu Taneja and Anuja Arora. “Clu-PoF-A Novel Post Filtering Approach for Efficient Context Aware Recommendations”. In: *Procedia Computer Science* 122 (2017), pp. 834–841. URL: https://ac.els-cdn.com/S1877050917326923/1-s2.0-S1877050917326923-main.pdf?_tid=d5e5d093-f901-4a39-953a-385eb836a5cb&acdnat=1523263769_37aac36df0de8c188394f4674f1fde3f.

[37] Vibhor Kant Tanisha Jhalani and Pragya Dwivedi. “A Linear Regression Approach to Multi-criteria Recommender System”. In: *Data Mining and Big Data* (2016), pp. 235–243. URL: https://doi.org/10.1007/978-3-319-40973-3_23.

[38] Jhih-Yuan Huang Wei-Po Lee Chun-Ting Chen and Jhen-Yi Liang. “A smartphone-based activity-aware system for music streaming recommendation”. In: *Knowledge-Based Systems* 131 (Sept. 2017), pp. 70–82. URL: https://www.sciencedirect.com/science/article/pii/S0950705117302757.

[39] Eric W. Weisstein. “Correlation Coefficient.” In: (). URL: http://mathworld.wolfram.com/CorrelationCoefficient.html. (accessed: 12.05.2018).

[40] Brian Whitman. “How music recommendation works — and doesn’t work”. In: (2018). URL: https://notes.variogr.am/2012/12/11/how-music-recommendation-works-and-doesnt-work/.

[41] Joost de Wit. *Evaluating recommender systems*. May 2008. URL: http://essay.utwente.nl/59711/1/MA_thesis_J_de_Wit.pdf.

Appendix A

Privacy Policy

Owner and Data Controller

Types of Data collected

Among the types of Personal Data that this Application collects, by itself or through third parties, there are: heart rate and other vital data.
Complete details on each type of Personal Data collected are provided in the dedicated sections of this privacy policy or by specific explanation texts displayed prior to the Data collection.
The Personal Data may be freely provided by the User, or, in case of Usage Data, collected automatically when using this Application.
All Data requested by this Application is mandatory and failure to provide this Data may make it impossible for this Application to provide its services. In cases where this Application specifically states that some Data is not mandatory, Users are free not to communicate this Data without any consequences on the availability or the functioning of the service.
Users who are uncertain about which Personal Data is mandatory are welcome to contact the Owner.
Any use of Cookies – or of other tracking tools – by this Application or by the owners of third-party services used by this Application serves the purpose of providing the service required by the User, in addition to any other purposes described in the present document and in the Cookie Policy, if available.
Users are responsible for any third-party Personal Data obtained, published or shared through this Application and confirm that they have the third party's consent to provide the Data to the Owner.

Mode and place of processing the Data

Methods of processing

The Data Controller processes the Data of Users in a proper manner and shall take appropriate security measures to prevent unauthorized access, disclosure, modification, or unauthorized destruction of the Data.
The Data processing is carried out using computers and/or IT enabled tools, following organizational procedures and modes strictly related to the purposes indicated. In addition to the Data Controller, in some cases, the Data may be accessible to

certain types of persons in charge, involved with the operation of the site (administration, sales, marketing, legal, system administration) or external parties (such as third-party technical service providers, mail carriers, hosting providers, IT companies, communications agencies) appointed, if necessary, as Data Processors by the Owner. The updated list of these parties may be requested from the Data Controller at any time.

Place

The Data is processed at the Data Controller's operating offices and in any other places where the parties involved with the processing are located. For further information, please contact the Data Controller.

Retention time

The Data is kept for the time necessary to provide the service requested by the User, or stated by the purposes outlined in this document, and the User can always request that the Data Controller suspend or remove the data.

The use of the collected Data

The Data concerning the User is collected to allow the Owner to provide its services, as well as for the following purposes: Handling activity data.

The Personal Data used for each purpose is outlined in the specific sections of this document.

Detailed information on the processing of Personal Data

Personal Data is collected for the following purposes and using the following services:

Handling activity data

This type of service allows the Owner to use the activity data collected by your device in order for this Application to operate or to provide specific features. This may include movements, heartbeat, change in altitude or data about the surroundings.

Depending on what is described below, third parties may be involved in the activity tracking.

Most devices allow for the User to control which Data is accessed or stored.

Apple HealthKit (Apple Inc.)

HealthKit is an activity data handling service provided by

Apple Inc. that enables the Owner to access or store activity data.
Personal Data collected: heart rate and other vital data.
Place of processing: US – Privacy Policy

Additional information about Data collection and processing

Legal action

The User's Personal Data may be used for legal purposes by the Data Controller, in Court or in the stages leading to possible legal action arising from improper use of this Application or the related services.
The User declares to be aware that the Data Controller may be required to reveal personal data upon request of public authorities.

Additional information about User's Personal Data

In addition to the information contained in this privacy policy, this Application may provide the User with additional and contextual information concerning particular services or the collection and processing of Personal Data upon request.

System logs and maintenance

For operation and maintenance purposes, this Application and any third-party services may collect files that record interaction with this Application (System logs) or use for this purpose other Personal Data (such as IP Address).

Information not contained in this policy

More details concerning the collection or processing of Personal Data may be requested from the Data Controller at any time. Please see the contact information at the beginning of this document.

The rights of Users

Users have the right, at any time, to know whether their Personal Data has been stored and can consult the Data Controller to learn about their contents and origin, to verify their accuracy or to ask for them to be supplemented, cancelled, updated or corrected, or for their transformation into anonymous format or to block any data held in violation of the law, as well as to oppose their treatment for any and all

legitimate reasons. Requests should be sent to the Data Controller at the contact information set out above.
This Application does not support “Do Not Track” requests. To determine whether any of the third-party services it uses honor the “Do Not Track” requests, please read their privacy policies.

Changes to this privacy policy

The Data Controller reserves the right to make changes to this privacy policy at any time by giving notice to its Users on this page. It is strongly recommended to check this page often, referring to the date of the last modification listed at the bottom. If a User objects to any of the changes to the Policy, the User must cease using this Application and can request that the Data Controller remove the Personal Data. Unless stated otherwise, the then-current privacy policy applies to all Personal Data the Data Controller has about Users.

Information about this privacy policy

The Data Controller is responsible for this privacy policy, prepared starting from the modules provided by Iubenda and hosted on Iubenda's servers.

Definitions and legal references

Personal Data (or Data)

Any information regarding a natural person, a legal person, an institution or an association, which is, or can be, identified, even indirectly, by reference to any other information, including a personal identification number.

Usage Data

Information collected automatically from this Application (or third-party services employed in this Application), which can include: the IP addresses or domain names of the computers utilized by the Users who use this Application, the URI addresses (Uniform Resource Identifier), the time of the request, the method utilized to submit the request to the server, the size of the file received in response, the numerical code indicating the status of the server's answer (successful outcome, error, etc.), the country of origin, the features of the browser and the operating system utilized by the User, the various time details per visit (e.g., the time spent on each page within the Application) and the details about the path followed within the Application with special reference to the sequence of

pages visited, and other parameters about the device operating system and/or the User's IT environment.

User
The individual using this Application, which must coincide with or be authorized by the Data Subject, to whom the Personal Data refers.

Data Subject
The legal or natural person to whom the Personal Data refers.

Data Processor (or Data Supervisor)
The natural person, legal person, public administration or any other body, association or organization authorized by the Data Controller to process the Personal Data in compliance with this privacy policy.

Data Controller (or Owner)
The natural person, legal person, public administration or any other body, association or organization with the right, also jointly with another Data Controller, to make decisions regarding the purposes, and the methods of processing of Personal Data and the means used, including the security measures concerning the operation and use of this Application. The Data Controller, unless otherwise specified, is the Owner of this Application.

This Application
The hardware or software tool by which the Personal Data of the User is collected.

Legal information
Notice to European Users: this privacy statement has been prepared in fulfillment of the obligations under Art. 10 of EC Directive n. 95/46/EC, and under the provisions of Directive 2002/58/EC, as revised by Directive 2009/136/EC, on the subject of Cookies.
This privacy policy relates solely to this Application.