KTH Computer Science
and Communication

Learning Playlist Representations for Automatic Playlist Generation

ERIK AALTO

Master's Thesis at Spotify and CSC
KTH Supervisor: Carl Henrik Ek
Company Supervisor: Boxun Zhang
KTH Examiner: Danica Kragic

TRITA xxx yyyy-nn

Abstract

This is a skeleton for KTH theses. More documentation regarding the KTH thesis class file can be found in the package documentation.

Referat

Denna fil ger ett avhandlingsskelett. Mer information om LaTeX-mallen finns i dokumentationen till paketet.

Abstract

Acknowledgements I would like to thank ....surprise :)

Should be Acknowledgements and not abstract

Contents

Chapter 1

Introduction

1.1 What is Spotify?

Spotify is a music streaming service which charges premium users a fee and presents free users with ads. Record companies are then paid according to the popularity of the tracks for which they hold digital rights. Spotify was launched in October 2008 and today has over 60 million active users, from which over 15 million are paying for the premium user service.

1.2 Project Motivation

Spotify today has more playlists than songs in their music library. Spotify also provides curated playlists as a form of music recommendation for their users.

Given that a user has a preference for a specific playlist, an interesting feature would be to generate a playlist similar to the one a user has a preference for, but with different songs. This type of feature is interesting as it allows users to get music recommendations fitted to their needs. Such a feature could also give Spotify a competitive edge in the hardening competition for music streaming customers.

1.3 What are Recommender Systems?

Recommender systems provide an automated way to filter and rank information of interest for a certain user, possibly also taking time into account. A famous example of recommender systems is the product recommendation once initiated at Amazon, *"Users who bought this product also bought"*. Another example of recommender systems are the movie recommendations provided by Netflix. Movie recommendations are interesting and non-trivial as a specific user at a certain time is likely to not be interested in the majority of movies provided by Netflix. The same thing applies to music, at any given moment a user is likely to not want to listen to the majority of songs in a music library . A last example of recommendation could be restaurant recommendation, where time and context are important factors. Recommending

a simple hamburger restaurant is not likely to be of interest at date night, but it might be the perfect recommendation while driving the kids home after Saturday morning soccer game.

1.4 Project Aim

The aim of this thesis is to provide a scalable method for selecting candidate songs, in the context of playlist generation given a predefined playlist to mimic. This is an extension to the current field of music recommendation.

The work of this thesis is limited to finding candidate songs when generating playlists similar to Spotify *Browse* playlists. The focus is on creating a scalable model of doing so. This means that any type of feature engineering is excluded from this thesis and the thesis is also limited from looking into the problem of ordering songs in playlist generation.

Part I

Theory

Chapter 2

Background

Previous work within the recommender system domain mainly focuses on two approaches. These are collaborative filtering and content based approaches. A hybrid of these two approaches can also be used. Both collaborative filtering and content based approaches typically try to infer a user ranking for a specific item[11]. An item would in the context of music recommendation be a song, artist or album.

2.1 Collaborative Filtering

Collaborative filtering focuses on user's past behaviour. From this past behaviour of a specific user and past behaviour of similar users the ranking for the specific user for a certain item is inferred[18][19]. In other words, a user gets recommendations of items that other users with similar taste like[1]. Collaborative filtering suffers from something called the cold start problem, which occurs when the ranking for a specific item and user is inferred when there is no or little information of the current user behaviour[6]. Collaborative filtering has the advantage that it only relies on past user behaviour without the need of explicit user profiles. The fact that collaborative filtering only looks at user data means that it is domain free, i.e. the model is not dependent on whether users have rated books, movies, music or a combination thereof[8].

Collaborative filtering can be used with explicit user feedback, such as the movie ratings used by Netflix, but it can also be used with implicit user feedback[8]. In the music context implicit feedback could be whether a song has been played or skipped.

An example of collaborative filtering applied to music recommendation is the recommender system used by last.fm. Collaborative filtering is also part of the music recommendation pipeline used in production at Spotify.

Collaborative filtering methods can be divided into two categories, memory-based and model-based. Memory-based collaborative filtering algorithms can be seen as user based while the model-based algorithms can be seen as item based[18].

Memory-based collaborative filtering algorithms operate on the entire user-item

matrix where the full user history data set is used. The user-item matrix could for example consist of one user per row and one item per column. This data set is used to predict a preference for a previously unseen item for a specific user. To do this the rows most similar to the row corresponding to the specific user are found. The ratings of the users corresponding to these rows for the unseen item are then used to predict the rating for the specific user. As similar user's ratings are used to predict a specific user's rating memory-based models can also be thought of as neighbourhood models[8]. There are various ways implementing a memory-based model, but a naive way could be to find rows by using the cosine similarity and then simply averaging the rating of the top-n similar users for a specific item. This naive approach has a $O(MN^2)$ complexity where M is the number of users and N the number of items. One downside of this approach is that it does not scale very well when the user-item matrix is large. Another downside is that the user-item matrix is likely to be very sparse and using a nearest neighbour approach in this setting can lead to poor performance[18][19].

Model-based collaborative filtering means that the user history data is used to create a probabilistic model for ratings. At run time the model, rather than the entire user history data set, is used to make predictions of items for users. Model-based approaches are likely to scale better than memory-based ones[18]. One approach to model-based collaborative filtering is to use latent factors. This means that each user would be associated with a user-factors vector $x_u \in R^f$ and each item with an item-factors vector $y_i \in R^f$. The predicted value of a user for an item would then be the inner product between the corresponding user and item vectors, i.e. $\hat{r}_{ui} = x_u^T y_i$. To avoid overfitting the model can be regularized, which means including a bias. A cost function as follows is then obtained:

$$min_{x_*,y_*} \sum (r_{ui} - x_u^T y_i)^2 + \lambda(||x_u||^2 + ||y_i||^2) \tag{2.1}$$

The problem with equation 5.1 is that it assumes knowledge of explicit feedback. In the context of music recommendation the case is rather that implicit feedback is available than explicit. What can be done in this case is to use binary labels expressing whether a user has preference for an item or not. Having preference for an item could mean that the user has streamed that song and not skipped it for example. Therefore the binary variable p_{ui} is used to describe user preference.

There is however an uncertainty to the preference a user has. Has a user really preference for a song that come on Spotify Radio while the user was in another room? What can be done is to create confidence variables, that could depend on the number of times a song has been streamed. What can be done here is to use another variable

$$c_{ui} = 1 + \alpha r_{ui}$$

where r_{ui} is the number of times user u has streamed item i.

The resulting cost function then becomes:

$$min_{x_*,y_*} \sum c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda(||x_u||^2 + ||y_i||^2) \tag{2.2}$$

Problems still remain as users and items can contain bias. The remedy is to enter bias terms, the resulting cost function is then:

$$min_{x_*,y_*} \sum c_{ui}(p_{ui} - x_u^T y_i - b_u - b_i)^2 + \lambda(||x_u||^2 + ||y_i||^2) \tag{2.3}$$

Where b_u is the user bias term and b_i is the item bias term.

The resulting problem is a non-convex optimization problem, but by fixing either the user or item vectors the problem becomes convex and can be solved by the use of alternating least squares, where the cost function is guaranteed to get a lower value with each iteration[8].

2.2 Content Based Approaches

Content based approaches for recommender systems recommend items that are similar to items a user has had preference for in the past. This can be done by either comparing items to items or to create a user profile based on a users preferred itemsadomavicius2005toward. Content based approaches look at discrete features of items and tries to infer a similarity between two items given their similarity of features. A parallel can be drawn between content based recommendation and information retrieval. In the context of content based recommendation the cost function that is minimized is the distance between items.[1]. One of the main features of content based recommenders are that they are able to provide recommendations even when little user history data is available, something that is one of the major drawbacks of collaborative filtering[4].

Different approaches can be used to create the features used in content based recommendation in the music domain. One approach is to simply have human experts annotating tracks with information[14][20]. Other approaches could be to extract properties from the audio signal. One such example is the use of MFCCs, which creates features from short time intervals of each track[10] and another is to use Deep Belief Networks[5].

An interesting property of content based recommendation is that it allows for relevance feedback, for example with use of the Rocchio algorithm. The Rocchio algorithm allows for a user to select a subset of recommended items as relevant and move the recommendations displayed towards the direction of those items in the vector space items are represented in[15].

An example of a content based approach within music recommendation are the recommendations made by online radio station Pandora.

Downsides with content based recommendation are that a user can never be recommended something that is not similar to what the user has expressed preferences before in the past. Further, content based recommendation is limited to the features of items. If the features used to describe items are poor a content based recommender system is likely to perform poorly. Lastly, content based recommenders do not take sequential information into account. Thus a well written news article

is seen identical to the same article written backwards as they contain exactly the same words[1].

2.3 Hybrid Systems

Hybrid systems are recommenders that combine both the techniques of collaborative filering and content based filtering, with the purpose of thus obtaining better recommendations. The underlying assumption is that a combination of content based recommenders and recommenders using collaborative filtering can redeem the weaknesses those methods face on their own[4].

Hybrid recommenders can be made by combining the results of collaborative filering methods with content based methods, by incorporating properties of one method into the other or by creating a model that incorporates properties of both types of systems[1].

Chapter 3

Previous Work

3.1 Probabilistic Graphical Models for Playlist Generation

Earlier attempts of playlist generation has been made by Microsoft Research. Ragno, Burges and Herley has made a model for playlist generation that can take any type of ordered playlist material, such as curated playlists or albums, as training data, and constructs an undirected graph between songs that are within the reach of a nth-order Markov model. In this graph nodes constitute songs and edges get their weights depending on how many times two songs fulfill the nth-order Markov property. Once this is done the undirected graph is converted into a directed graph where edges weightÂ´s, the transition probabilities, are normalized by the sum of outgoing weights from each node. Once the undirected graph is made a playlist can be generated by selecting an initial seed song and simply performing a random walk in the graph. This model assumes that the connectivity between songs does not have to take order into account and that one can prevent playlist drifting by adding higher order Markov properties[17].

From a contextual playlist generation perspective a problem with the approach taken by Rango, Burges and Herley is that if you generate a playlist from a random walk you cannot chose the playlist context for the generated playlist on before hand. Another problem with the probabilistic graphical model approach to playlist generation is that the graph created during training phase only works for songs that are in the training data set. This model is not generalizable so you cannot get a similar playlist to a playlist you like, but with different songs.

3.2 Gaussian Processes for Playlist Generation

Another approach to playlist generation is to use gaussian processes, this approach has been taken by Platt et al, also at Microsoft Research. Here the authors try to learn a gaussian process prior from training data. This prior is then used together with a set of songs, for which a user has expressed preference, to generate a playlist given an initial seed song. In the training phase a blend of linear kernels is used to

learn the relationship of meta data features among songs that come in sequence. The coefficients for each linear kernel is learnt by finding the coefficient that minimizes the difference between the empirical covariance between songs and the value given by the linear kernel. Empirical covariance is in this case as simple as whether the training data songs belong to the same album or not. Once the training phase is done the playlist generation phase consists of predicting the user preference for each song in a set of candidate songs, i.e. the f-star function in this case is the predicted user preference for a song. The f-star value is calculated by weighing the blend of linear kernels between a seed song and each candidate song with a factor. This factor is the sum of similarity between the initial seed song and each user preference song weighted by how central each user preference song is in the preference space. Playlist generation is then done by simply choosing the songs with highest f-star value[16].

This model generalizes to new songs, but the user preference space is seen as one single space. This is a simplification of reality where a user preference space is probably divided into several categories, for example a workout preference space and a chill-out preference space, something the model provided does not take into account, which can be claimed as a weakness in terms of playlist context generation. Neither does the model take the ordering of songs into account.

Chapter 4

Representation Learning

Representation learning is about learning the factors that represent something. The idea behind representation learning is that the data used for a task often can be represented in a simpler way that is more suited for the task at hand[2]. This idea is nothing novel, even as far back as during the time of Greek rationalists and atomists the idea that observed data was dependent upon something latent was present[12]. Representation learning assumes that there is an intrinsic representation of data that is less complex than the observed representation. One example could be an image represented by pixels. If the image is of size 320 x 320 pixels it can be thought reasonable to assume that the image has an equivalent amount of degrees of freedom as pixels. However in an image of a man wearing a shirt every pixel is not independent as most shirt pixels are adjacent to other shirt pixels and thus not independent for example. As all number of pixels in an image do not vary independently of each other learning a representation of images also means that the number of factors learned are less than the number of pixels for each image in the data set. A lower number of learned factors than observed ones implies that a dimensionality reduction is made while learning a representation. Another explanation of learning representations is the one of Factor Analysis, which intends to describe how a number of observed variables vary together. One example could be points on a two dimensional plane in a three dimensional space. The points on the plane covary, but only in two dimensions. Therefore a dimensionality reduction can be made to describe the points in this plane, as they need not be described by three dimensions. However we cannot be sure that the points only vary in lets say the x or y direction. The directions in which the points vary must be learned and does not have to be well represented in the original dimensions of the vector space. These two learned dimensions of the plane then becomes the latent factors of the representation. Factor Analysis can be used in an exploratory way as a means of learning the underlying factors of something we want to represent, but it can also be used to synthetically generate data once the latent factors of the original data are learned.

4.1 Principal Component Analysis

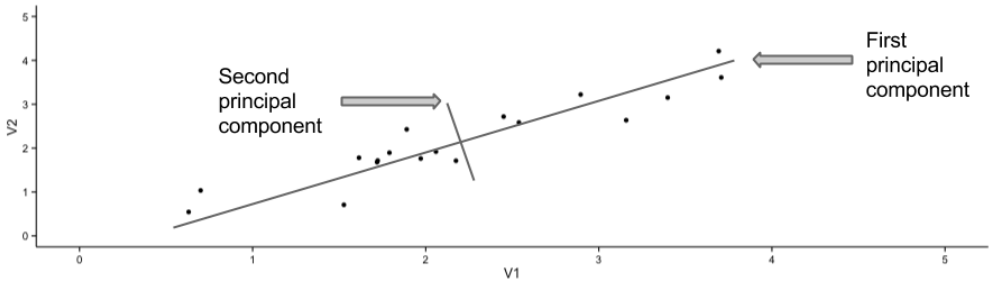
Learning a representation is about extracting latent factors from connected data. Connected data means that the data points in the data set examined are related to other data points in the same set. As there is covariance in the data latent factors can be learned. One way of deriving latent factors due to linear relationships in the data is Principal Component Analysis, PCA. What PCA can be said to do is to connect the covariance in a data set to a set of principal components that explain the variance in the data. To understand this connection lets look at the derivation of PCA:

$$C \in R^{DxD} X \in R^{NxD} C = (X-\mu)^T (X-\mu) argmin_A ||C - A||_F = \left\{ C = V\lambda V^T = \sum_{i=1}^{D} \lambda_i v_i v_i^T \right\} \tag{4.1}$$

$$= \left\{ A = \sum_{i=1}^{D} \gamma_i v_i v_i^T \right\} = || \sum_{i=1}^{D} \lambda_i v_i v_i^T - \sum_{i=1}^{D} \gamma_i v_i v_i^T ||_F = || \sum_{i=1}^{D} (\lambda_i - \gamma_i) v_i v_i^T ||_F \tag{4.2}$$

$$= || \sum_{i=1}^{d} (\lambda_i - \gamma_i) v_i v_i^T + \sum_{i=d}^{D} \lambda_i v_i v_i^T ||_F \tag{4.3}$$

Here X is our data and C is the covariance matrix of the data observed. A is an approximation of the covariance C and as can be seen from the derivation above. A approximates C up to a certain threshold of variance, if the full variance is covered then A will equal C, but if C only goes up to a certain threshold of variance a dimensionality reduction is made. In the general setting this means that a high threshold of variance, for example 90 percent, can be explained by $d < D$ dimensions. These dimensions that explain the major part of variance in the data can also be seen as the latent factors of the data. These factors are linear combinations of the original dimensions of data and are orthogonal to each other. Doing Principal Component Analysis of a data set is the same as doing an eigen decomposition of the covariance matrix of the data and selecting eigen vectors corresponding to eigen values up to a certain threshold. As PCA takes a covariance matrix as input and gives latent factors as output PCA can be said to connect the covariance of the data to a representation of the data.



As can be seen in the figure above, the first principal component lays in the direction that describes the largest part of variance in data. The principal components are orthogonal to each other.

4.2 Generalized vs Discriminative Models

4.3 OLOL

Playlists consist of songs and as a random sample of songs does not make as good a flow of music as a carefully mixed playlist one can conclude that there is a relation between the tracks in playlists. The tracks in a playlist, even though being correlated, vary somehow. If the tracks in a playlist would not vary, a playlist would consist of the same song over and over again. A statistical method to describe such a variation as the one in a playlist is called Factor Analysis[12]. In Factor Analysis observable and latent, or hidden, variables are discussed. Observable variables are the ones that we can see or observe.

Chapter 5

Computational Complexity

Computational complexity theory is about analyzing the amount of resources needed to solve a particular problem and classifying problems according to how difficult they are to solve. Resources needed to solve a problem generally mean running time or memory, but could also include randomness or communication. Analyzing the time it takes for an algorithm to solve a problem is also called time complexity. Time complexity is a quantification of the time needed to solve a problem as a function of the length or size of the input. A common way to describe time complexity is the asymptotic behaviour of running time needed as a function of input, also called *Big-O* notation. Time complexity analysis with *Big-O* notation takes the dominant terms into regard as input data grows towards infinity. For example if we have a problem that for the length of input n requires $(2n)^2$ operations the problem has a time complexity of $O(n^2)$. *Big-O* notation assumes that some operations called trivial operations take a constant time to perform, *O(1)*, and if n such operations are performed the time complexity becomes *O(n)*. All operations that do not grow with the size of input data are dropped using *Big-O* notation. For example if an algorithm needs a cubic number of operations for each input data term and an additional thousand operations that do not change with the size of input data the total amount of operations needed is $n^3 + 1000$ which becomes $O(n^3)$

Part II

Method

Chapter 6

Methodology

6.1 Problem Outline

The problem this thesis is trying to solve is to select a number of songs given a predefined playlist so that the selected songs constitute a playlist similar to the predefined one.

6.2 Assumptions

> *Without assumptions you cannot do machine learning*
>
> Ryan Adams

The goal of the thesis is to generate playlists, similar to seed playlists choosen by the user. In order to do this there is a need for assumptions regarding playlists.

The first assumption for this thesis is that curated playlists, playlists made by professionals whose work is to create good playlists, suited for a specific context are suitable training data to create a model that generates playlists suited to the same playlist context.

The second assumption made is that features that belong to each track in a curated playlist contain enough information to create a representation of the context this curated playlist is made for.

The third assumption is that a playlist can be looked at as a good mixture of songs, i.e. there is an inherit variance in the playlist that defines it. This is a clear distinction from assuming that playlists only consist of songs similar to each other.

6.3 Data

From the Spotify hadoop cluster all available playlists where extracted and then filtered based upon whether they were created by Spotify playlist curators or not.

Once playlists were filtered, feature data consisting of discrete values for genre, mood and tempo were added to each track within the selected subset of playlists.

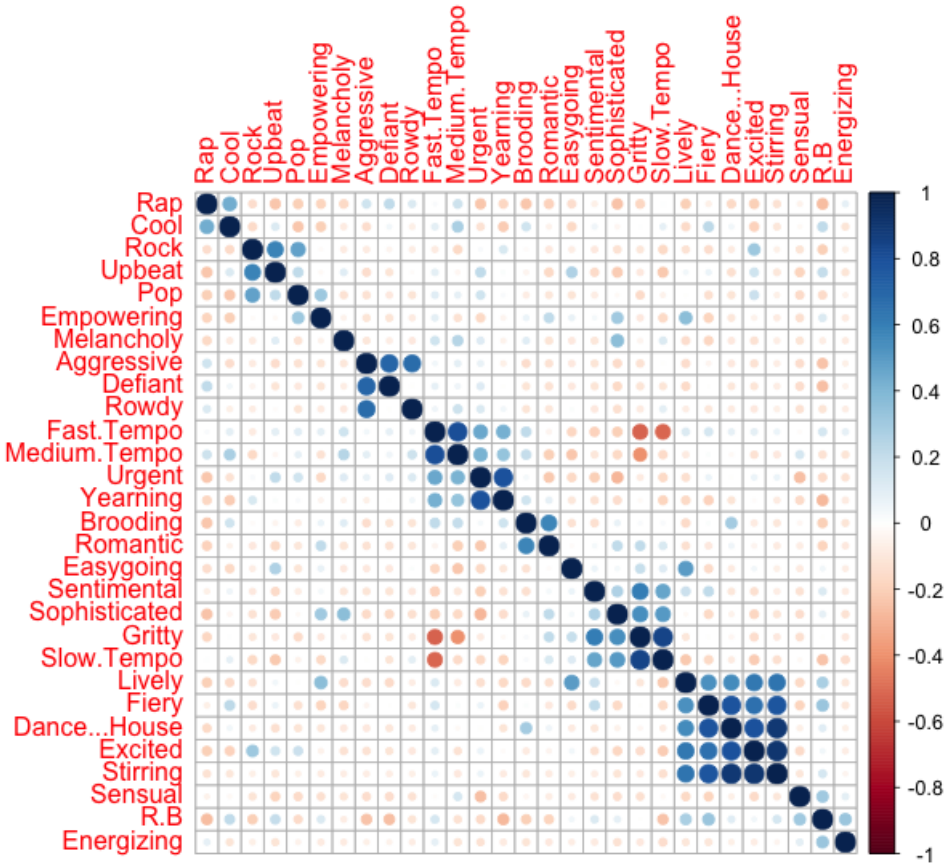
6.4 Pre-processing

The selected data did contain track duplicates within playlists. These were removed as they would otherwise affect each playlistÂ´s covariance of features.

6.5 Exploratory Data Analysis

To get an overview of whether features of tracks in a curated playlist relate to each other within the playlist correlation exploratory data analysis was made through plotting. More specifically correlation plots on a playlist level were made. The idea behind plotting correlations instead of covariances is that the magnitude of the correlation shows the strength of the linear relationship between features, while a covariance plot would be polluted should different features be on different ranges. By plotting correlations the problem of calculating correlations for features with zero variance, given a playlist context, emerged. Zero variance terms are a problem in the correlation setting as calculating the correlation for a zero variance term would imply dividing by zero, a mathematically undefined operation. This problem was solved by setting the correlation for feature relations with zero covariance to zero. It can be argued whether this is mathematically correct or not. But the approach can be motivated in this setting by the fact that plots are done to get an intuition of the data. A correlation of zero for features with zero covariance thus gives a better intuition of relationships in the data set compared to setting the correlation to one.

Performing plots of feature covariances of a playlist also shows whether there are linear realtionships among features in that playlist or not.



As we can see from the example plot above, there are clearly linear correlations among features for our example playlist.

6.6 Learning Playlist Characteristics

Once that linear relationships have been spotted in the data the next step is to create a model that can learn the representation of a specific playlist. One simple approach to learning latent factors in data is principle components analysis, PCA. Explaining the characteristics for a certain playlist context could be seen as equivalent of explaining the variance of features for tracks, given a curated playlist suited to the specific playlist context. Therefore extracting the main characteristics for a playlist context can be done by extracting the principal components, for the curated playlist representing that playlist context. It is reasonable to assume that the data that is modelled by PCA is not noise free, why PCA is performed up to a certain threshold for the variance explained and thus a dimensionality reduction is made. Using this approach extracting eigenvectors for the covariance matrix, rather than correlation matrix, is a motivated choice. The motivation behind this

choice is that scaling the covariance matrix to a correlation matrix is a nonlinear transformation. If we want to use apply the principal components of a correlation matrix to the original data, then the original data need to undergo the same transform as transforming covariances to correlations. For a data set where each curated playlist makes up less than one percent of the total data it would be impractical to transform the original data over and over as we extract the principal components for each playlist context. Doing so would also not be feasible in terms of scalability. Using the covariance matrix for extraction of features is therefore motivated as the principal components of the covariance matrix can be directly related to the existing data.

6.7 Handling zero variance terms

Even though there are no zero variance terms in the whole data set, there are some terms that have zero variance within a certain curated playlists. These terms will not be handled by the principal components describing a playlist context, as principal components describe the variance of a playlist. Despite not being handled by the principal components zero variance terms might still have an important role in describing a playlist context. For example, if we have a curated jazz playlist then it is probably an important factor that all of the tracks in this playlist have a zero value for rap. The importance of this can be easily understood by imagining the opposite, what if those tracks would have a constant non-zero value for rap? Then a non-zero value for rap associated with jazz would be an important indicator for that playlist, why the absence must also be an important indicator.

6.8 Selecting candidate songs for a playlist context

The process of selecting candidate songs given a specific playlist context is an interesting and ambiguous problem without a given approach. Earlier work is focused mainly on item to item recommendation, i.e. recommending similar items of the same type given preferences for items of a certain times. But when it comes to selecting appropriate songs for a playlist context the items are of different kinds. The goal is to recommend songs, one type of item, given a playlist describing a playlist context, which is another type of item.

One initial idea to select songs for a given playlist context could be that songs are either good candidates or not. This is a reasonable assumption, as for example for a rock classics playlist context then songs are either rock classics or not. Given that this is a binary classification problem, an efficient two class classifier might seem as a good idea at a glance. A support vector machine, SVM, is an optimal two class classifier by definition, as a SVM maximizes the margin between classes[3], and has the capability of multi class classification with for example the one versus all approach[7]. There is however one problem with support vector machines, or any classifier that requires training data within contextual playlist generation. The

problem is that it is easy to define training data which labels a song as belonging or not belonging to a certain playlist. But it is hard to define what songs that belong to other playlists, than the one describing a specific playlist context, which are still relevant for that playlist context. For example a song belonging to a house Workout playlist may very well be a suitable candidate for a house party playlist. It is actually often the case that many songs belong to several playlists, describing different playlist contexts. Given this example a discriminative model turns out to be a bad fit for the problem this thesis is trying to solve. If a song belongs to a house party playlist then it is reasonable to assume that it would be outside the margin defining a house workout playlist if feeded to a SVM, even though this particular song might very well be a suitable match for the house workout playlist. This rules out the use of SVMs for the purpose of this thesis, as SVMs need to know the mapping between songs and playlist contexts to work. The same mapping that we are trying to find.

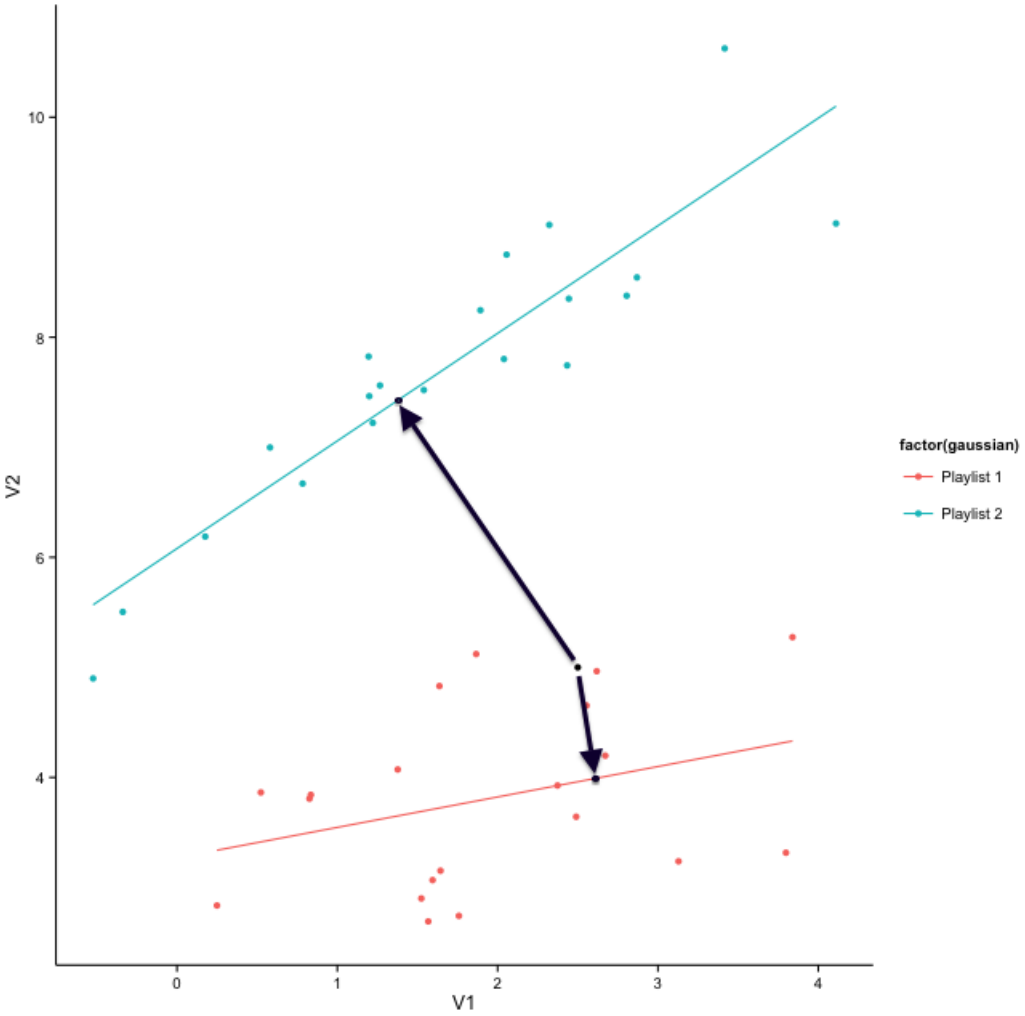
A second idea to selecting songs suitable for a specific playlist context would be to use centroid based clustering. The wikipedia definition of clustering is as follows: "clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense or another) to each other than to those in other groups (clusters)". One could for example cluster all tracks in curated playlists and then simply assign each song that is not part of a curated playlist to the cluster providing the best fit for each track. But one problem is that there is not a one to one mapping between tracks and playlists, one playlist can contain many tracks and one track can belong to many playlists. This is different from clustering where each cluster consists of many points, but each point only belongs to one cluster, which yields centroid based clustering impropriate for the scope of this thesis.

A third approach to finding candidate songs given a playlist context would be to tweak the normal usage of collaborative filtering. The common approach of collaborative filtering is to use a sparse matrix to infer the rating of items for one user given the ratings of similar users. What can be done instead is to use binary ratings and instead of inferring ratings for a user one could infer ratings for songs given a playlist. What this means is that playlists that contain the same songs as a playlist describing a playlist context one is interested in will be used to infer songs that are good matches for the specified playlist context.

6.8.1 Subspace method

A last approach for track candidate selection would be to use the subspace method. Given that the principal components for a playlist, describing the variance of that playlist suited for a playlist context, are at hand, one can simply treat each track as a vector rather than a point. Each vector can then be projected into the principal component space for that playlist context. The underlying assumption is then that points that have a low relative change in magnitude under projection are well described by the characteristics defining the playlist context, and thus good candi-

dates. Tracks that are not well described by the playlist context characteristics on the other hand, will change under projection and will therefore also have a high relative change in magnitude. As all tracks will be projected this is actually a ranking algorithm where tracks with lower relative change in magnitude will have a higher rank and tracks with higher change in magnitude a lower rank. To illustrate how the subspace method works lets look at the following picture:



Here two playlists are illustrated in the two dimensional plane and their first principle component is the line through the points. As can be seen projecting the point into the principal component space of plaaylist 1 means a lower change of magnitude for the point than projecting the point into the principal component space of playlist 2. This means that the point, representing a track, provides a better fit for playlist 1 than playlist 2.

There are however problems with the subspace method. Lets say that there is a playlist context that is defined by variance in the dimensions jazz, blues and rap

and our vector space consists of the dimensions jazz, blues, rap and rock. If there is a song that is characterized by jazz and blues only, then this song will go unchanged under projection. As the relative change in magnitude is none then this song will be suggested as a suitable candidate for the jazz, blues, rap playlist context. However a playlist context consisting of jazz, blues and rap is likely to be pretty peculiar and a song characterized by jazz and blues only is not likely to be a suitable match for such a playlist. Another problem would be songs that consists of zero values for all features, these songs would also go unchanged under any playlist context projection, but are not likely to be good candidates for all playlist contexts. Also if by looking at the definition of covariance

$$\sigma(X, Y) = E[(X - E[X])(Y - E[Y])^T] \tag{6.1}$$

one sees that covariance only takes the relative difference into consideration. That is if the variance of a variable is on the scale of five to twenty or from twenty to thirty five the direction of variance when applying the Principal Component Analysis will be the same. Therefore one can say that the subspace method only takes direction but not location into consideration.

Further, the subspace method is a linear transformation and it can be questioned if a linear transformation is powerful enough to describe the necessary mappings.

6.9 First Track Manifold

After studying linear relationships among features within playlists the study was extended to see if there were linear relationships among the ordering of songs as well. To do this the subspace method was used again. But instead of extracting the principal components for the variance of a playlist, the principal components describing the variance of all the start songs of all curated playlists were extracted. Once this was made a sample of songs from the curated playlist data set were projected into the first track manifold space created by the principal components of the first tracks. This was made to see if start tracks would have higher ranking, i.e. lower relative change in magnitude, than other tracks.

6.10 Playlist Comparison

As principal components were chosen to describe playlists, it is reasonable to assume that if principal components analysis works well for describing playlist characteristics, then the same approach should also work well for comparing playlists. Playlists were compared pairwise. To compare two playlists all eigenvectors from each playlist were multiplied by each other. If we think of playlist A as

not really cosine similarity, not normalized

$$A = U_1 \lambda_1 U_1^T$$

and of antother playlist B as

$$B = U_2 \lambda_2 U_2^T$$

then the operation performed to compare them can be seen as

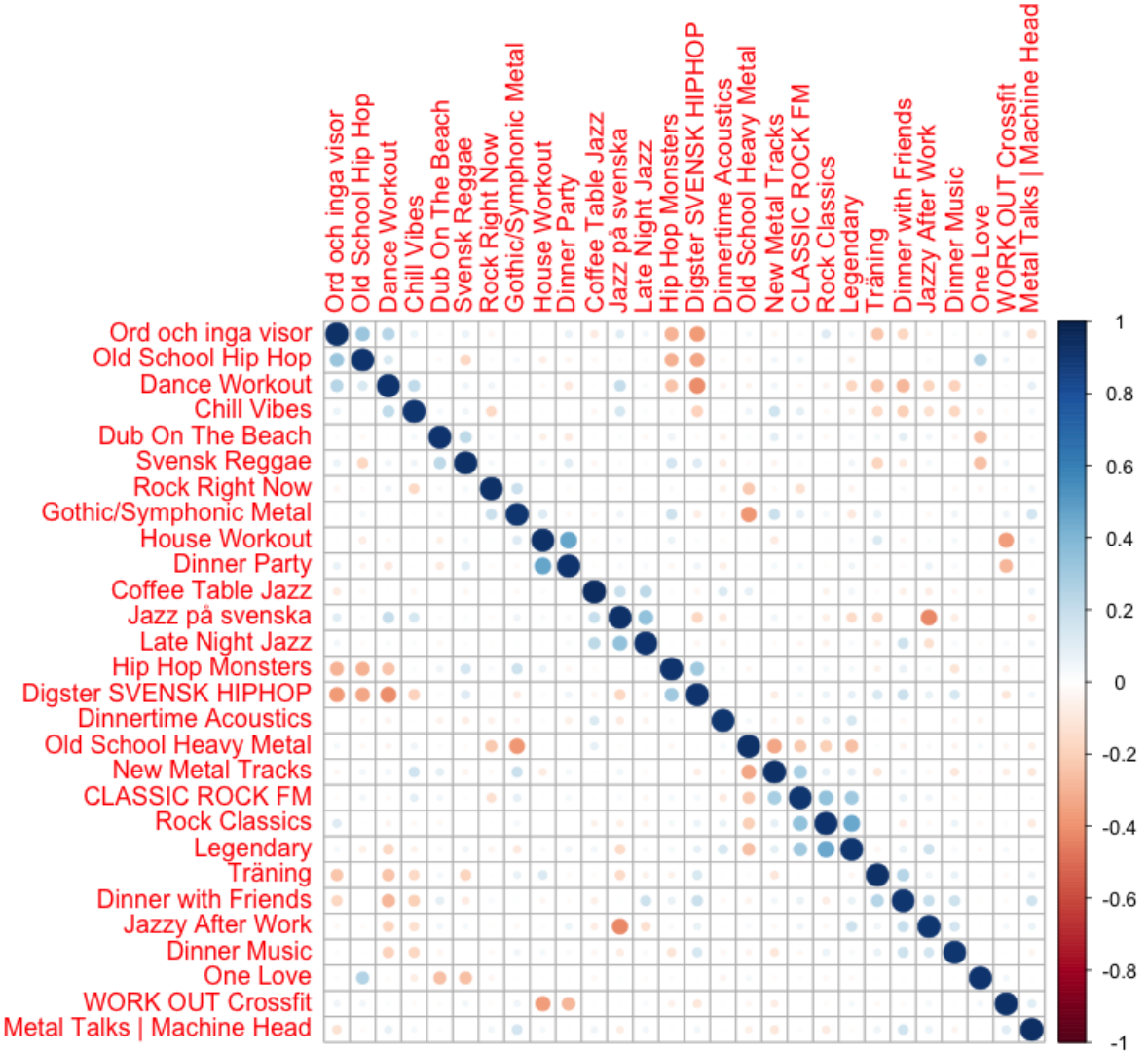
$$M = U_1 U_2^T$$

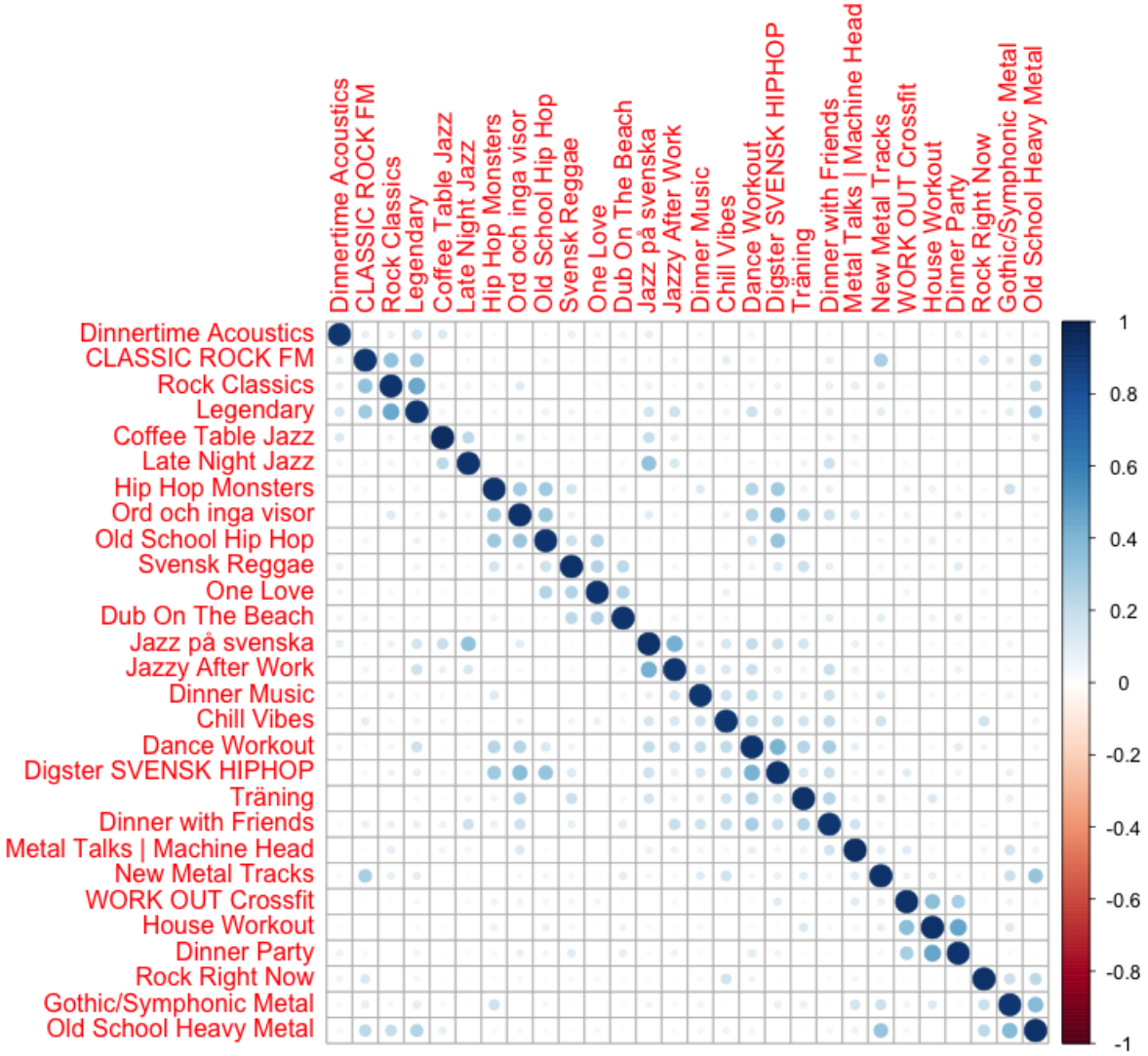
where M is the resulting matrix from the comparison. By doing this the cosine measure of vector similarity for each pair of vectors was obtained.

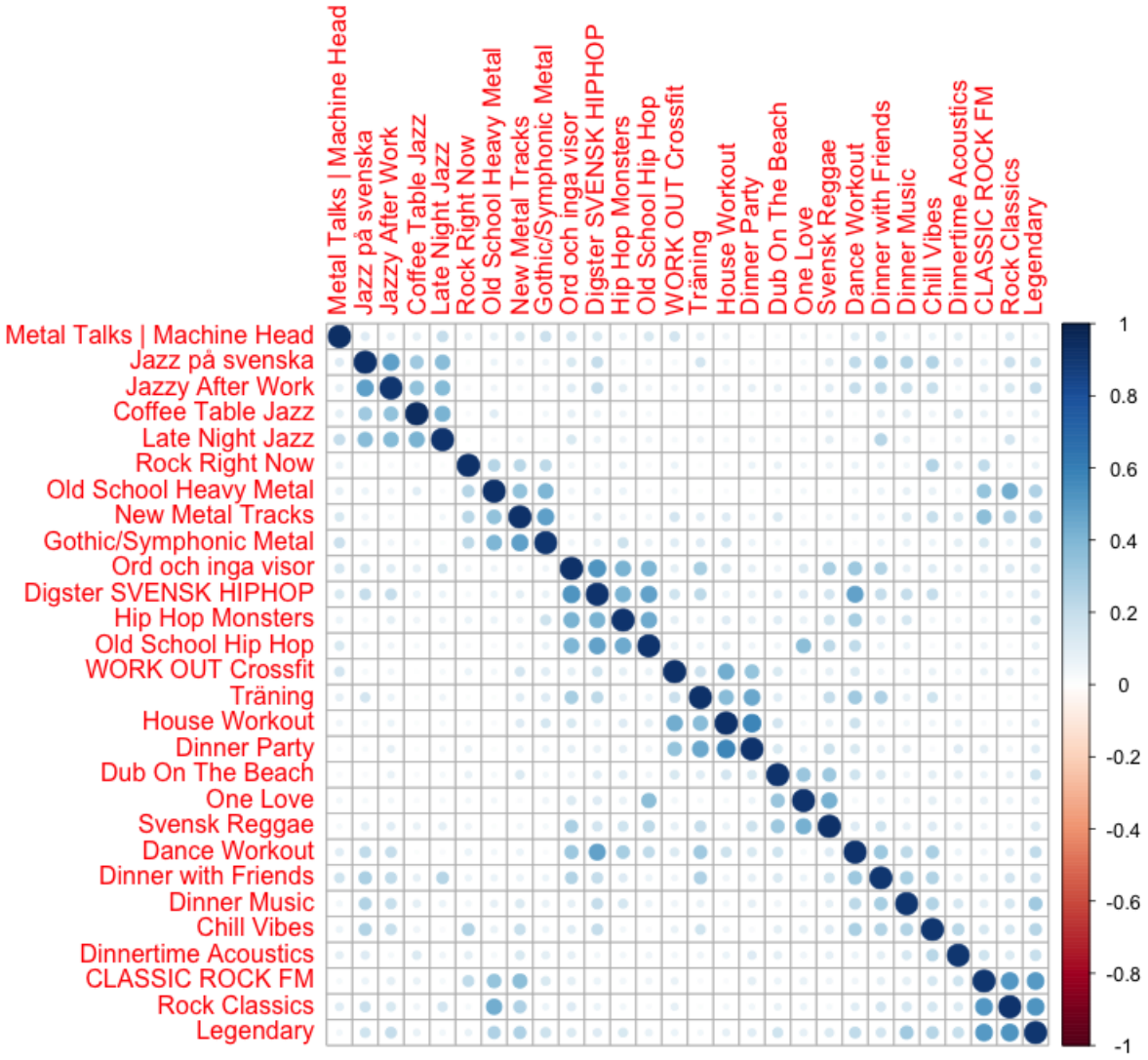
The problem with this approach is that it gives an unbalanced comparison. By simply looking at the similarity of eigenvectors implies that eigenvectors corresponding to low eigenvalues have the same importance as eigenvectors corresponding to high eigenvalues. This means that components explaining a high part of the characteristics of a playlist are regarded an equal importance as components explaining a low part of playlist characteristics. To remedy this problem the cosine score between eigenvectors from each playlist was scaled by the square root of the product of the corresponding eigenvalues, which mathematically can be formulated as

$$M = U_1 \sqrt{\lambda_1} \sqrt{\lambda_2} U_2^T$$

The result obtained from this multiplication was a square matrix. To rank the similarity between these matrices some type of transformation from a matrix to a single value is needed. The initial idea was to aggregate the entries of the matrix, but an aggregated value does not tell which values that have led up to the aggregated result. Therefore by simple adding values of the matrix one would not know if the similarity comes from dot products between vectors corresponding to high or low eigenvalues. Therefore it was choosen to only take the trace of the matrix into consideration. Three different approaches were taken: the sum of the values of the diagonal, the absolute value of the sum of the diagonal and the sum of the absolute values of the matrix diagonal. These approaches were taken to compare 28 playlists among themselves.


Ord och inga visor
Old School Hip Hop
Dance Workout
Chill Vibes
Dub On The Beach
Svensk Reggae
Rock Right Now
Gothic/Symphonic Metal
House Workout
Dinner Party
Coffee Table Jazz
Jazz på svenska
Late Night Jazz
Hip Hop Monsters
Digster SVENSK HIPHOP
Dinnertime Acoustics
Old School Heavy Metal
New Metal Tracks
CLASSIC ROCK FM
Rock Classics
Legendary
Träning
Dinner with Friends
Jazzy After Work
Dinner Music
One Love
WORK OUT Crossfit
Metal Talks | Machine Head
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1


Dinnertime Acoustics
CLASSIC ROCK FM
Rock Classics
Legendary
Coffee Table Jazz
Late Night Jazz
Hip Hop Monsters
Ord och inga visor
Old School Hip Hop
Svensk Reggae
One Love
Dub On The Beach
Jazz på svenska
Jazzy After Work
Dinner Music
Chill Vibes
Dance Workout
Digster SVENSK HIPHOP
Träning
Dinner with Friends
Metal Talks | Machine Head
New Metal Tracks
WORK OUT Crossfit
House Workout
Dinner Party
Rock Right Now
Gothic/Symphonic Metal
Old School Heavy Metal
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1



As can be seen summing the absolute values of the matrix diagonal provides a good clustering of similar playlists in the data set used. From a theoretical perspective summing the absolute values is also the method that makes the most sense. Eigenvectors from the covariance matrix explains the direction of variance, but it does not really matter if the variance is seen as going from A to B or from B to A. Hence the direction of eigenvectors from a playlist feature covariance matrix does not really matter. If the direction of eigenvectors do not matter then the resulting values of the matrix diagonal, provided by multiplying the weighted eigenvectors from two different playlist covariance matrices, do not matter either and it makes sense to use the sum of absolute values as an aggregated measure is used.

6.11 Approximate Nearest Neighbours

OL

Part III

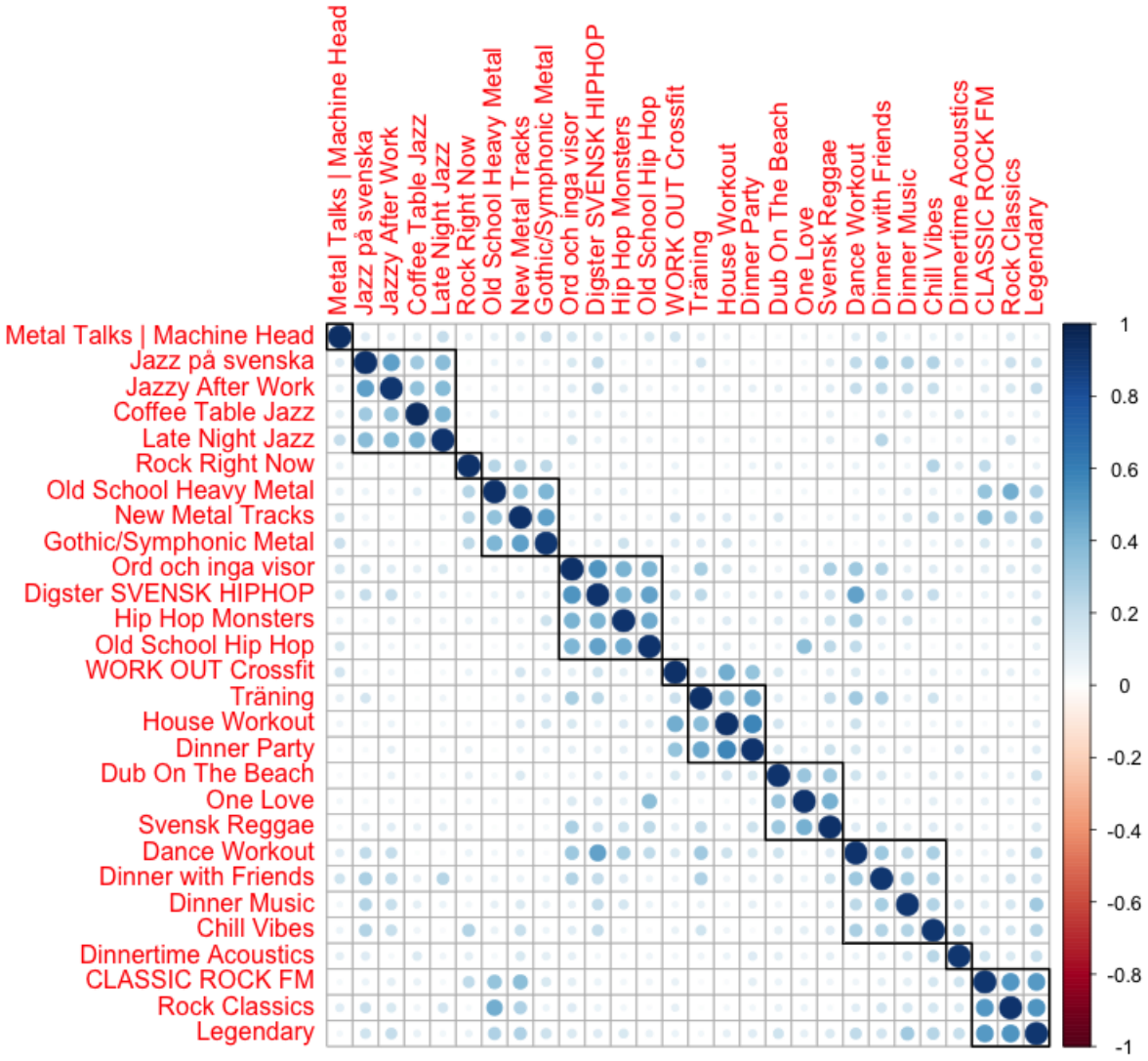
Evaluation

Chapter 7

Results

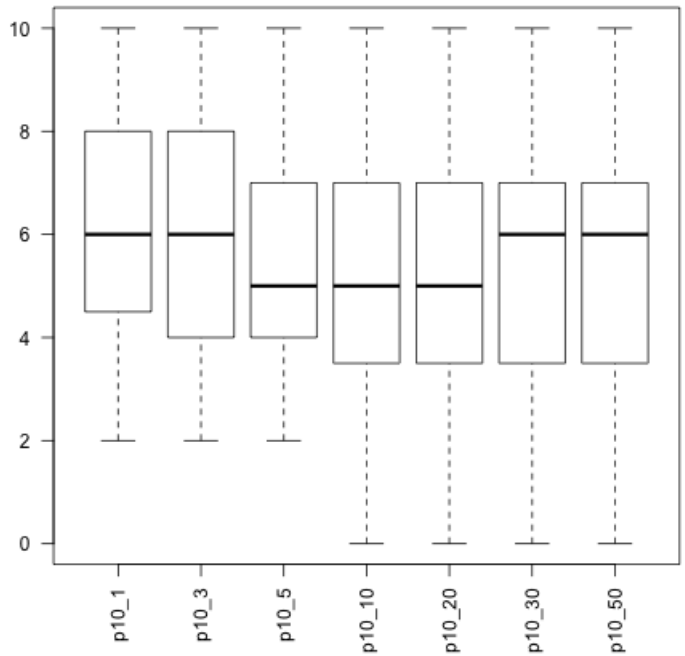
7.1 Precision

Evaluating a generated playlist is a difficult task as there is no known ground truth for what a good playlist is. Even when asking normal persons or domain experts opinions about what consists a good playlist are likely to differ. There are of course sanctioned methods for evaluating data when features are present, using the cosine distance as an evaluation metric is one such approach. However using such as metric presents new problems. If the cosine distance, for example, is used for evaluation this would imply that the cosine distance also should also be used for the model. But if the evaluation metric and the objective function one tries to minimize are the same is there really an actual evaluation framework present or is the entire evaluation pipeline simply a tautology? Imagining the opposite is not compelling either. If one objective function is minimized by the model and another is used for evaluation one finds oneself in a situation of comparing apples and oranges, which is unlikely to be desired. Finding a way to evaluate generated playlists is without doubt a difficult task, but some metric is still needed as a proxy. The first evaluation approach choosen for evaluating generated playlists finds its roots in the method of describing playlist similarity described in the methods section.

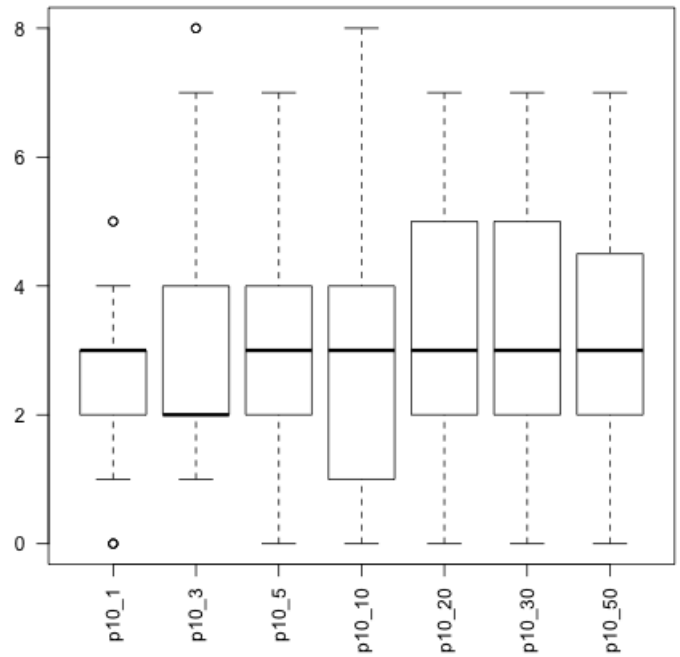


Based on the method of comparing playlists by taking the dot product of the principal components of two playlists, weighting by eigen values and transforming the resulting matrix into a value by summing the absolute values of the trace of the matrix an additional hierarchical clustering step can be added to create meaningful playlists clusters, as seen in the picture above. As these clusters make up a sensible segmentation of playlists they where used as a base for evaluation. When the subspace method was applied to rank candidate songs for a seed playlist all songs that originates from a playlist within the same cluster as the seed playlist were considered true positives. All other songs were considered false positives. For the evaluation task a subset of data was used were only the tracks of the seven clusters consisting of more than one playlist from the figure above were used and

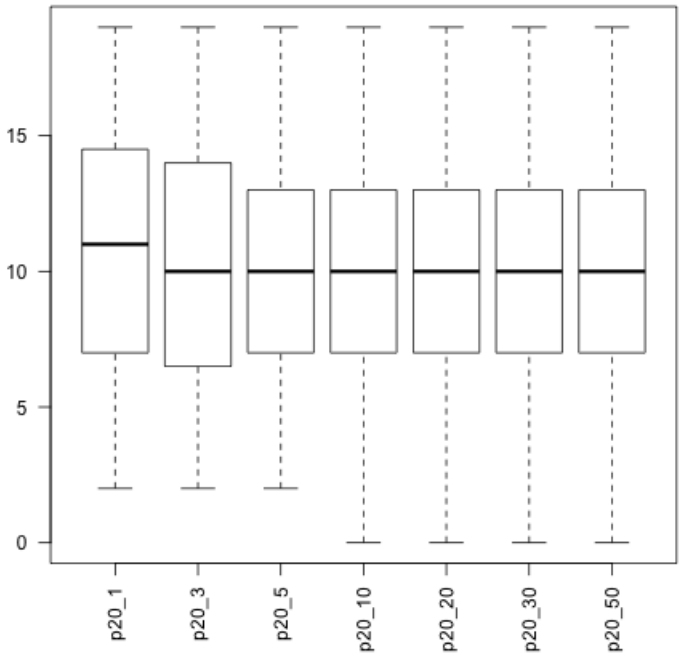
pre-filtering was made by using approximate nearest neighbours with an euclidean distance measure. The approximate twenty nearest neighbours were used and the number of trees in each approximate nearest neighbour forest was varied. This is a non-optimal way of using approximate nearest neighbours as the proper way would mean to select a by magnitude larger number of neighbours than actually needed due to the fact that the method is an approximate. The reason for not doing doing so is that as small data set was used for evaluation using the two-hundred nearest neighbours would mean that the pre-filtering step would loose its effect as almost the entire data set would pass through pre-filtering. Precision was calculated for the ten, twenty and thirty top ranked songs. Calculating precision this way is a conservative measure as songs from playlists outside the cluster also might be good candidates. To give a reference to how the model performs a random sample of songs after the approximate nearest neighbour step was also used were precision was calculated the same way as for the model.



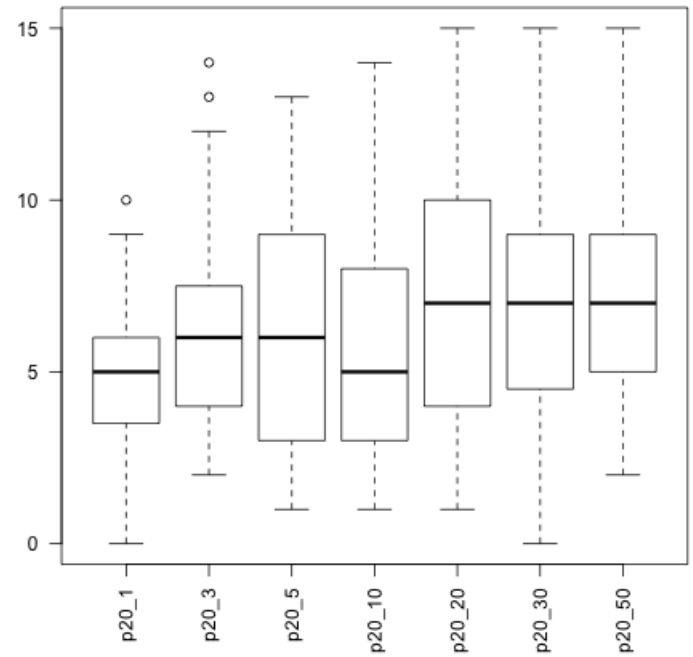
Model p@10



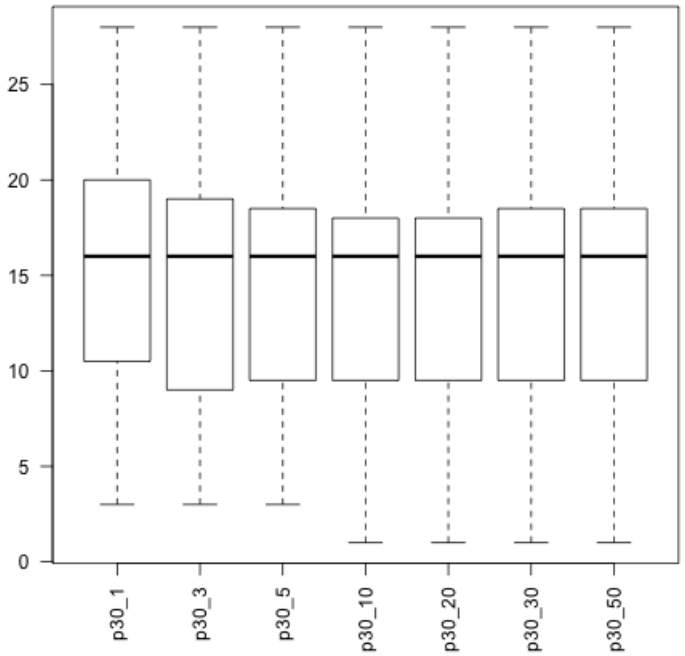
Random p@10



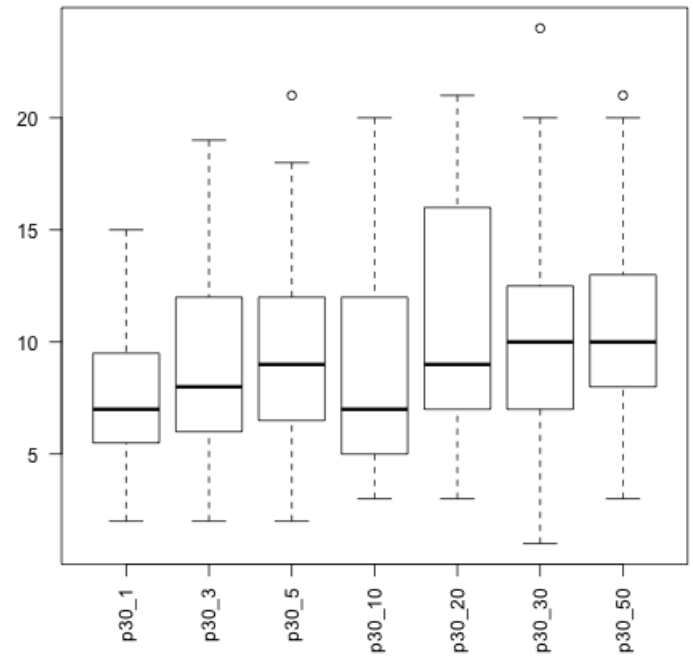
Model p@20



Random p@20



Model p@30



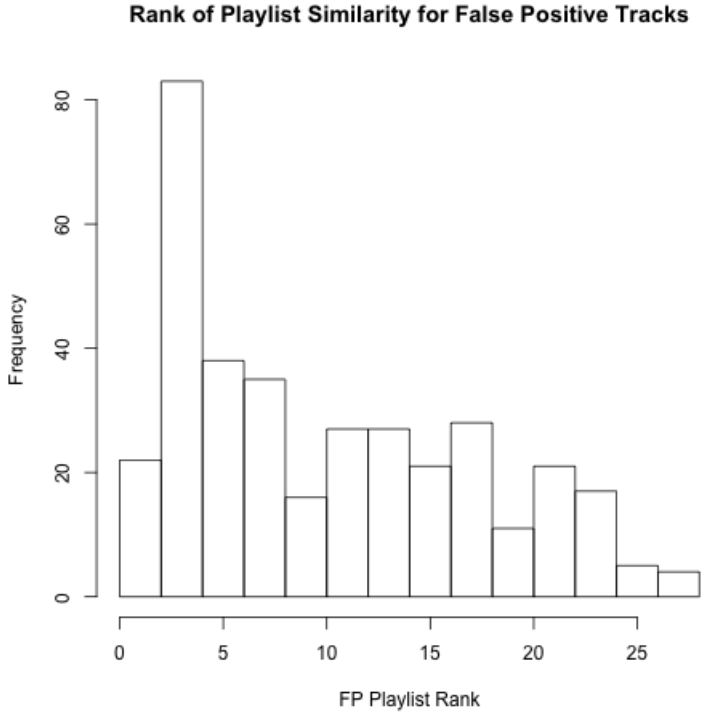
Random p@30

As can be seen the model outperforms the baseline in many cases by 100 percent higher precision and never performs less than 40 percent better than the baseline.

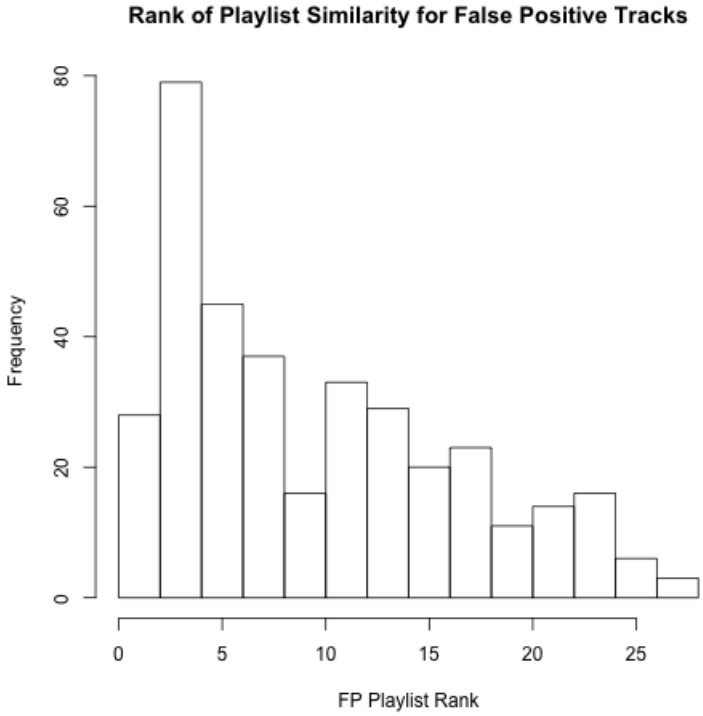
7.2 Confusions

As can be seen in the image showing the playlist clusters there is often a high correlation between a playlist inside a cluster and another playlist outside the cluster, as for example between the playlists *Digster SVENSK HIPHOP* and *Dance Workout*. Using precision by only considering tracks from playlists inside the clusters as true positives is therefore a conservative measure and the actual results might therefore be better than what the precision results entail. To quantify and investigate to what extent false postives come from closely playlists histograms of the rank of playlists from which the false postive tracks originated were made. Here the rank is the rank of how highly ranked the playlist the false positive sample originated from was related to the seed playlist for the entire data set.

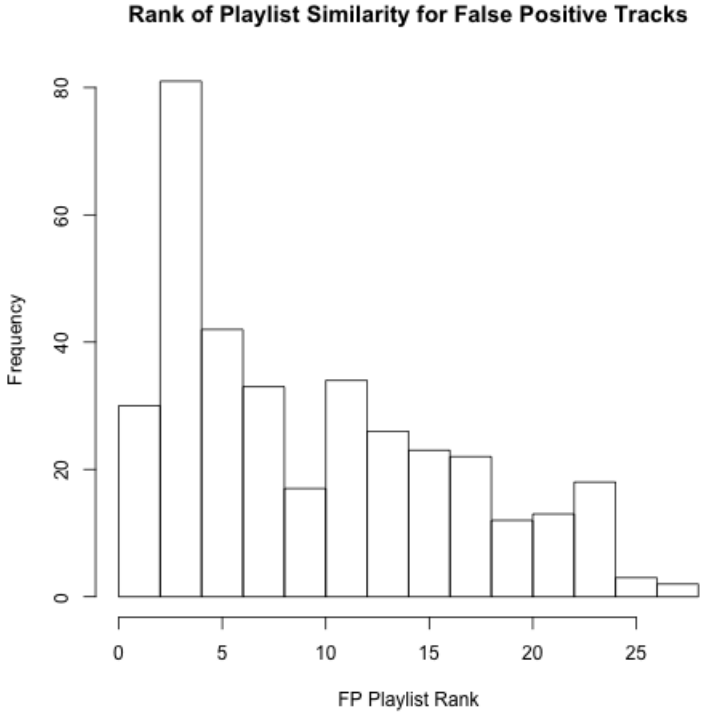
Histograms were made for all number of trees for the approximate nearest neighbour pre-filtering step.



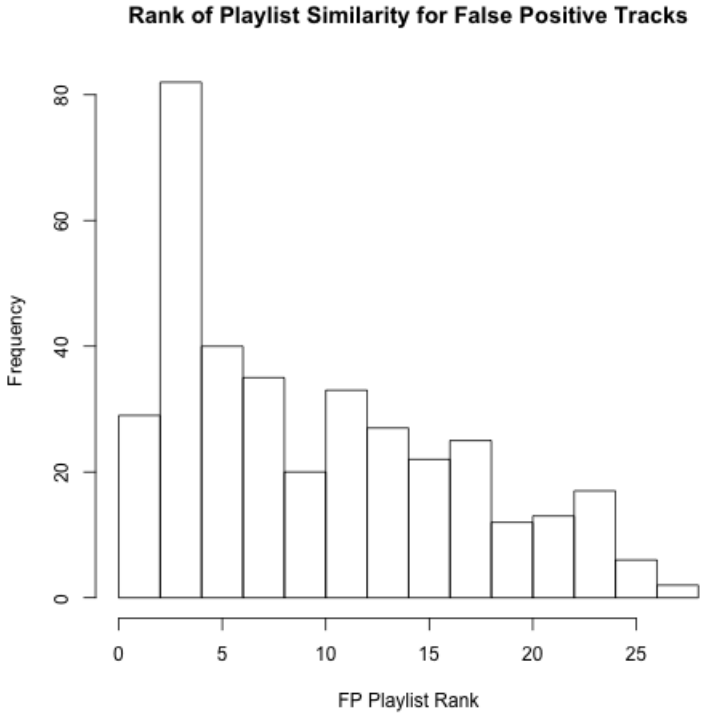
1 tree



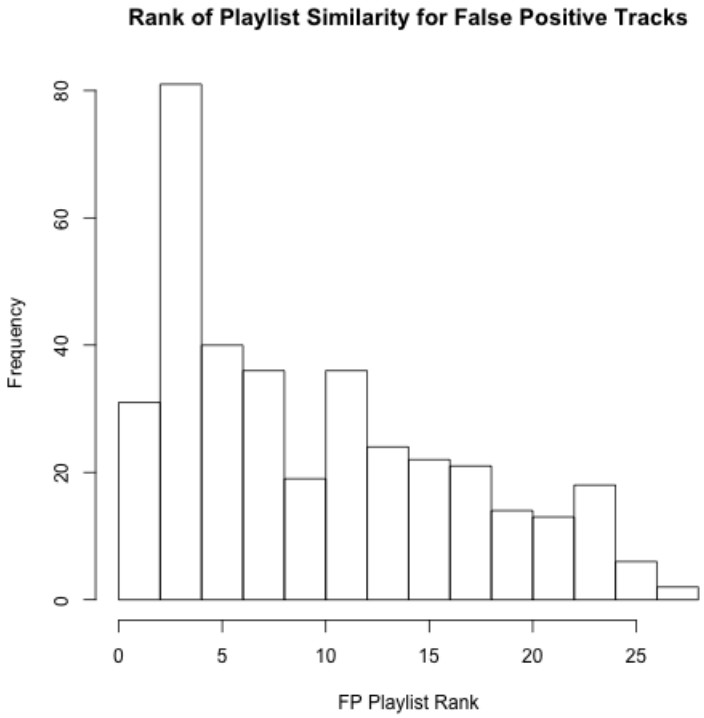
3 trees



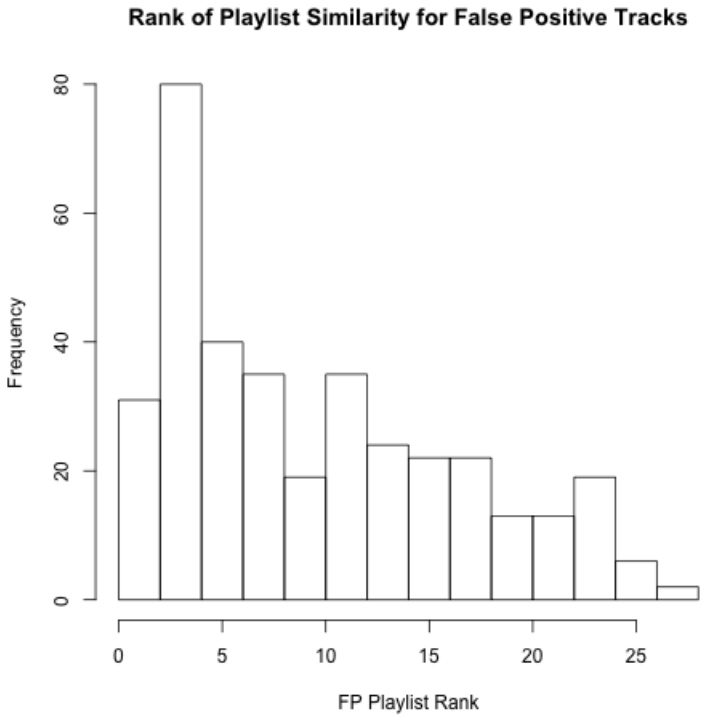
5 trees



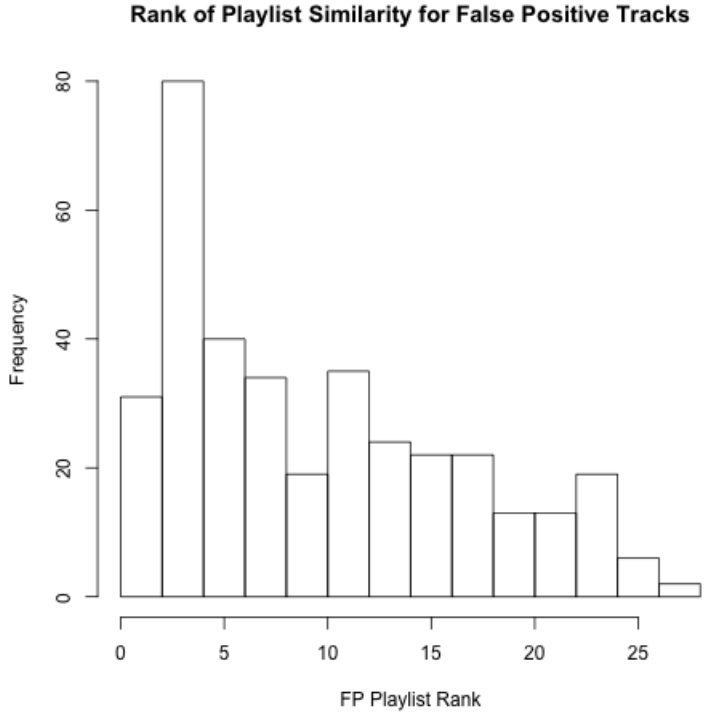
10 trees



20 trees



30 trees



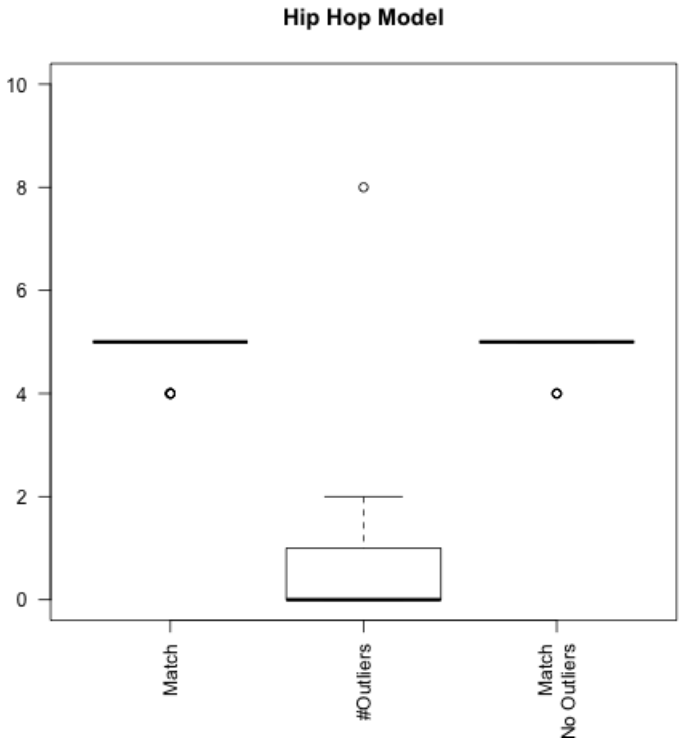
50 trees

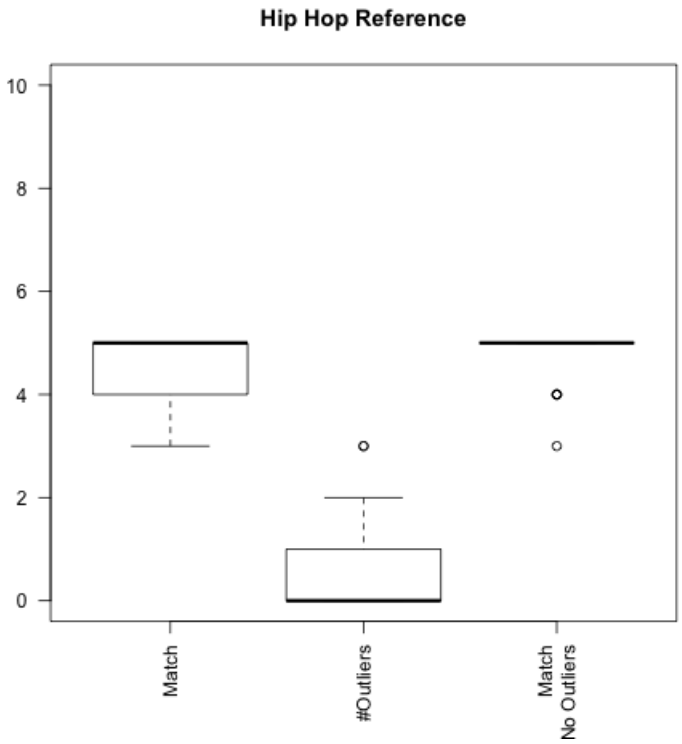
There is a similar behaviour for all number of trees in the approximate nearest neighbour forest where roughly 40 percent of all false positives come from the five playlists most similar to the seed playlist.

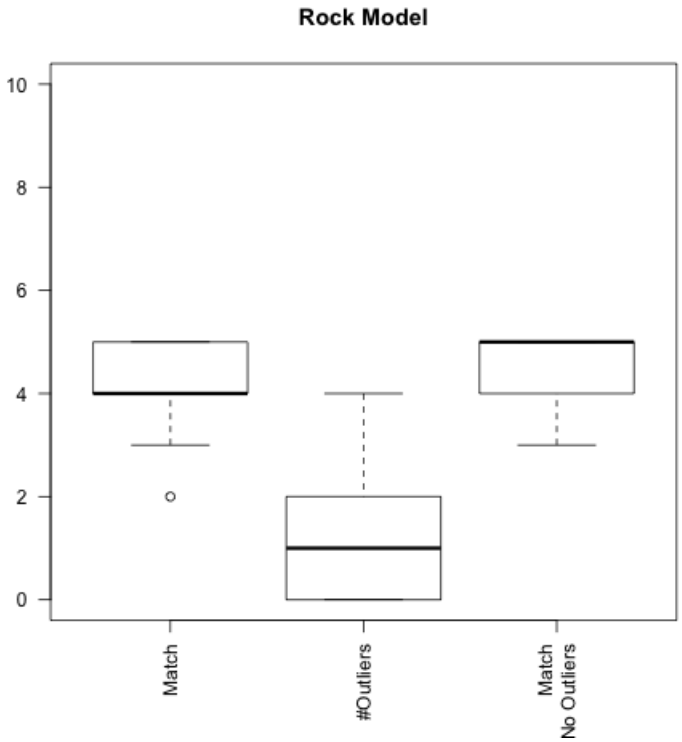
7.3 Qualitative Evaluations

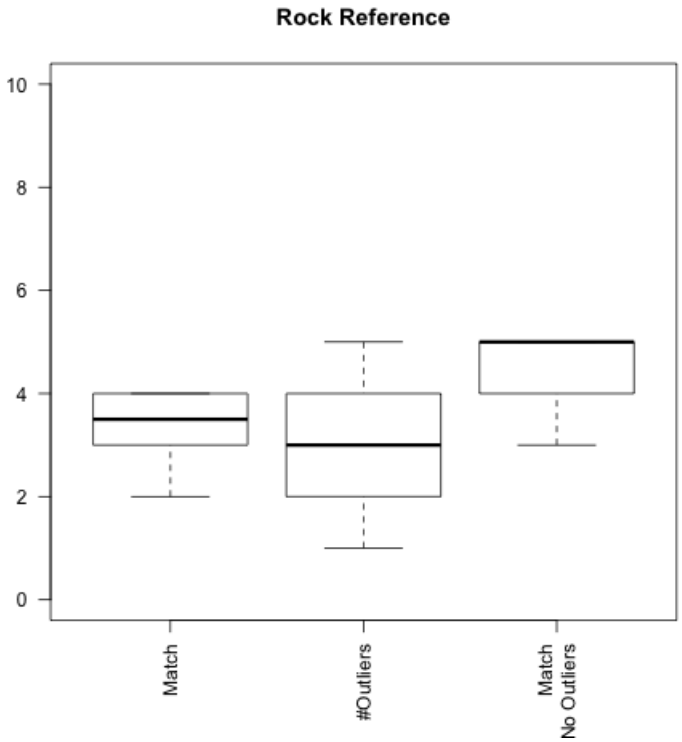
To find out if the confused false positives really meant a higher amount of good candidate songs and to get an idea of how the selected candidate songs sounded to the human ear qualitative evaluations were performed. Three playlists for which the seed playlists Old School Hip Hop, Old School Rock and Old School Heavy were selected. The reason for selecting these playlists are that they contributed to uniformity of the test as they all were *"Old School"* and all were genre playlists. Also Hip Hop was seen to precision a high performing playlist, Rock middle performing and Metal low performing so to see how well the performance of the quantitative measures responded to listeners opinions. To have a reference point both playlists with songs selected by the model as well as songs from the original curated playlists were selected. To haven an even distribution of biases from the persons listening to the playlists new playlists were created where ten songs in order came from the top ranked candidate songs by the model and the other ten songs, also in order, were the first ones in the seed playlist. Users were then presented with these three playlists and were asked how well the first or last ten songs matched the theme, how many outliers there were and how well the songs matched the theme with outliers

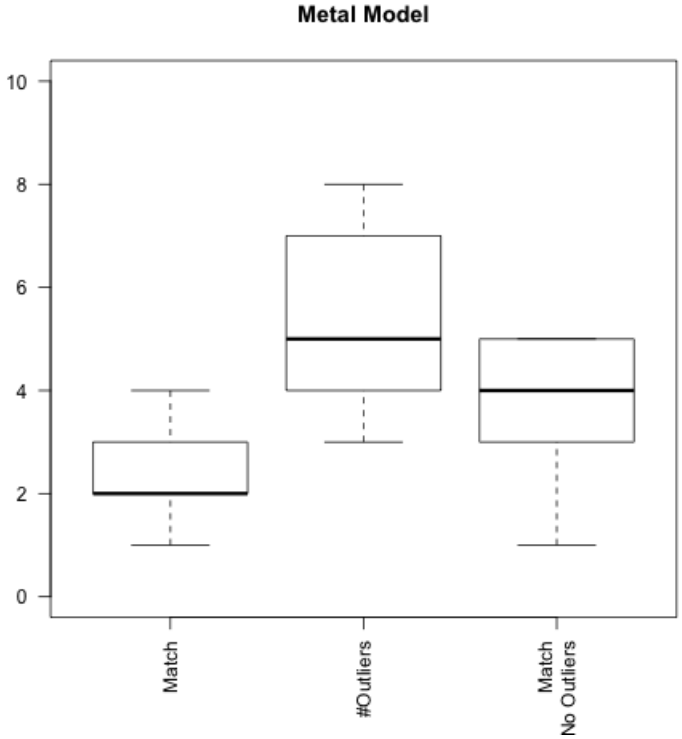
removed.

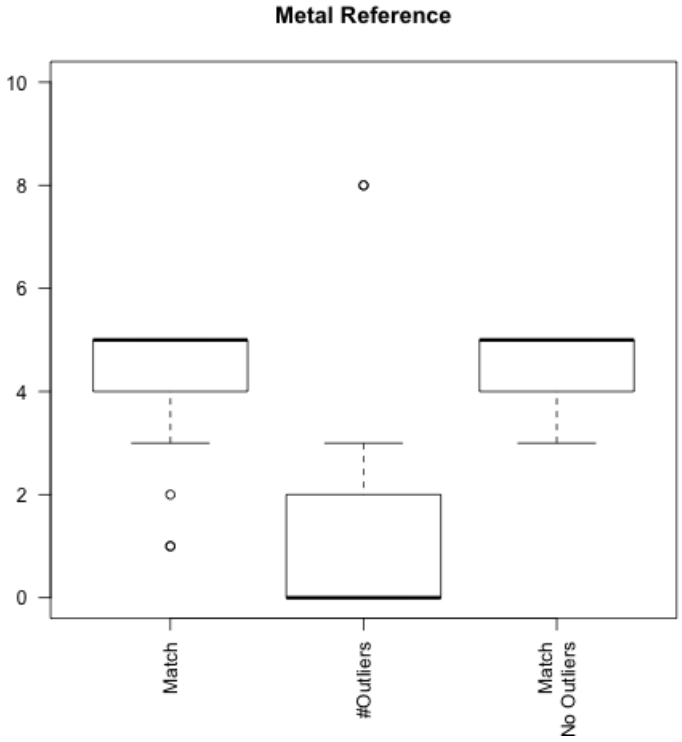

Hip Hop Model
10
8
6
4
2
0
Match
#Outliers
Match
No Outliers


Hip Hop Reference
10
8
6
4
2
0
Match
#Outliers
Match
No Outliers


Rock Model
10
8
6
4
2
0
Match
#Outliers
Match
No Outliers


Rock Reference
10
8
6
4
2
0
Match
#Outliers
Match
No Outliers





As can be seen form the user evaluations surprisingly the tracks from the model

were ranked higher than those taken from the curated playlists for both Hip Hop and Rock. The model selected tracks for metal were ranked lower, but the users rankings confirmed the notion that precision as used for the quantitative evaluations was a conservative measure as users found less outliers than there were false positives from the quantitative evaluation.

Chapter 8

Computational Complexity Analysis

One reason to why time complexity is important in the field of Machine Learning is that it gives a unified measure of the time and resources needed to solve a problem. One example is if the run-time for an algorithm is asked a not uncommon answer is that it takes x time. An answer as such gives no information of how the algorithm scales, whether it is implemented in an efficient manner or not or if the run-time is hardware dependent. For example, an algorithm that take one day to solve a problem, but has linear time complexity can solve the same problem for the double amount of data in two days. For an algorithm with quadratic running time it would take four days to solve the same problem should input data be doubled. Another example is if an algorithm is run on a brand new computer or a computer that is 15 years old, simply stating the run-time does not give a unified measure as run time will change depending on hardware. Lastly, given the size of input data and the time complexity of the algorithm a hint of the expected run-time can be obtained and used for debugging purposes. As can be easily understood analyzing time complexity within the machine learning field is essential to understand the scalability of a machine learning algorithm. An algorithm that grows faster than linear with data quickly becomes unfeasible as the amount of data used grows large. In the case of Spotify the number of tracks in the song library and the number of users are in the magnitude of millions or tens of millions which means that even a linear run-time might be too slow to be practically usable.

8.1 Time Complexity of Model

The first step in the model is to create a covariance matrix from the features of tracks in the seed playlist. The computational complexity of creating a covariance matrix is $O(ND^2)$ where N is the number of tracks in the playlist and D is the number of features that represent each track[9]. Once the covariance matrix is created an eigen value decomposition is needed to get an orthogonal representation of features in the data, the time complexity for the eigen value decomposition is in the magnitude of $O(D^3)$ where D is the number of rows, or columns due to symmetry, in the

covariance matrix which is equal to the number of dimensions of features for each playlist track[13]. After the eigendecomposition a projection matrix is needed to project tracks into the principal component space of a seed playlist. To create the projection matrix one matrix inversion with complexity of $O(D^3)$ and three matrix multiplications, also with time complexity of $O(D^3)$ are needed, with a resulting time complexity of creating the projection matrix of $O(D^3)$. To project each track into the principal component space a matrix vector multiplication is needed with time complexity of $O(D^2)$ and this is done for each track which yields a resulting time complexity of $O(ND^2)$ where N are the number of tracks to be projected. The calculation of relative change in magnitude requires calculating the length of each track represented as a vector which includes a squared root operation and thus has the time complexity of $O(D^2)$. Creating a covariance matrix, eigendecompose it and creating a projection matrix does only have to be done once and as the number of tracks in a playlist does not exceed the magnitude of hundreds and the number of dimensions for each track is fixed the steps of the model before projecting tracks can be regarded as constant operations. Using approximate nearest neighbours for one track is a constant time operation and needs to be done for each track in the seed playlist which gives a resulting time complexity of *O(N)* where N is the number of tracks in the seed playlist. The resulting complexity of the model thus becomes $O(nD^2)$ where n are the number of tracks obtained by using approximate nearest neighbours and $n \ll N$ where N is the total number of tracks in the music library. This yields a resulting time complexity which is sublinear in the number of tracks and quadratic in the number of features.

Chapter 9

Discussion and Future Work

9.1 Discussion

9.2 Future Work

Bibliography

[1] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions on*, 17(6):734–749, 2005.

[2] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(8):1798–1828, 2013.

[3] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[4] Asela Gunawardana and Christopher Meek. A unified approach to building hybrid recommender systems. In *Proceedings of the third ACM conference on Recommender systems*, pages 117–124. ACM, 2009.

[5] Philippe Hamel and Douglas Eck. Learning features from music audio with deep belief networks. In *ISMIR*, pages 339–344. Utrecht, The Netherlands, 2010.

[6] Jonathan L Herlocker, Joseph A Konstan, Loren G Terveen, and John T Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53, 2004.

[7] Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *Neural Networks, IEEE Transactions on*, 13(2):415–425, 2002.

[8] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 263–272. IEEE, 2008.

[9] Vivek Kwatra and Mei Han. Fast covariance computation and dimensionality reduction for sub-window features in images. In *Computer Vision–ECCV 2010*, pages 156–169. Springer, 2010.

[10] Beth Logan et al. Mel frequency cepstral coefficients for music modeling. In *ISMIR*, 2000.

[11] Prem Melville, Raymond J Mooney, and Ramadass Nagarajan. Content-boosted collaborative filtering for improved recommendations. In *AAAI/IAAI*, pages 187–192, 2002.

[12] Stanley A Mulaik. A brief history of the philosophical foundations of exploratory factor analysis. *Multivariate Behavioral Research*, 22(3):267–305, 1987.

[13] Victor Y Pan and Zhao Q Chen. The complexity of the matrix eigenproblem. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 507–516. ACM, 1999.

[14] Pandora. The music genome project. Available from: `http://pandora.com/mgp` [cited 13:th of May 2015].

[15] Michael J Pazzani and Daniel Billsus. Content-based recommendation systems. In *The adaptive web*, pages 325–341. Springer, 2007.

[16] John C Platt, Christopher JC Burges, Steven Swenson, Christopher Weare, and Alice Zheng. Learning a gaussian process prior for automatically generating music playlists. In *NIPS*, pages 1425–1432, 2001.

[17] Robert Ragno, Christopher JC Burges, and Cormac Herley. Inferring similarity between music objects with application to playlist generation. In *Proceedings of the 7th ACM SIGMM international workshop on Multimedia information retrieval*, pages 73–80. ACM, 2005.

[18] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001.

[19] Xiaoyuan Su and Taghi M Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in artificial intelligence*, 2009:4, 2009.

[20] George Tzanetakis and Perry Cook. Musical genre classification of audio signals. *Speech and Audio Processing, IEEE transactions on*, 10(5):293–302, 2002.

Appendix A

RDF

And here is a figure

Figure A.1. Several statements describing the same resource.

that we refer to here: A.1