Dissertation Type: enterprise

DEPARTMENT OF COMPUTER SCIENCE

Interactive Session-Aware Music Recommender System

Stefan Klas

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Engineering in the Faculty of Engineering.

Thursday 9th May, 2019

Declaration

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of MEng in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

Stefan Klas, Thursday 9th May, 2019

Contents

List of Figures

Executive Summary

This dissertation is about the exploration of designing and building a music streaming application with a functioning interactive session-aware recommendation system. The interest in building more sophisticated recommendation systems for music has been growing in recent years, with the success of streaming services that give subscribers a vast choice of songs at their fingertips. These services have themselves built functionality to recommend songs to users but currently possess some shortcomings. One particular aspect is the lack of personalised recommendations in real-time. Session-aware recommendation systems, of which algorithmic designs have only been explored more recently, provide one solution. This approach has not yet seen fruition in a commerical application.

The key reason we focus our recommendation system on a user's current session is to interpret their current musical interests. These may change for a multitude of factors but real-time observation allows for far more relevant recommendations that improve the user's experience. Through the conglomeration of past listening sessions, we are able to discover users with similar tastes. We use this information to make active adjustment for the current session to produce recommendations that users will like. The system gives users some degree of understanding around what information the recommendations were based on, which they can alter if needed.

The main contributions and achievements of the project are as follows:

- Built a scalable cloud based music streaming service that allows for song exploration, viewing of historical listening sessions, search and recommendation functionality and allows for some user interaction (8000 lines of code). This has been developed to a professional standard using state-of-the-art web development tools allowing for further expansion and easy maintenance.
- Proposed a completely new means of producing session-aware recommendations through a hybrid ensemble method by using a Latent Dirichlet Allocation (a generative statistical model) as part of the collaborative component and a Word2Vec neural network model as part of the content component (900 lines of code).
- Created a new updated data set making use of the 30Music session data set but with updated Last.fm tags.
- Explored and conducted behavioural testing of the newly proposed hybrid recommendation method (400 lines of code).
- Identified aspects and areas that will benefit from further investigation. An example relates to possibly more suitable implementation choices regarding 3rd party technologies in order to cater for scalability as it would be required by commercial systems. More related to future research is for example the question how a hybrid recommendation system as implemented here could strike a possibly more optimal balance between exploitation and exploration.

We believe that the outcome is a working prototype implementation that demonstrates a currently underutilised approach for users to find songs that they may enjoy. Users are keen to get focused and relevant recommendations, and strong evidence including our results suggests observing the session is a good means to achieve this.

Supporting Technologies

The following list describes software technologies and services used to realise and implement the session-aware recommender system.

- I used the Python `Gensim` package [3] for my implementation of the LDA model, the Word2Vec model, and CoherenceModel.
- I used the Python `funkSVD` class from the `Surprise` package [7] for carrying out the Funk SVD operation.
- The web application part of my system was implemented using a combination of the `React and Redux` JavaScript libraries.
- I used AWS services including EC2, DynamoDB and Lambda for cloud storage and execution.
- I used Auth0 as free authentication service to manage user authentication to the site.
- I used the Spotify API search function to obtain 30 second previews of songs and album covers.
- I used the `pyLDAvis` Python library to analyse the LDA model trained. This can be obtained at [30]
- I used `Node.js` open-source run-time environment to host the webserver.
- I used `MySQL` open-source relational database management system locally to initially form my data set.

Notation and Acronyms

AWS	:	Amazon Web Services
CBOW	:	Continuous Bag of Words
Funk SVD	:	Matrix factorisation proposed by Simon Funk
GRU	:	Gated Recurrent Unit
HDP	:	Hierarchical Dirichlet Process
LDA	:	Latent Dirichlet Allocation
LSA	:	Latent Semantic Analysis
MIR	:	Music Information Retrieval
MRR	:	Mean Reciprocal Ranking
RNN	:	Recurrent Neural Network
RS	:	Recommendation System
SUS	:	System Usability Scale
SVD	:	Singular Value Decomposition
t-SNE	:	t-distributed Stochastic Neighbor Embedding
Word2Vec	:	2-layer Neural Network used for producing word embeddings

Acknowledgements

I would like to express my gratitude to Dr Cian O'Donnell for his supervision in this project. He has given invaluable guidance on completing the project.

Thank you to Dr Ivan Palomares Carrascosa, for giving additional advice and for pointing me in the right direction.

Chapter 1

Contextual Background

1.1 Introduction

Recommender systems have become an important research field since the mid 1990s and have seen significant engagement within the music domain since 2001. They provide a set of techniques to make suggestions to users that are personalised to their tastes. Intuitively, their effectiveness increases with the amount of information gathered about the user, either explicitly or implicitly. Some systems struggle with cold starts when users first use the system, as no information about them is in place [20]. With the information being gathered about users, privacy should always be considered and is gaining greater attention within the field [35][19]. As a note, our implementation ensures that all user information is securely stored and credentials passed between websockets are encrypted.

Music streaming services such as Spotify and Apple Music have become the norm to access songs on demand at any time in any location. Over 190 million active users use Spotify monthly, subscribing to the convenience and price to have around 40 million tracks accessible on demand. With so many songs available comes endless choice and the need for recommender systems to help users find songs they enjoy.

Many of these streaming platforms have implemented recommender systems over the past few years. Instances include Spotify Discover Weekly which builds a recommendation playlist based on songs listened to in the ongoing week by a particular consumer. Pandora, Spotify and others also enable users to start a *radio station* based on a song, artist or tag. The radio station then recommends similar songs thereafter. These systems have had huge success and have become an integral part of the streaming service provided. For instance, Spotify Discover Weekly has reached around 40 million users and is used by many on a weekly basis. In addition to users benefiting from a time efficient way of discovering new songs, the streaming services also benefit themselves. This is because the statistical distribution of playbacks of songs in music libraries generally has a very long tail. Only a few songs have high frequency of playback. These songs, based on their popularity, tend to have the highest artist royalty costs. Therefore, by suggesting songs which may have lower playbacks, streaming service providers can reduce their costs. These cost savings can amount to billions each year, as was concluded from a study in the adjacent movie domain of Netflix recommendations [17].

Even though recommender systems already exist and are in use, they are far from being perfect. This is down to the fact that users' tastes depend on a multitude of factors, of which many are currently not considered in commercial systems. As a result, research into a wide range of alternative methods has recently been happening. This includes amongst others automatic playlist continuation [15], context-based/context-aware music recommendations [24], group music recommendations [25] and session-based/session-aware recommendation systems [22][14][33].

Our implementation will incorporate a *session-aware* recommendation system to assist users in discovering songs in line with their current session. As opposed to *session-based* systems which purely analyse the current ongoing user session, session-aware solutions consider the user's past, historical sessions in the recommendation process [27]. Therefore, keeping track of historical data is important to make the recommendations more accurate.

1.2 Project Description

The aim of the project is to build a music streaming service that uses a *session-aware* approach to recommend songs. The notion of a session here differs from the definition in the Oxford Dictionary. We use the definition given in [33]:

> *Definition (Session).* A session is a set of items (e.g., referring to any objects, e.g., products, songs or movies) that are collected or consumed in one event (e.g., a transaction) or in a certain period of time or a collection of actions or events (e.g., listening to a song) that happened in a period of time (e.g., one hour).

Therefore, more explicitly in our case, a session denotes one sitting where the user listens to songs continuously on the streaming service. The service will provide the functionality to search for songs, get recommendations and allow for listening to the songs to make a judgement on the recommendations.

Session-aware recommendations attempt to suggest songs that best fit the current session the user is having. The motivation behind using this approach is that users often alter their listening habits on a daily basis, even from session to session. This is due to a plethora of reasons including moods and emotions. The session-aware approach tries to model the user's current preferences such that recommendations are far more focused and relevant. We believe that this is a technique that has not been implemented yet by the large music streaming services and therefore we aim to demonstrate that this could work as our primary goal.

A secondary focus for our streaming service is to increase interaction with the recommendation system. Currently there is little to no intuition given to users about why certain songs are being recommended on streaming services nor does it give them any real control. This has been gaining recent attention, such as through the survey by Markus Schedl et al 2017 "Current challenges and visions in music recommender systems research" [28]. This mentions on several occasions the need for greater interaction and feedback for the user.

By giving users a high level intuition for descriptive features ("tags") assigned by the system to songs present in the current session, users will have a greater understanding about why certain new songs are being recommended to them. Users should be allowed to make edits in case the system has made false assumptions. Therefore, our approach will allow users to remove songs that they have already listened to and added to their ongoing session. They may desire to do this if they think that the song features, as revealed to them by our system, are not adequate for them. This gives users a higher degree of control. As a result it will lead to a refinement of future recommendation results. A trade-off between providing too much and too little information has to be considered.

Overall we believe such new features are important for two main reasons. First, they benefit streaming service providers in reducing their costs as explained above. Second, consumers can more efficiently discover new songs which they are likely going to enjoy, given the vast choice available to them.

We aim to use state-of-the-art algorithms from the areas of machine learning and web application development in order to produce the best possible results.

1.3 Challenges

There are significant challenges involved in building a music streaming service and producing good session-aware recommendations.

User interface: In regards to the streaming service, one challenge is building a user interface that is intuitive for the user to use. This involves a clear layout for easy navigation and a responsive system that should respond to data requests in an appropriate time frame.

Scalability: A further challenge involves building a scalable cloud solution that can easily be expanded for both an increase in the user base but also a growth in the song library that is available to the recommendation system.

Specifically for the recommender system, a number of key challenges exist.

Measure for user satisfaction: The question arises which criterion is best suited for measuring user satisfaction from the recommendations made. This is very much a subjective aspect, varying from person to person. For instance, [21] shows concerns raised by some participants that consecutive songs are not varied enough and that a mix of familiar and unknown songs are preferred. Therefore, a combined approach of qualitative and quantitative evaluation is required to understand the full picture and make an accurate conclusion.

Historical session data collection: This challenge involves the gathering of data about user listening habits across a broad set of users such that the data will be appropriate for the recommendation system. This is desired in the form of session records of individual users. A good data collection would ideally consist of:

- Multiple sessions per user.
- Signficiant overlap with regards to songs listened to in different user sessions. This is important such that user similarity techniques can be effectively applied.

There are some music data sets avaialable (such as The Million Song data set [5]), that consist of ratings which users have provided about individual songs. However, in such data sets, there is no notion of session. In contrast, for our session-aware system, we require a data set that includes records of sessions. Such data sets are in short supply.

Adequate methods for different layers of the recommender system: The recommendation challenge itself can be split into two hierarchical layers. The *inter-session layer* handles the dependencies between sessions (of same or different users) whilst the *inner-session layer* analyses the song features within a given session.

Inner-session challenges relate to correctly interpreting the song features within a session. Features such as the genre, style and instruments have an influence on whether other songs would fit into the ongoing session. This once again is subjective as Berenzweig et al. [11] note in the sense that the judgements of the similarity between pairs of artists are not consistent between listeners and may vary with an individual's mood.

Inter-session challenges are associated with understanding the correlations between different users and correlations between different songs across sessions.

- Regarding user correlation: If two users have had similar sessions in the past, the odds are that they will like each other's songs within their respective sessions. The measurement of how similar user sessions are as well as how much this measurement should influence the suggested songs are important aspects to get right.
- Regarding song correlation: If for example two songs often appear together in a session, they are also more likely to appear together in future sessions. This indicates a higher correlation between the two songs.

General challenges of producing music recommendations: There are some specific aspects that make the music recommendation problem different to recommendations in other domains. These should at least be considered, as commonly used techniques may not necessarily transfer well into the music domain.

- *Re-consumption of the same songs is common*: Users often listen to their favourite songs repeatedly. This can happen in any particular time frame, whether that is daily, weekly etc.. Therefore if the recommendation algorithm supports repeated consumption, a decision on which songs to repeat and after how long a period needs to be determined.
- *Music is often consumed passively*: Users often tend to listen to music in the background whilst carrying out some other activity at the same time. This means it can often be difficult to collect feedback or accuratly learn the user preferences.
- *Music is consumed on the spot*: Users want personalised recommendation immediately having given feedback to the system. This means the system should ideally actively learn and be able to adapt quickly the recommendations made.

- *Consumption times of songs are short*: As songs tend to be between two and four minutes, there is only so much information that can be analysed about each song before making a recommendation. This constrasts to the movie domain, where hours of footage can be dissected. On the other hand, if the recommendation does not perform well in a particular instance, the impact on the user is minimal.

Chapter 2

Technical Background

2.1 State-of-art Approaches to Music Recommendations

For building the session-aware music recommendation system within the streaming service, many different machine learning techniques can be applied. Each has advantages and disadvantages that should be considered before incorporating any technique into a solution design.

2.1.1 Recurrent Neural Networks and Gated Recurrent Units

The use of recurrent neural networks (RNN) and gated recurrent units (GRU) within RNN has seen increasing engagement within the *session-based* music recommendation domain. This makes intuitive sense as RNNs are constructed as nodes connected in a directed graph along a temporal sequence, that dynamically changes in size as more data is fed in. Each node can represent a song in an ongoing session which is fed in as input to the network and the output then is the recommended next song. The recurrence relates to the ongoing, evolving music listening session.

The paper in [18] was one of the first to report about this approach, which showed very positive results by employing GRUs. The use of GRUs avoids the vanishing gradient problem of vanilla RNNs. In a RNN with GRUs, sessions were used to train the update and forget gates of the GRU layers. The forget gate is used to decide how much past information shall be forgotten while the update gate decides how much past information should be passed along to the future. Part of this architecture also requires the extraction of metadata and audio features from the songs and embedding of this information in the appropriate format for neural networks. This can include a vast array of approaches including working with emotion-based features, spectrogram similarities, artist similarities etc.

There are some drawbacks with this approach. As it is *session-based*, it does not consider the user's data from previous sessions. This results in a prediction that is not personalised specifically to the user's taste. As a result, it will suffer in performance when considering users that have unusual music sessions. The architecture also relies on the specific, sequential nature of a session, where if the song order is altered, the result from the recommendation will differ. The relevance of song orders has recently come into question as research in [32] alluded to.

Moreover, use of neural networks always has the issue of lack of intuition to understand why particular songs are being recommended. This makes it difficult to relay any insight to the user, which would assist them in gaining more confidence in the system.

Finally, neural networks tend to require large amounts of data for training purposes. In our context, this implies that large amounts of historical music sessions would be required. The risk is that with a small quanity of available sessions, overfitting of the neural network models would be difficult to avoid.

2.1.2 Collaborative Filtering

To recommend items based on the choice of similar users, collaborative filtering has been the most successful approach that is widely used for recommender systems across all domains. It makes the assumption that if users X and Y rate items similarly, they will rate other items similarly. Within the music domain, the items are songs. Users with similar listening habits can be recommended each other's songs which they will hopefully enjoy. Therefore, the objective is to fill in a user-song rating matrix such as the one shown below in Fig. 2.1.

This very simple matrix contains ratings users have given to particular songs, ranging from 0 to 1. The greater the value, the greater the preference for a song. Unrated songs have an entry of '?'.

Looking at this table, the song ratings for user 1 and user 2 are far more similar for the songs which they have both already rated, compared to user 3 (see Song 1 and Song 2). As a result, we hypothesize that Song 3 would be rated by user 2 roughly in line with user 1.

Once the empty fields in the matrix are fully populated with approximations (predictions) for user ratings, we can generate a song recommendation. For user 2, the top recommendation would likely be Song 5, given that this song would likley receive the highest rating value.

Users/Songs	Song 1	Song 2	Song 3	Song 4	Song 5
User 1	0.4	0.5	0.1	?	0.9
User 2	0.4	0.5	?	0.3	?
User 3	0.9	0.1	?	0.3	?

Figure 2.1: User-Song Rating Matrix

In order to populate a rating matrix, two types of methods are commonly used in collaborative filtering which we describe next.

Memory-based methods

These are forms of predicting user-song ratings based on a nearest neighbour method. Within this category, two approaches can be distinguished. The first one is a *user-based* method which populates a user-song matrix as the one shown above, whilst the second is a *song-based* method that populates a song-song matrix.

A *user-based memory-based method* predicts an unobserved song rating $r_{i,j}$ for user i and song j by finding a group of users similar to the target user i. Similarity is measured in terms of actual ratings originally given by users for songs. The greater the similarity, the closer the users' representation will be. This permits to measure similarity through a K-nearest neighbour method. The neighbours' ratings for this song j are then averaged to predict the rating for song j for the target user i. This method produces results that are often easy to explain but suffers when the rating matrix is sparse. This is because the calculated nearest neighbours may themselves also not have rated the particular item (or song).

The *song-based memory-based method* finds the rating similarities between all pairs of songs [6]. It then uses the most similar songs to the user's already rated songs to create a list of recommended new songs. The benefit of using this approach instead of a user-based method is that it avoids the expensive computational cost of finding similarities between all pairs of users. As user profiles change quickly, this would require a frequent recomputation of the entire model.

Both memory-based methods that use a nearest neighbour methodology suffer from issues with scalability. With millions of users and songs, the nearest neighbour algorithm requires a large amount of computational time.

Model-based methods

In this approach, machine learning and data mining algorithms are used to train and model users' preferences. These approaches include decision trees, Bayesian methods and latent factor models. These perform much better in situations when the rating matrix is sparse, which is often the case. They also show better scalability.

One state of the art latent factor model that has seen large success is matrix factorisation. This works by decomposing the sparse user-song rating matrix of dimension N x M into two matrices of size N x K and K x M. K represents the dimension of the latent space. From this decomposition, each user and each song get a K-dimensional representation.

The recommendation process then attempts to construct a new N x M rating matrix which embeds the correlations present in the orignal matrix, but without empty matrix fields. Back to Fig. 2.1, the new matrix then has no empty fields with a '?' in it, however, it will likely have sligthly modified values for the already populated matrix fields.

This can easily be visualised using the illustration from Charu C. Aggarwal, (2016) Fig. 2.2.

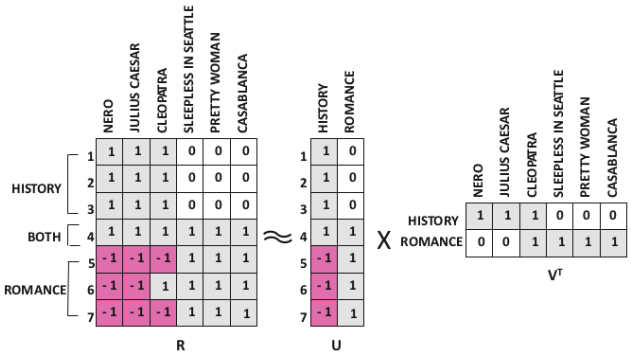

Figure 2.2: Matrix factorisation for movies as per Charu C. Aggarwal, (2016) [10]

As Fig. 2.2 shows for the case of movie ratings, R represents the ratings 7 users gave to different movies (like e.g. Casablanca). Here, these are binary ratings of either 1 or -1 indicating whether a user liked a movie or not. All non-rated movies have been assigned a value of 0.

R is then approximated using two other smaller matrices U and V of rank 2 (K=2). Due to the dimension of the latent space being so low, none of the unrated movies receive any new, different ratings. However, this would change with a higher number of latent factors. Furthermore, because of the low dimension in this case, it turns out that four of the original ratings will be misrepresented, namely the entries in rows 4 to 7 in column 'Cleopatra'.

Latent factor models that use matrix factorisation have the follwing benefits: They consider the correlation between all users and all songs when constructing the predicted ratings as opposed to looking at a subset of nearest neighbours only. Solutions tend to be more robust. Implementations are more scalable, in particular when the rating matrix becomes very large.

A challenge is to estimate the dimension of the latent space, K. The optimal result will be highly dependent on the correlations found in the original rating matrix. The greater the dimension K, the closer the approximation, but the more likely overfitting occurs.

Several matrix factorisation techniques exist and this topic has become in itself an ongoing active research area. In principle many use the ideas from Singular Value Decomposition (SVD) in order to achieve their results.

The SVD of the rating matrix can be written as $R = U\Sigma V^T$ where Σ is a diagonal matrix of non-negative real numbers of dimensions M x N and U, V are of dimensions M x M and N x N respectively. By reducing the dimensionality, we essentially only consider the K largest singular values in Σ. This is similar to the process used in principal component analysis. Therefore, $\tilde{R}$, which holds the approximated ratings, equals $\tilde{R} = U\tilde{\Sigma}V^T$ under the constraint that $rank(\tilde{R}) = K$.

Funk SVD is a particular technique proposed by Simon Funk to produce far more precise predictions. It has been widely adopted in the recommendation systems domain and has been used to produce Netflix movie recommendations. It shares the same idea as mentioned above but contains some differences. Firstly, unlike SVD, the latent factors are not constrained to be orthogonal. This allows for a more accurate representation of the correlations between songs and users. Furthermore, it only considers the songs that are actually rated. As opposed to SVD which models all empty rating matrix fields with a rating of 0, here we ignore those fields completely (they are not fed into the input of the method).

This again produces significantly more precise latent factors based on the actual ratings given. Finally regularization terms are added in order to prevent overfitting.

It is written as an optimisation function which should be minimised using stochastic gradient descent:

$$\arg\min_{U,V} \|R - \tilde{R}\|_{\mathrm{F}} + \alpha\|U\| + \beta\|V\|$$

where $\|.\|_{\mathrm{F}}$ is the Frobenius norm, $\tilde{R}$ is the predicted rating matrix whilst R is the actual rating matrix.

The algorithm contains an iteration count to decide how many times the training for a cell in the rating matrix should be run. The stochastic gradient descent procedure goes as follows:

1. Randomly initialise U and V. Each row of U and V is a latent factor of dimension K, represented as U_u and V_i respectively.
2. For a given number of iterations

   For all known ratings r_{ui}

   Calculate $\dfrac{\partial f_{ui}}{\partial U_u} = \dfrac{\partial (r_{ui} - U_u * V_i)^2 + \alpha\|U_u\| + \beta\|V_i\|}{\partial U_u}$

   Calculate $\dfrac{\partial f_{ui}}{\partial V_i} = \dfrac{\partial (r_{ui} - U_u * V_i)^2 + \alpha\|U_u\| + \beta\|V_i\|}{\partial V_i}$

   Update U_u as $U_u \leftarrow U_u + \lambda * \dfrac{\partial f_{ui}}{\partial U_u}$

   Update V_i as $V_i \leftarrow V_i + \lambda * \dfrac{\partial f_{ui}}{\partial V_i}$

   where λ is the learning rate.

Even though collaborative filtering has many advantages, it also has some shortcomings.

First, it suffers from the *problem of cold start*, when a new user first starts using the system. As no historical listening data exists for this user, the user's row within the rating matrix is completely empty. No correlation exists with any other users and therefore recommendations are more random.

Second, it suffers from *popularity bias*. As popular songs are rated more often, there is an increasing number of strong correlations that exist involving these songs which will be discovered using SVD/Funk SVD.

Finally, in the form stated above, collaborative filtering would not be fit for purpose for a session-aware recommender system, as there is no grouping of songs into individual sessions. Instead, each song is dealt with separately, and information about sessions, if it were available, would be lost. Architectural changes would be required to make use of it in this setting.

2.1.3 Content Filtering

In contrast to collaborative filtering, content filtering techniques look solely at the song features to suggest new songs. These songs will have similar features to the songs they are compared to. Thus, the decision making to arrive at a recommendation is purely taking into account the properties and features characterising content. Different types of features can be observed including acoustic audio features and metadata. Part of the challenge involves creating a similarity function to compare feature values, especially if these features are not quantative. The downside of purely using a content filtering approach is, that it doesn't produce personalised information which is specific to individual users. Given the same input, the recommendations produced for two different users is the same.

Audio-Based Approach

Audio processing such as working with the pitch, loudness, observed chord changes and mel-frequency cepstral coefficient are some techniques that have been applied. These can further be used to classify songs into genres and subgenres or produce similarity scores. For instance, [29] shows a Restricted Boltzmann Machine being used to perform genre retrieval using a music spectrogram. More recently, there has been increasing focus on using deep learning methods to tackle the music information retrieval

(MIR) task [13]. Many of these solutions also make use of a 2-dimensional spectrogram representation with frequency/time axis to train the models. This means they essentially treat the song input as an image which is transformed using methods such as Short-Time Fourier Transform or Constant-Q Transform. These can then be passed into a convolutional neural network to perform chord recognition, instrument detection, tempo detection etc. This approach can retrieve very fine-grain details but the model interpretabilities remain an issue. Furthermore, such specific details as producing beat trackings may not really be of importance to the end user and finding the greatest similarity measurements may not actually increase user satisfaction in the final recommendations.

Metadata-Based Approach

In constrast to the audio processing approach, metadata that comes along with the songs is used to determine similarities. For instance, this could be the release year, genre or lyrics of the track. This approach is used as part of the Pandora recommendation system, where musical experts tag the songs with particular qualities such as "chilled upbeat". Songs with similar semantic descriptive words such as "relaxing" should then have increasing likelihood of being recommended. Techniques such as Term Frequency-Inverse Document Frequency (TF-IDF) can be used to determine the importance of the tags and latent semantic analysis (LSA) can establish the similarities.

The content filtering approach can easily be used for session-aware recommender systems. This would involve extracting the features of the session as it evolves to find similar songs. We have made use of a metadata-based approach using a Word2Vec neural network model to determine the "tag" similarities.

Word2Vec Neural Network and Word Embeddings

Word2Vec is a two-layer neural network that processes text. By inputting a text corpus, it produces word embeddings as the final result, representing words in vector space in N dimensions. The idea is that words that share common contexts are located in closer proximity to one another in vector space and therefore have greater similarity. This similarity can be measured using cosine similarity between the two vectors.

For a quick example let's assume a subset of words or music tags is [["indie", "indie-pop", "alternative", "relaxing", "acoustic"], ["metal", "dark metal"]], whereby the items within inner brackets [,] happen together in the same context. By training the neural network with this given input, the resulting word embedding for "indie" will be far closer to "acoustic" than to "metal".

There are two ways in which the Word2Vec neural network can be structured.

The first one, known as Common Bag of Words (CBOW) has as input multiple context words and outputs a single target word. The second one, known as Skip-Gram, receives as input the target word and outputs multiple context words. Therefore, CBOW learns to predict the word from the context whilst Skip-Gram learns to predict the context from the word. Having a frequent word appear multiple times surrounded by the same context and an infrequent word also surrounded by the same context will give differing results between the two. Given that context, CBOW will spit out the frequent word and computationally ignore the infrequent word. This may not be an ideal solution if considering all words is important. Instead Skip-Gram allows for the input of the infrequent word and therefore does not suffer from this problem.

The structure for a Skip-Gram neural network is shown in Fig. 2.3.

Given a vocabulary of words V, x is a one-hot encoding of the input word and y_i is an array of probabilities which sum to 1 representing the chance of the context words. t_i then represents the true one-hot encoding of the context words.

For instance, in Fig. 2.3, given a vocabulary of five words and a subset of the words as ["indie", "alternative", "acoustic"],
x = "encoding of word *indie*" = $[0, 0, 0, 0, 1]$
t_1 = "encoding of the word *alternative* which is the true context word 1" = $[0, 1, 0, 0, 0]$
t_2 = "encoding of the word *acoustic* which is the true context word 2" = $[0, 0, 1, 0, 0]$
y_1 = "prediction of context word 1" = $[0.1, 0.9, 0, 0, 0]$

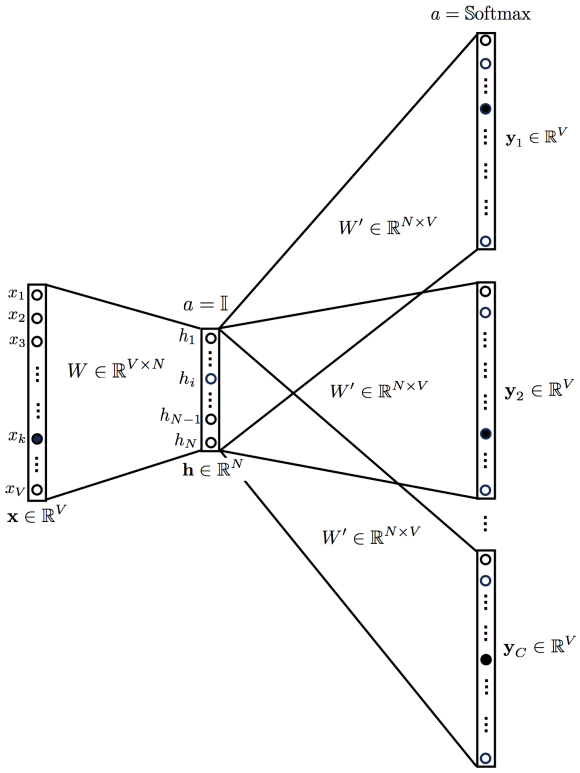

Figure 2.3: Word2Vec Skip-Gram neural network structure as per Alex Minnaar, (2015) [8]

y_2 = "prediction of context word 2" = $[0.5, 0.2, 0.3, 0.0]$

Due to the error existing between t_i and y_i, the weightings of the neural network need to be adjusted given training data. One set of weights, W' is the weight matrix between the hidden layer and the output layer whilst the other set of weights W is the weight matrix between the input and the hidden layer. Due to the one-hot encoded input, rows of W essentially hold the word embeddings that we are interested in to then determine similarity between words. In order to adjust the weights based on the training data, the usual backpropagation technique is applied as shown below.

The equations are as follows:

As seen in the diagram, we have C context words, which are indexed as c = 1, ... C.

$$h = W^T x \tag{2.1}$$

For h, see the hidden layer in Fig. 2.3.

$$u_c = W'^T h = W'^T W^T x \tag{2.2}$$

$$y_c = \mathbb{S}oftmax(u_c) = \mathbb{S}oftmax(W'^T W^T x) \tag{2.3}$$

The process of backpropagation starts with defining a loss function which should be minimised.

$$\mathcal{L} = -\log P(w_{c,1}, w_{c,2}, ..., w_{c,C}|w_o) \tag{2.4}$$

w_o represents the original word mapped to x as a one-hot encoded vector, e.g 'metal'.

$w_{c,i}$ represents the ith context word (not its one-hot encoding).

$P(w_{c,i}|w_o) = y_{i,j^*}$ where j* is the index in which $t_{i,j*} = 1$.

This means, the lower the value predicted at the position where the actual value should be 1, the greater the loss function.

This can then be simplified as follows:

$$\mathcal{L} = -log \prod_{c=1}^{C} P(w_{c,i}|w_o) \tag{2.5}$$

$$\mathcal{L} = -log \prod_{c=1}^{C} \frac{exp(u_{c,j*})}{\sum_{j=1}^{V} exp(u_{c,j})} \tag{2.6}$$

$$\mathcal{L} = -\sum_{c=1}^{C} u_{c,j*} + \sum_{c=1}^{C} log \sum_{j=1}^{V} exp(u_{c,j}) \tag{2.7}$$

We aim to find $\frac{\partial \mathcal{L}}{\partial W'_{i,j}}$ and $\frac{\partial \mathcal{L}}{\partial W_{i,j}}$ for adjusting the weights.
For $W'_{i,j}$ this goes a follows:

$$\frac{\partial \mathcal{L}}{\partial W'_{i,j}} = \frac{\partial - \sum_{c=1}^{C} u_{c,j*}}{\partial W'_{i,j}} + \frac{\partial \sum_{c=1}^{C} log \sum_{j=1}^{V} exp(u_{c,j})}{\partial W'_{i,j}} \tag{2.8}$$

$$\frac{\partial \mathcal{L}}{\partial W'_{i,j}} = -\sum_{c=1}^{C} \frac{\partial u_{c,j*}}{\partial W'_{i,j}} + \sum_{c=1}^{C} \frac{\partial log \sum_{j=1}^{V} exp(u_{c,j})}{\partial W'_{i,j}} \tag{2.9}$$

Iff j* = j will $W'_{i,j}$ influence $u_{c,j*}$. Considering this and using 2.1 and 2.2, the left partial derivative can be written:

$$\frac{\partial u_{c,j*}}{\partial W'_{i,j}} = \delta_{jj_c^*} * h_i = \delta_{jj_c^*} * (\sum_{k=1}^{V} W_{k,i} x_k) \tag{2.10}$$

where $\delta_{jj_c^*}$ is the Kronecker delta.

The only influence $W_{i,j}$ has on $u_{c,t}$ is when $t = j$. Therefore the right partial derivative can also be simplified:

$$\begin{aligned} \frac{\partial log \sum_{j=1}^{V} exp(u_{c,j})}{\partial W'_{i,j}} &= \frac{1}{\sum_{j=1}^{V} exp(u_{c,j})} \frac{\partial \sum_{j=1}^{V} exp(u_{c,j})}{\partial W'_{i,j}} \\ &= y_{c,j} * h_i \\ &= y_{c,j} * (\sum_{k=1}^{V} W_{k,i} x_k) \end{aligned} \tag{2.11}$$

Putting these two results together:

$$\frac{\partial \mathcal{L}}{\partial W'_{i,j}} = \sum_{c=1}^{C} (\delta_{jj_c^*} + y_{c,j})(\sum_{k=1}^{V} W_{k,i} x_k) \tag{2.12}$$

$$W'_{i,j} = W'_{i,j} - \eta \frac{\partial \mathcal{L}}{\partial W'_{i,j}} \tag{2.13}$$

For $W_{i,j}$ similar steps are involved to arrive at:

$$\frac{\partial \mathcal{L}}{\partial W_{i,j}} = \sum_{k=1}^{V} \sum_{c=1}^{C} (\delta_{kk_c^*} + y_{c,k})(\sum_{k=1}^{V} W'_{j,k} x_i) \tag{2.14}$$

$$W_{i,j} = W_{i,j} - \eta \frac{\partial \mathcal{L}}{\partial W_{i,j}} \tag{2.15}$$

Recall, our main goal is to find maxtrix W, as this represents the embeddings of all words of the vocabulary V in an N-dimensional space. Now we have a means to change the word embeddings for each instance of training data. Therefore, by running the algorithm iteratively over several epochs, we can determine the best or optimal word embedding representation.

In the music context, this method can be used to determine similarities between songs through analysing the words used in their descriptive features. In particular, for a session-aware recommender system, we can envisage using the above method to determine the similarities of

- different words (through their word embeddings) that in turn describe songs within the current session and
- words that describe potential new candidate songs which we may want to recommend to a user.

2.1.4 Hybrid Solution

A hybrid recommender solution involves a combination of a collaborative and a content filtering technique. Due to the downsides of using these methods in isolation, combining them allows for more robust inference and therefore better recommendations. This approach can take the form of:

- *Ensemble design* - where the results of the different techniques are combined to produce an output.
- *Mixed system* - where items recommended by the different techniques are presented together side by side.

A hybrid solution of the ensemble type was used in the final implementation of the session-aware recommender system, combining content-based and collaborative-based components in order to make the system more robust.

To use the collaborative component in a session-aware manner, we have used a different machine learning technique to assist us, namely the Latent Dirichlet Allocation, which we introduce next.

2.2 Latent Dirichlet Allocation

Latent Dirichlet Allocation (LDA) is a topic model that allows for the discovery of abstract "topics" in a collection of documents containing words. It makes the assumption that each document is made up of a mixture of topics and that each word has a likelihood of being within a certain topic.

It is constructed using the Dirichlet distribution (Dir(α)) which is a multivariate generalization of the beta distribution and is parameterized by a K-dimensional vector α (concentration parameters). Different α values will produce different Dirichlet distributions. When all α values are less than 1, the distribution takes a concave shape while values greater than 1 produce a convex distribution. For the use of LDA, lower α values tend to be selected. This is due to the assumption made that documents only contain a small number of topics and that topics only contain a small number of words.

The LDA method can be used in the session-aware music setting if the songs can be described by a collection of words. In this case, the corpus represents the different sessions, a document represents one session and the words are the descriptive features of the songs in that session.

The figure 2.4 below gives an overview of how it would be used. The descriptive tags of songs from historical sessions are used to train the LDA model. This produces two kinds of distributions which are the Topic-Word Distribution and the Topic-Document Distribution.

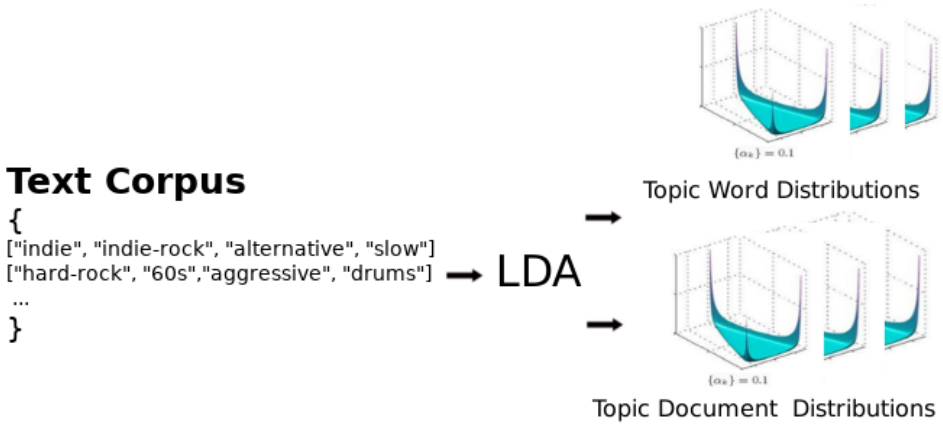

Figure 2.4: LDA method in session-aware music setting

The LDA approach can be understood as using a generative process. A little notation is required to descrbe it:

Notation

- K: is the number of LDA topics. Unlike in other methods such as hierarchical Dirichlet process [31], this needs to be defined initially. In the music setting, this represents the different "music flavours".
- M: is the number of documents. In the music setting this is the number of sessions.
- V: represents the vocabulary of words. In the music setting it refers to the vocabulary of song features or song tags.
- N_i: is the number of words in document i. This relates to number of unique song features occuring inside session i.
- θ_i: distribution of topics in document i. In the music setting, this is the distribution of latent music topics in session i. (Topic Document Distribution)
- φ_k: distribution of words in topic k. This represents the distribution of words in a specific latent music topic. (Topic Word Distribution)
- $z_{i,j}$: identity of the topic of the jth word in the ith document.
- $w_{i,j}$: identity of the word in ith document, that is the jth unique word in that document.

Given this, the generative process is as follows:

1. Choose $\theta_i \sim Dir(\alpha)$ where $i \in \{1..M\}$. θ_i is an element of the *support* of this Dirichlet distribution, of which the entries add to 1 (lies on the K-1-simplex). α here represents the hyperparameter of the topic to document distribution.

2. Choose $\varphi_k \sim Dir(\beta)$ where $k \in \{1..K\}$. Similarly φ_k is an element of the *support* of this Dirichlet distribution which is representing the topic to word distribution.

3. Going through each word in each document where $i \in \{1..M\}$ and $j \in \{1..N_i\}$:

   a) Choose a topic $z_{i,j} \sim Multinomial(\theta_i)$. This multinomial is a categorical distribution of K possible categories.

   b) Choose a word $w_{i,j} \sim Multinomial(\varphi_{z_{i,j}})$. This multinomial is a categorical distribution of V possible categories.

In summary, by following this process outlined above, we are able to generate documents. We use the process of Bayesian inference to tweak the distributions such that we can create documents that are similar to the training corpus that we insert. Once we have this, we theoritically have a way to replicate the training corpus if we wish to do so.

The Bayesian inference for the complete joint probability that we aim to solve is:

$P(\boldsymbol{W}, \boldsymbol{Z}, \boldsymbol{\theta}, \boldsymbol{\varphi}; \alpha, \beta) = \prod_{i=1}^{k} P(\varphi_i; \beta) \prod_{j=1}^{M} P(\theta_j; \alpha) \prod_{t=1}^{N_j} P(Z_{j,t}|\theta_j) P(W_{j,t}|\varphi_{j,t})$

Looking at a specific point of this joint probability distribution gives some intuition. Assume we are given two topics, three documents and four words. As the formula indicates, we need to loop through all the multiplication symbols considering the probabilities at each stage.

For topic 1:

What is the probability of this topic-word distribution → e.g. [0.4, 0.3, 0.2, 0.1]
For document 1:

What is the probability of this topic-document distribution → e.g. [0.2, 0.8]
For word 1:

(What is the probability of a topic from the above topic-document distribution → e.g. 0.2) *
(What is the probability of word given this chosen topic → e.g. 0.4)

... [continued]

We finally want to find the posterior distribution $P(\boldsymbol{Z}, \boldsymbol{\theta}, \boldsymbol{\varphi}|\boldsymbol{W}, \alpha, \beta) = \frac{P(\boldsymbol{Z}, \boldsymbol{\theta}, \boldsymbol{\varphi}, \boldsymbol{W}|\alpha, \beta)}{P(\boldsymbol{W}|\alpha, \beta)}$. However as calculating the denominator is intractable, variational inference is used to approximate a solution. The proof for this is long and therefore is omitted but can be found in [16].

All this can be used in the session-aware music domain to find different types of sessions that both users themselves and different users are having. It allows for a far more general categorisation of session types than just observing e.g. the genre or subgenre.

2.2.1 User Feedback For Recommender Systems

There are different ways to get feedback from users that should be incorporated to improve the recommendation made. This can come in the form of both explicit and implicit feedback.

Explicit Feedback

Explicit feedback is typically of the form of ratings the user explicitly gives to songs. This could either be a numerical star system from 1-5 or a thumbs up/thumbs down response. The main benefit with this approach is that accuracy is very high where the system can be relatively certain that the response reflects the user's opinion. However, often users do not give this additional feedback resulting in scarce information gathered.

Implicit Feedback

Implicit feedback exploits users' behaviour as they navigate the system and tries to infer user preferences from their behaviour. For instance, users may skip some songs which could suggest that they did not enjoy them. Similarly, users may navigate back to songs they have already listened to. This could be taken into account to model the level of desire of users' requests for novelty in song recommendations (e.g. either high or low).

Chapter 3

Project Execution

3.1 Requirements on the Overall System

In this section we list a small set of minimum requirements which we deemed important from a user experience point of view for our prototype implementation.

- Users shall be able to sign up to the prototype streaming service and create an account.
- The system shall enable users to actually play the audio of songs. This will help them to assess whether the recommendations are suitable.
- Exploration of songs by users shall be possible through means like searching or selecting music songs from an offered list.
- The user shall be able to see what their current, ongoing music session comprises, in terms of songs and their associated tags.
- A user shall be able to retrieve their historical music sessions and replay songs from those past sessions.
- The streaming service shall be able to give recommendations in a timely manner.

3.2 Design

Designing the architecture required for a session-aware streaming platform requires consideration of different aspects.

- Streaming Service Architecture Design
- Recommendation System Design
- Designing for user interaction with the recommendation system

3.2.1 Streaming Service Architecture Design

The full architectural design for the streaming service is shown in Fig. 3.1 below. The architecture can easily be split into a front-end component, consisting of a web-application and authentication system and a back-end component for data storage, data collection and recommendation functionality. While initially the architecture included a futher database for storing actual mp3 files, this was later omitted and replaced with the Spotify API. Working with the Spotify API provides us with a huge repertoire of free song previews, of length 30 seconds, that can be streamed into an audio player. The API also provides other functionality such retrieving album images, that improve the user experience.

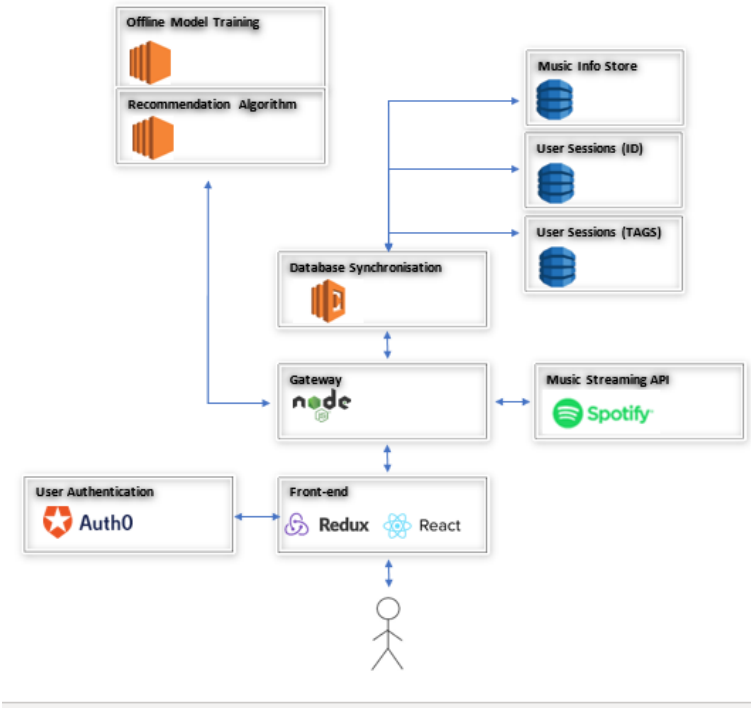

Figure 3.1: Streaming service architecture design of our solution

To build a music streaming service, the architecture requires a few components which are grouped into front-end and back-end.

Front-end Architecture Components

The music streaming service has been designed to be used through a web browser. ReactJS, which is a JavaScript library for building user interfaces, originally created by Facebook and later open-sourced, was chosen to build the interactive user interface for a few reasons.

Firstly, ReactJS works with states within the application where views are efficiently updated and rendered only when necessary. This means that parts of the webpage can remain unchanged whilst others change based on user input. This capability was essential to build a music player that can continue playing whilst the user navigates the site, with no webpage reloading occuring.

Secondly, due to the use of component classes in ReactJS, both reuse and expansion of components for extra features would require lower development and maintenance time.

Redux, which is an open-source JavaScript library for managing application state, was also added to work in combination with ReactJS. As our single-page application has to hold many states including the current active song, song queues, current sessions and user authentication details, management of the states can become both complicated and incomprehensible. This can especially become confusing as many of these states are retrieved from asynchronous function calls. Redux solves this using the three principles: “only one store, store is read only and state changes can only be made with pure functions”.

For user authentication, Auth0 authentication service was used to provide login functionality for the webpage. In usual fashion, users receive an access token and an id token, of which the id token will be used to identify the user and assist in finding historical sessions and recommendations specific to the user. Auth0 has the benefit of being both cost efficient and provides a dashboard for management of logged in users if necessary.

The full front-end functionality is depicted in Fig. 3.2 in form of a UML activity diagram. It also shows the actions with which the system responds to different kinds of user interaction. The process starts with

the user signing in to the service or optionally creating a user account first. Once the user arrives at the main landing page, they have multiple options to further proceed and interact with the service.

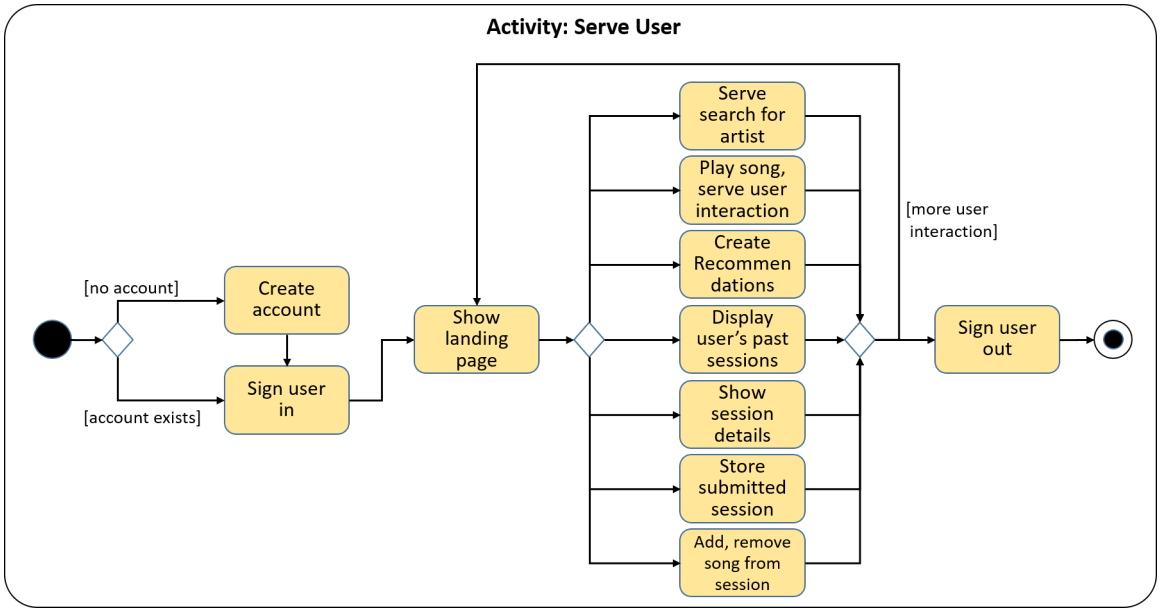

Figure 3.2: UML diagram showing front end interaction

Back-end Architecture Components

A large proportion of the back-end infrastructure is hosted on Amazon Web Services (AWS). This is a cloud solution, that provides effortless scaling of back-end resources if required to adjust for traffic to the site.

The NodeJS server, hosted on an Amazon EC2 node acts as the access point, where both demands from the front-end and responses to front-end requests pass through. Part of its role is ensuring that the requests from the front-end can be fulfilled before sending the requests on. For instance, a user's session should not be submitted for storage if it is empty.

The EC2 node also hosts the recommendation part of the system. First, it includes an online portion that responds to requests made by users. Second, it also includes an offline portion which trains the implemented LDA and Word2Vec models and creates the rating matrices e.g. in a 24-hour cycle. This ensures that further session data gathered from active users is incorporated to improve recommendations made.

The DynamoDB NoSQL databases provide a convenient means of storing metadata that is required for different parts of the system. This has been split into storing music information (such as artist, tags, name) and storing user sessions. The user session data has been stored both in terms of 'ids' and as 'tags' in order to reduce the computation required and decrease response time.

Finally, a Database Synchronisation component which we have written as an Amazon AWS Lambda function, is used when users submit a music session they have had. Its purpose is to ensure that the two different databases for user sessions are in synch. Otherwise, conflicting information could cause inaccuracy and errors occuring.

3.2.2 Recommendation System Design

In designing the recommendation system, we have opted for a hybrid method in order to build a robust recommendation system that performs well in all circumstances. This consists of both a content and collaborative filtering component that will be combined to produce a top-10 song recommendation output based on the current session the user is having.

Some important assumptions have been made with our design. First, the songs that will be recommended will not have been present in previous sessions of the same user. Therefore we are modeling all users as having a preference for novelty and discovery of new songs. The second assumption is that users listen to relatively similar types of songs in a continuous session. A mix and match of many wildly different types will result in difficulties for the collaborative component to find any correlation and will produce poor results in the recommendations.

The defining features of songs, that we will make use of, come from the process of *collaborative tagging*. This is a communal effort to describe items allowing for effective exploration of information. Company Last.fm has for many years provided the functionality for consumers to tag different songs. Over the years, many people have added descriptive words to the songs, including sub-genres, instruments and emotions. Based on observing only the most frequently occurring tags, a reasonably accurate descriptions of the songs can be produced.

Collaborative Filtering Design

As mentioned in Section 2.1.2, collaborative filtering in its normal form does not facilitate a session-aware approach. However, inspired by findings from paper [23], we have proposed a new solution around this problem.

The design works as follows: Given the historical sessions of different users, the sessions are first converted into tags. This is done by using the Last.fm database to find the tags assigned to each song in a session. These tags form a corpus, which can be processed by an LDA model to find topic-session distributions and the topic-tag distributions. A **topic-session distribution** describes the likelihood of latent topics to be present in a given session. A **topic-tag distribution** describes the probability of tags occurring in a specific topic.

Next, instead of having just one user-song matrix, a user-song matrix is created for each topic. For every topic k, the position (i,j) in the kth topic-user-song matrix is filled if the user i has listened to the song j in a session. The value inserted is the probability that the session in which the song occurred is of that particular topic k.

Fig. 3.3 shows the structure and Algorithm 1 shows the processing steps in pseudo code for clarity.

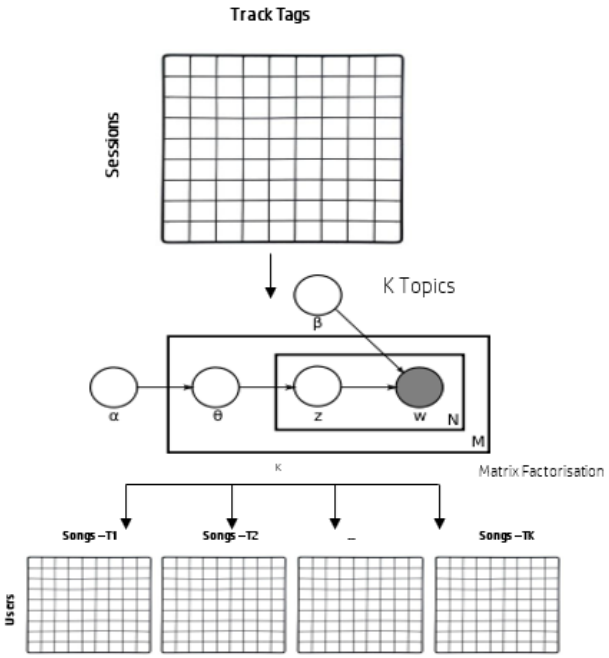

Figure 3.3: From historical sessions to filling in topic-user-song matrices

Algorithm 1 Collaborative Filtering (Offline)

```
1: procedure sparseRatingMatrices(D)                          ▷ D is historical sessions
2:     R ← matrix(topicNum,userNum,songNum)
3:     for session S in D do
4:         userIndex ← findUserIndex(S["userId"])
5:         topicSessionDis ← lda(S["songs"])
6:         for topicIndex, topicProb in topSessionDis do
7:             for song SO in S["songs"] do
8:                 songIndex ← findSongIndex(S0)
9:                 if R[topicIndex][userIndex][songIndex] == 0 then
10:                    R[topicIndex][userIndex][songIndex] ← topicProb
11:                else
12:                    R[topicIndex][userIndex][songIndex] ← 0.5*(R[topicIndex][userIndex][songIndex]+
   topicProb)
13:    return R
14: procedure fillSparseRatingMatrices(R)
15:    A ← clone(R)
16:    latentFactorDimension ← 10
17:    for topicIndex in R do
18:        A[topicIndex] ← funkSVD(R[topicIndex], latentFactorDimension)
19:    return A
```

Notes: Procedure *SparseRatingMatrices* prepopulates the topic-user-song matrices with probabilities as defined above for songs which users have already listened to.

Line 2: We create an empty 3-dimensional matrix R to store the two-dimenstional user-song matrices across all topics.
Line 3: The process loops through all recorded sessions in the database.
Line 4: For a session, we find its user and map it to a user index.
Line 5: The LDA algorithm is run with the tags of the session as input and tells how likely latent topics occur in this session.
Line 6: We loop through the topics in the topic-session distribution
Line 7: We loop through all songs of the historical session currently analysed
Line 8: We find the index of each song so we can reference an entry in matrix R
Lines 9-12: We assign new topic probabilities to the entries of matrix R.

Once these implicit probabilistic ratings have been added, these matrices will be relatively sparse. Using the Funk SVD algorithm, defined with 10 as the dimension of the latent factors, correlations between users and songs will be discovered and will be used to fill in the rest of the matrices with proposed ratings.

Notes: Procedure *FillSparseRatingMatrices* fills the remaining missing matrix entries using the Funk SVD algorithm.

Line 15: We make a clone of matrix R.
Line 16: We set the dimension of the latent factors to 10.
Line 17: We loop through all latent topics present in the matrix R.
Line 18: The Funk SVD algorithm returns a two-dimensional matrix as we iterate over the 3rd dimension. It substitutes empty matrix entries with non-zero values and may modify already existing non-zero probabilities as present in matrix R.

Both procedures always run on the historical data, so not on the current session data. Funk SVD specifically takes as input only the actually filled-in entries of topic-user-song matrices, so unfilled matrix entries corresponding to unrated songs are not considered, in contrast to normal SVD.

The resulting outcome of this collaborative filtering design is that users who have previously had similar sessions will have a greater likelihood of being recommended each other's songs.

Content Filtering Design

For the content filtering component of our hybrid recommender system, we will similarly use the Last.fm song tags to train a Word2Vec Skip-Gram neural network. The reason we use Skip-Gram is that many of the descriptive tags will be infrequent. This produces a word embedding of the song tags within an N-dimensional latent space where the cosine similarity function can be used to numerically measure similarity. By calculating the average cosine similarity between a new song's tags and the current session tags, we get a probability of how well that song would descriptively fit together with the current session.

Algorithm 2 Content Filtering (Online)

1: **procedure** PREDICTSIMILARITY(C, S) ▷ *C is current session tags, S is song tags*
2: $v1 \leftarrow$ mean([word2vec(word) **for** word **in** C])
3: $v2 \leftarrow$ mean([word2vec(word) **for** word **in** S])
4: $cossim \leftarrow \frac{v1 \cdot v2}{||v1|| ||v2||}$
5: **return** $cossim$

To briefly describe the above algorithm:

C is the group of tags of all the songs present in the currently ongoing music session.
S is the group of tags associated with a particular candidate song which we check for suitability to be recommended.
Line 2: We observe the word embeddings in C (vectors in the latent space) and average them. This results in vector 1.
Line 3: We do the same for a newly proposed candidate song. Vector 2 is the average of the vectors associated with S.
Line 4: We calculate the cosine similarity between these two vectors.

Hybrid Recommender System Design

Our final recommender implementation combines both the collaborative and content filtering components to give a rating score for each song in the database. Using the current session the user is having, the top 10 songs, that the user has not previously had in a session, are sent as recommendations to the front-end of the solution. The final part of the pseudo code is shown below:

Algorithm 3 Hybrid Recommender System (Online)

1: **procedure** RECOMMENDSONGS(C, $userId$, A) ▷
C is current session tags, userId identifies user, A is filled rating matrices
2: $alpha \leftarrow 0.5$
3: $A^{(n)} = normalise(A)$
4: $topicCurrentSessionDis \leftarrow lda(C)$
5: $userIndex \leftarrow findUserIndex(userId)$
6: $songRatings \leftarrow [0...0]$
7: **for** *songIndex,song in songDataset* **do**
8: **for** *topicIndex in topics* **do**
9: $songRatings[i] \leftarrow songRatings[i] + topicCurrentSessionDis[topicIndex] * ((1 - alpha) * A^{(n)}[topicIndex, userIndex, songIndex] + alpha * normalise(predictSimilarity(C, song))$
10: **return** $top10SongRatings(songRatings)$

The above procedure creates the top 10 songs that shall be recommended to a particular user based on their current session tags and the topic-user-song matrices (all held in A).

Line 2: A parameter alpha is used to weight the contributions which the content-based and collaborative-based components make to the final recommendation result. As default, we found that an alpha value of 0.5 works well.
Line 3: We normalise the rating values in the 3-dimensional matrix A to the range [0 - 1].
Line 4: We run the LDA algorithm on the group of current session tags to find the topic distribution

of the current session. This is a categorical distribution expressing the likelihood of a topic being associated with the session.
Line 5: To be able to reference a specific user, we find the user's index in the matrix.
Line 6: Song ratings are initialised to zero.
Line 7: We loop through all songs in the song database.
Line 8+9: For a particular candidate song, we accumulate the song ratings across all topics. We pick up with factor alpha the contribution from the content-based filtering and with factor (1-alpha) the contribution from the collaborative-based filtering. This weighted sum is then multiplied with the chance that the current music session is of a specific topic. Similarly as we have normalised the rating values of A, we normalise the output from the predictSimilarity measurement.
Line 10: We return the top-10 rated songs as recommendations whereby those songs have not been listened to by the user before.

The reason, why we normalise the rating values is as follows: First, it gives easy intuition into the contributions made from each of the components. Second, this then allows for tuning of the hyperparameter alpha, such that the optimal weightings are given to the two filtering components to overall provide an optimal result.

It is important to mention that Algorithm 3 runs in an online setting, whilst Algorithm 1 runs in an offline setting. As a result the 3D rating matrix from Algorithm 1 is passed in as parameter (A). This means that the collaborative filtering component is not updated for the current session but relies only on historical sessions. This design decision has been made to ensure a responsive recommendation algorithm that can produce results in seconds.

The final resulting recommendation makes its suggestion like this:
Each song has a unique identifier *uid*.

- User 1 has previously had a "jazz like session" with songs: ["$uid : 32$","$uid : 3$","$uid : 56$","$uid : 102$"]
- User 2 has previously had a "jazz like session" with songs: ["$uid : 32$","$uid : 3$","$uid : 56$","$uid : 122$"]
- User 2 has also previously had an "indie like session" with songs: ["$uid : 12$","$uid : 13$","$uid : 14$","$uid : 17$"]
- User 1's current session: ["$uid : 12$","$uid : 17$"]
  - There is a high chance that the current session is of "indie like topic".
  - Collaborative component: User 2 shares a similar listening session with User 1 in jazz. For the "indie like session" user 2 also liked "$uid : 13$" and "$uid : 14$"
  - Content component: It observes whether tag word embeddings for songs "$uid : 13$" and "$uid : 14$" align with tags in the current session.
  - If both content and collaborative parts give high ratings to candidate songs, those songs are highly likely to be recommended.

Having discussed the algorithms to produce the recommendations, we can now summarise the design of the recommender system to cater for this.
Fig. 3.4 shows an activity diagram of the backend system part that is used for offline data processing.

Fig. 3.5 shows the diagram for the activity to create the song recommendations online during an active user session.

Designing for user interaction with the recomendation system

Part of the objective of the system, is to give users greater intuition about what the recommendations are based on. To fulfill this, we have designed a feature where users are able to view the current session they are having and further explore the tags that come along with songs in that session. This then allows them to remove songs, that they feel may not have the appropriate tags to assist the recommendation process.

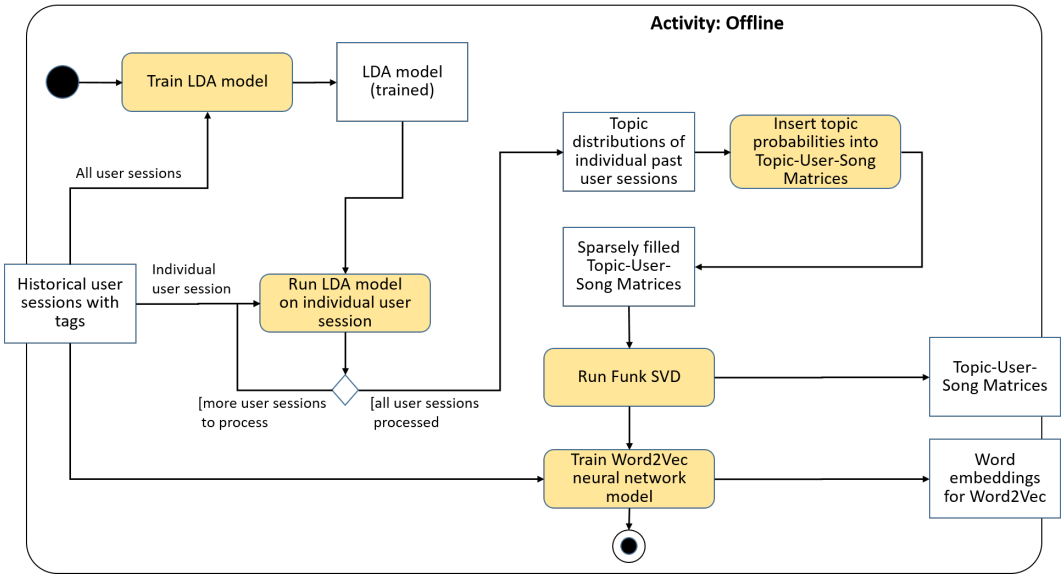

Figure 3.4: UML diagram showing the offline process for the recommendation system

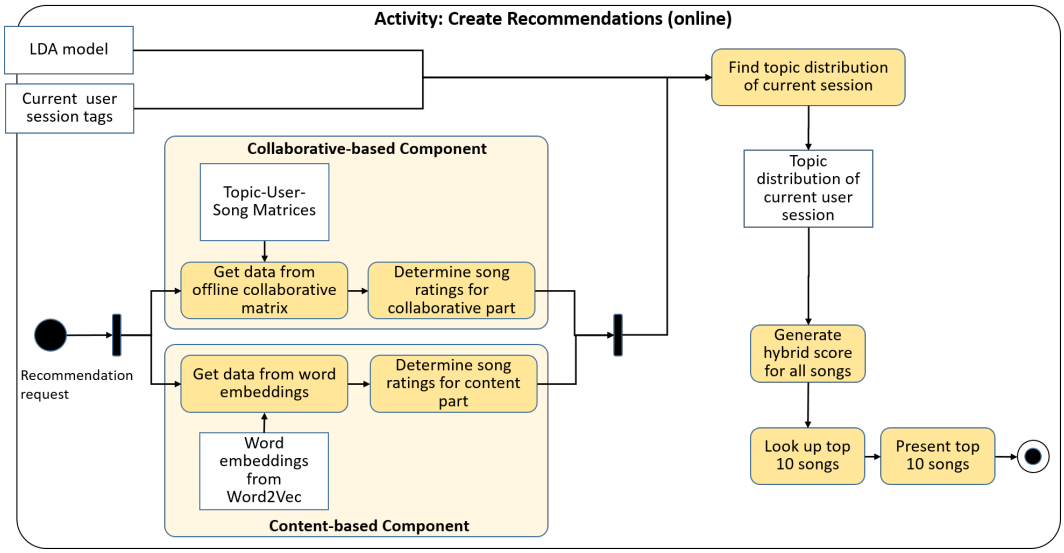

Figure 3.5: UML diagram showing the online recommendation process

In addition, the initial design was to allow users to edit song tags and give tags different weights. Assume a user is engaged in an ongoing music listening session on our streaming service and the session is currently of a 'jazz type'. The user could then specifically add further tags from a vocabulary to the session and have control over the importance of such tags through giving them different weights. This would allow a user to specify that they want for example more of 'saxophone' in the ongoing session.

3.3 Implementation

The implementation to build the session-aware streaming platform can be split into a few distinctive parts.

1. Data Set Preparation
2. Latent Dirichlet Model Training and Filling of Rating Matrices
3. Word2Vec Model Training
4. Spotify API
5. User Authentication

3.3.1 Data Set Preparation

In order for the system to produce recommendations and allow for the evaluation of the quality of the recommendations, historical music sessions containing information about songs listened-to were required. We used the 30Music data set [1] which contains a collection of around 2 million user sessions, retrieved from radio stations through the Last.fm API [4]. From this large collection, only a small subset of 154 session were selected for initial training purposes which amounts to around 1007 unique songs. The sessions were selected at random but had to contain at least 5 songs. This was done as the greater the number of songs, the more precise the descriptions and therefore the better the recommendation performance.

For each song in a session, the Last.fm topTags API was used to retrieve the most up-to-date descriptive tags. This API would respond with a maximum of 20 tags, but could sometimes be empty. Therefore, before uploading of the tags (and other information) to the databases, a final check was made to ensure that sufficient tags were present to describe the complete session.

The session data was then uploaded to the "userSessions(ID)", "userSessions(TAGS)" databases and the song metadata was stored in "music info store" (see Fig. 3.1).

3.3.2 Latent Dirichlet Allocation Training and Filling of Rating Matrices

Given the stored sessions and their respective tags, training of the LDA model was required to discover the different types of sessions users were having. The corpus was the set of historical session tags which was fed into the LdaModel class of the *gensim library*.

Before this training stage, the session tags needed to be cleaned to improve accuracy. As these tags came from the process of collaborative tagging, anyone could tag any song in any form they liked. Therefore, some of these tags produced noise which needs to be removed. A corpus frequency filter was used, which basically filters out all the infrequently used tags. As a result the filter removed around half the present tags.

In addition, we implemented an alternative filtering mechanism that allows for finer-granular filtering. It is called TF-IDF and stands for *term frequency inverse document frequency* using the class TfidfVectorizer from the library *sklearn*. With many tags e.g. due to a long ongoing user session, it helps to determine how important individual tags are to this session which the TF-IDF method allows. The importance of a tag increases with the occurrence of the tag in the session, but is offset by how often that tag occurs in the complete corpus. The importance can be mapped to a weighting factor which we combined with the frequency filtering criteria. However, this alternative had minimal impact on the results due to our relatively small data set.

Part of the challenge for this process was choosing the number of latent topics that should be present as a parameter for the LDA model to produce the best results. As these topics represent different "types" of sessions which is an abstract idea, there is no clear and correct answer. One strategy that was attempted was to use a coherence measurement on the trained LDA model. This produces a score for assessing the quality of the learned topics [26].

The coherence of each topic is measured as:

$$coherence = \sum_{i<j} score(w_i, w_j)$$

where $w_1..w_n$ are the top N words describing the topic.

Using the UMass measure,

$$score(w_i, w_j) = log \frac{D(w_i, w_j) + 1}{D(w_i)}$$

where $D(w_i)$ is the count of documents containing word i and $D(w_i, w_j)$ is the count of documents containing both words i and j. In our music setting, words and documents correspond to tags and sessions, respectively.

In essence, the UMass score gives a score about how well infrequent words in a topic are described by frequent words. The graph showing the results produced by our solution for a particular LDA (as LDAs are dynamic) is shown in Fig. 3.6. This is with the top N words considered being 20, aggregated over all topics. The higher the coherence score, the clearer the topics.

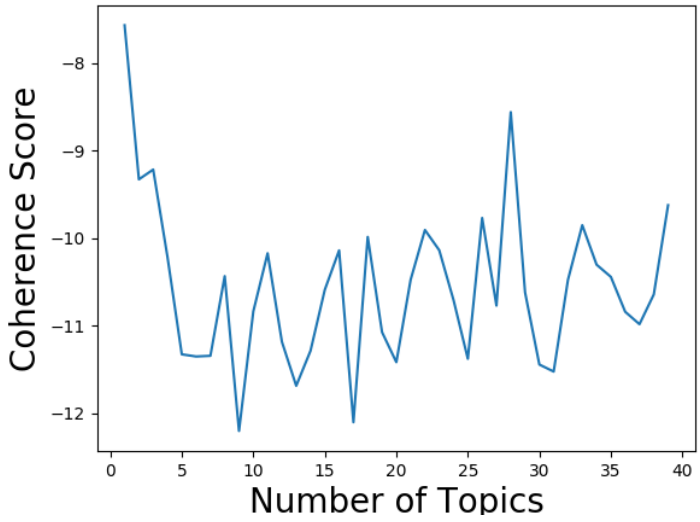

Figure 3.6: Coherence measurement for a trained LDA model

Ignoring the high scores for 1 or 2 topics, peaks can be found starting at 12, 16 etc. with a large rise at 27 topics.

As a second tool to determine the ideal number of topics we used the *pyLDAvis* library, which produces an interactive topic model visualization [30]. It allows for a detailed way of interpreting the topics formed, showing the most prominent words per topic and the distance between topics in topic latent space. Fig. 3.7, right side, shows the prominent words found in Topic 4 which could be described as a "folk singer-songwriter" topic. As can be seen, this topic has picked up the term "indie" (see the red portion of the bar) but the majority of this term's frequency is found in one ore more different topics (see the blue portion of the bar).

After using this approach to manually assess the different number of topics, 16 topics were picked. Using a greater number such as 27 was deemed to overfit on the small test data (of 154 sessions). For a larger data set, this would be a far more appropriate way to discover all the different types of sessions user have and therefore produce a more personalised recommendation system.

Having trained an LDA model, this was then used to fill the different topic-user-song rating matrices. As shown in Algorithm 1, the ratings for the user songs were inserted and then Funk SVD was performed to fill in the missing values. Recall, that for implementation purposes, we merged the individual topic-user-song matrices for all topics into a single 3-dimensional matrix.

The funkSVD algorithm was performed using the *scikit-surprise* library, which comes with a range of matrix factorisation techniques. This algorithm was applied to find 10 latent factors that represent correlations in the rating matrices.

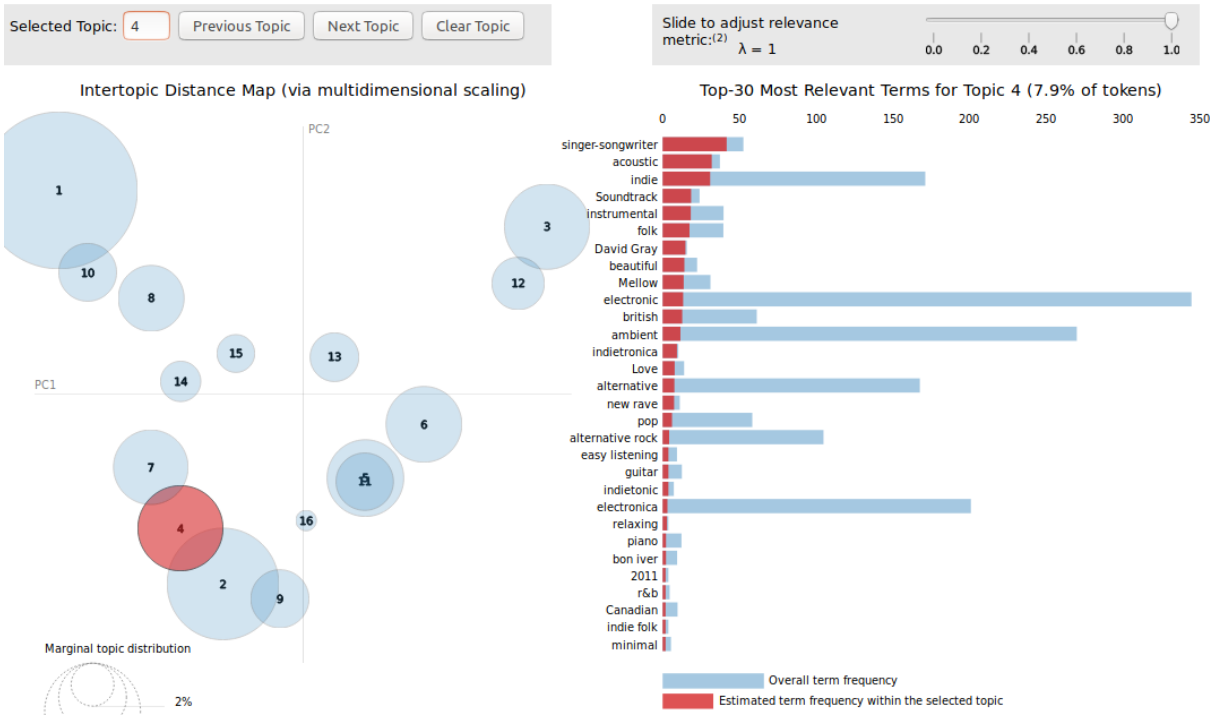

Figure 3.7: Interactive topic model visualisation with Topic 4 currently selected

3.3.3 Word2Vec Model Training

The training of the Word2Vec model was required for the content filtering component of the recommendation system. As was shown in Section 2.1.3, this involves training a neural network to process a latent representation of the words. This was done on the session-tag corpus.

Similarly to the LDA training, the session tags need to first be cleaned to improve accuracy. Once this was done, each tag in a session and the surrounding tags in that session were passed into the neural network. This meant the neural network trained with the assumption that the words in the session had similar semantic meaning. Then, with the repetition of multiple words often occurring together, this similarity measurement would become more accurate.

The above was realised using the *Word2Vec* class in the gensim library, with a default latent space of 100 dimensions. The window size for the training procedure was set to a very large number, which meant that for each tag, all the other tags in the session where considered and deemed to be in the same context.

3.3.4 Recommender System

The trained LDA model, rating matrices and Word2Vec model were stored as files on the online EC2 recommendation server. The online recommender system (Algorithm 3) was then run on a Flask server hosted on Gunicorn and would read these static files in when needed. This allowed for a fast response time, for the benefit of the user.

3.3.5 Gateway Server and Spotify API

The requests made from the front-end were dealt with using a NodeJS webserver which is hosted on PM2 process manager. To process such requests, we used the JavaScript *request* library and to process and relay the requests to the AWS components we used the *boto3* library. Functionality includes:

- Getting song metadata from music store

- Getting user historical data (previous sessions)
- Getting recommendations for a user
- Getting search results from music store
- Getting paginated song list from music store
- Submitting user session to session databases
- Getting Spotify previews and additional information (e.g. album cover images)

Any session submitted would first pass through a custom AWS lambda function we created. This was used to ensure that the two session databases ("User Sessions(ID) and User Sessions(TAGS)") were in sync.

The Spotify song previews were discovered using the search functionality of the Spotify API. Using the song's name and artist, the first song that contained a preview link was used. In a few cases, this gave results for alternative versions of the song. Information about the song popularity and artist cover was also gathered.

3.3.6 Front-End Implementation

The front-end web application was implemented using a combination of HTML, CSS, ReactJS and Redux. Part of this process involved building the audio player from scratch, using only the lower level html audio tag. This was done to allow for easy extension and customisation of the design in the future. The process involved building the states the player could be in and keeping track of these states as the user navigates the site. For instance, if the user is currently listening to a song and clicks on a different song in the list, most would assume that this new song would start playing automatically. It is these user actions and the desired outcomes that all had to be considered.

The front-end also makes use of a *react-infinite-scroller* package. This was used to request further songs when the user scrolls to the bottom of the page. In addition, it allows for a far more fluid means of searching for different songs as opposed to having to manually request more songs each time.

A final front-end-related decision was made to add genres of the songs listed. This was done because looking only at the song attributes of name and artist would not give the user a good idea of the nature of a song. To realise the feature, we mapped the first few tags for the song through a subgenre to genre function. If a subgenre was not found, it would display "n/a".

Chapter 4

Critical Evaluation

The evaluation of the music streaming service which we developped can be separated into two different parts. One part explores the user experience in using the streaming service, involving participants to trial the service. The other part looks specifically at the performance of the recommendation system.

4.1 User Experience

The figures below show some screenshots of the web application interface when a user has created an account on the streaming service.
Fig. 4.1 shows the User Profile Home Page that a user initially arrives at. Available songs can be browsed, played and added to their current session.

Fig. 4.2 shows the Current Session Page. It represents the user's current session and includes the top tags for each song in their session.

Fig. 4.3 represents the occurrence of a recommendation request and the response delivered by the system.

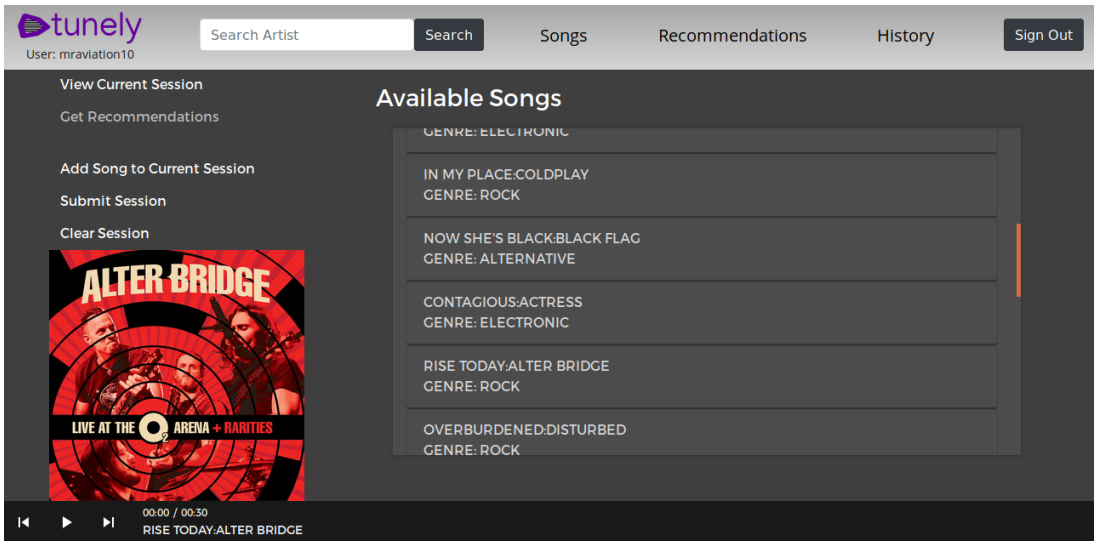

Figure 4.1: User Profile Home Page

In order to evaluate the web application interface, we have used the system usability scale (SUS) developed by John Brooke [12]. It is a simple, ten-item attitude Likert scale, giving a high-level subjective view of usability. It produces a single score on a scale of 0 - 100. The reason that this has been used instead of other methods such as usability testing or heuristic evaluation, is its ease to administer the survey to

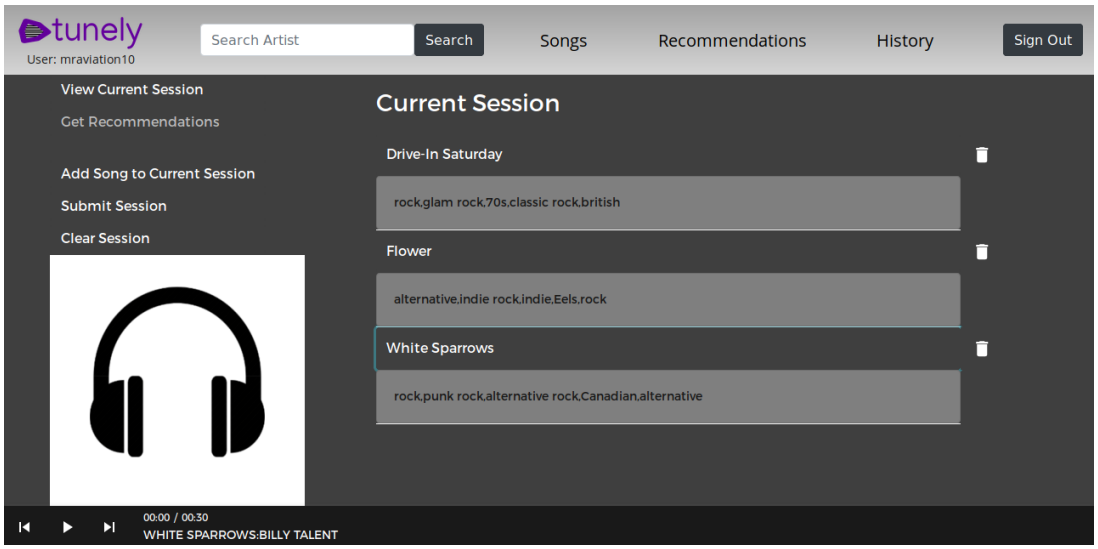

Figure 4.2: Current Session Page

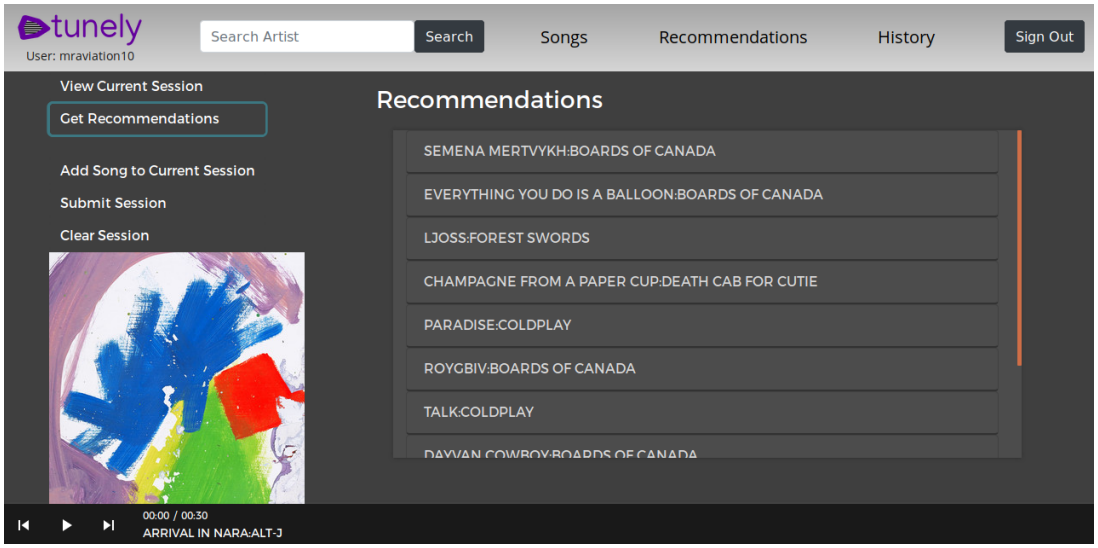

Figure 4.3: Recommendation Page

partcipants. It does not require the participant and assesser to be in the same room, nor does it require a User Interface expert.

The following 10 questions were asked:

1. I thought the system was easy to use and self-explanatory
2. I found the discovery of songs difficult and time-consuming
3. I thought the system provided sufficient feedback when I interacted with it
4. I thought the system lagged or was unresponsive
5. I thought the system produced some good recommendations
6. I thought the system was difficult to navigate
7. I felt confident that the system would produce good recommendations each time
8. I felt the system did not provide enough information about the artist and songs
9. I would use the system again if the songs were full versions

10. I felt the system produced similar recommendations each time. There wasn't enough diversity.

We were able to receive feedback from 12 participants, of which the break down is shown below. Participants were asked to respond to each question with 5 options from Strongly Disagree (1) to Strongly Agree (5).

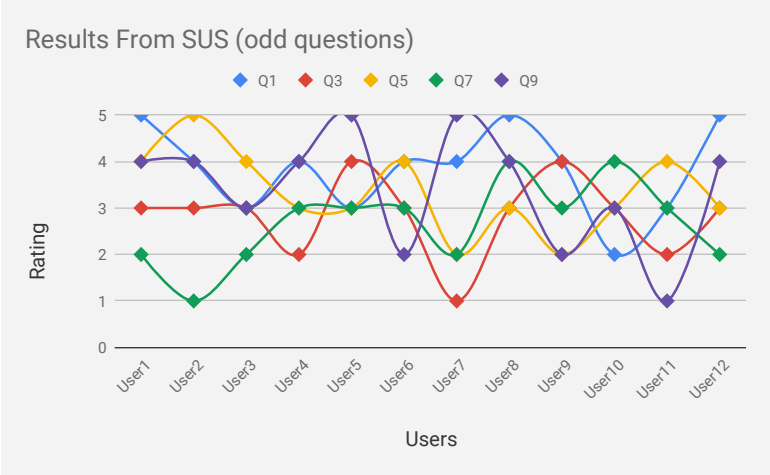

Figure 4.4: Results for odd questions in SUS (high value is good)

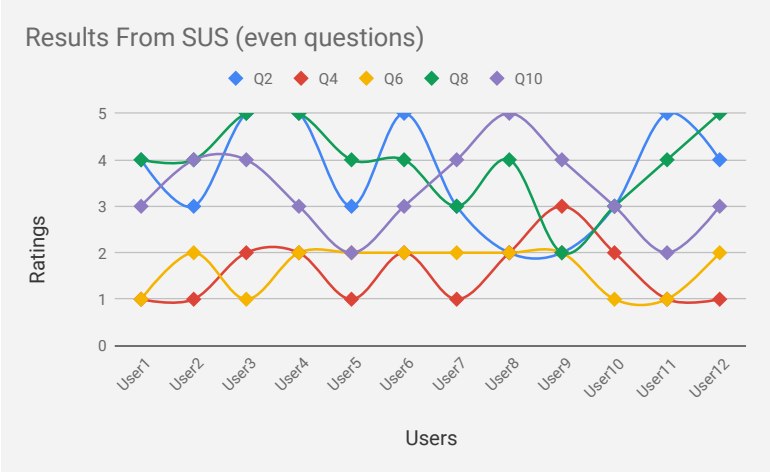

Figure 4.5: Results for even questions in SUS (high value is bad)

Using the SUS formula and averaging over the different users, the final score was **54.7**. This means users have characterised the interface as "ok", rating D on the SUS scale. It is clear from this survey where the system performs well and where more work is required.

The survey question 'ease of use of the system' produced one of the highest consistent scores. Our application benefits from being designed with some similarities to other existing systems. The biggest negatives were the process of discovering songs and confidence in the recommendations. The first aspect could easily be addressed by providing greater subcategorisation and playlists. The second aspect is likely to heavily rely on the small subset of songs actually available to be recommended and the relatively few sessions the system is trained on. Having said this, most test users did feel the system provided good recommendations at some points.

4.2 Performance of the Recommendation System and Behavioural Testing

The correct quantitative evaluation measurements to assess performance of recommender systems is subject to an ongoing debate. This of course is due to the very subjective nature of the recommendation task.

4.2.1 Visual Inspection and Plausibility Checks

Before quantitatively assessing the performance, we wanted to ensure through behavioural testing that the recommender system was functioning as intended and works in a plausible way. Specifically, we wanted to analyse the changes made in the rating matrices from the collaborative component. As each matrix is large, viewing the whole matrix in one go is not feasible. Therefore, we discovered for each user, those users, that had the greatest influence in the collaborative effect. An example of this is shown below for an illustration:

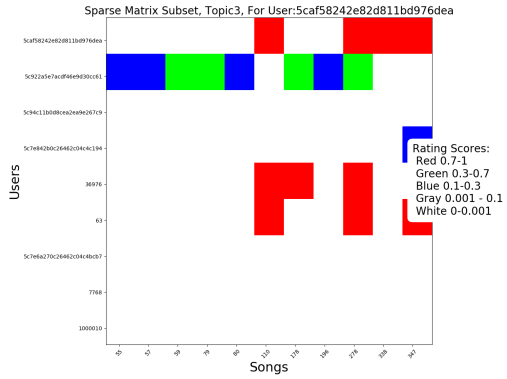

Figure 4.6: A portion of rating matrix before Funk SVD, Topic 3

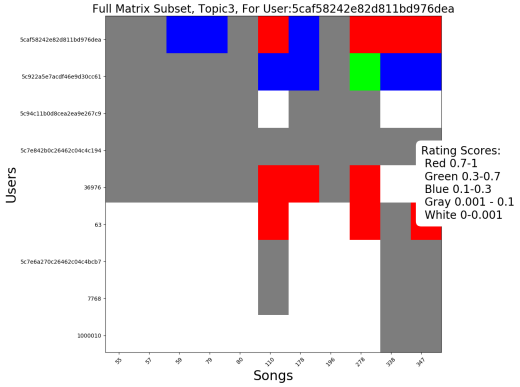

Figure 4.7: The same portion of rating matrix after Funk SVD, Topic 3

The images above show a portion of the rating matrix of topic 3, before and after Funk SVD was performed (Fig. 4.6 versus Fig. 4.7). The colour code in the image indicates the strength of the ratings.

The image on the left side shows some users that could influence the selected user in row 1 after Funk SVD is applied. The image on the right shows the result once the Funk SVD was performed. What we can observe is that ratings in this portion of the matrix have changed due to using the Funk SVD.

The user in row 2 had a large influence on the target selected user in row 1. Before the Funk SVD operation, these two users only shared one common song they had both listened to. Due to this sharing, they influenced each other's ratings which leads to altered ratings in the matrix. User 1 has gained ratings for 3 significant songs colour-coded in blue.

From the observation, the question arises, why e.g. the user in row 6 didn't get the second but last column filled in with a higher value. This is due to the correlation the user possesses with other users not visible on the grid. Therefore, the grid is only useful to show and illustrate the possible reasons for changes for the target user.

In conclusion this means that the algorithm is definitely finding correlation between users' past sessions, even if the observation is a little convoluted.

4.2.2 Producing Training and Test Data

When assessing and demonstrating the actual performance of the system, the first task involved splitting the historical session data into training and test data. This was done by:

1. Picking 30% of the users at random. From these users, their last session was removed to become part of the test data. All other historical sessions became part of the training data.

   Only 30% of users were tested as many users in the historical session data only had one session. Considering all users for the test data would result in the collaborative filtering component having little to no effect on the recommendations.

   Given this data split, the recommender system was trained from the training data.

2. The test data was then split per user into the last song in the session and the rest of the songs. These latter songs were used to represent a current session. The optimal outcome is that the last song would appear in the top 10 recommendation list.

To summarise the terminology:
Test session: is the current session of a test user excluding its last song.
Test song: is the actual, last song of the test user's session.

4.2.3 Results

Due to the very specific means of testing session-based/session-aware recommendations, the following metrics were considered but deemed unsuitable for this specific case:

- **Precision** measures from the recommended songs, how many songs the user likes. As we have no explicit feedback built into the system to rate recommendations, we could not assess this.
- **Recall** measures what proportion of songs which the user liked, were actually recommended. In this case, we could consider "user liked songs" as songs that were actually played. However as the recommendation only outputs 10 songs, it would be unlikely that the exact same song which the user actually played as the last one in his real session, would be recommended. This does not necessarily mean the system isn't producing good recommendations.

Instead, we looked into alternative, more suitable metrics.

Rank: The rank is the position at which a song would be placed into a list of recommended songs, in descending order. The top 10 ranks constitute the output of our recommender system. Rank 0 is best.

In addition, using the rank to establish the performance of session-aware/session-based recommendation systems is commonly used as part of the Mean Reciprocal Ranking measurement. MRR allows to compare results with other research.

Mean Reciprocal Ranking (MRR): Given Q queries, it calculates the average multiplicative inverse of the rank of the **test song**.

$$\text{MRR} = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{\text{rank}_i}$$

We calculated MRR on our test data, with the results shown below in Fig. 4.8. The table shows the comparison between our hybrid recommender system and a plain vanilla, normal Funk SVD implementation, which consists of only 1 user-item matrix. This user-song matrix does not capture the idea of sessions and thus ignores information contained in the current music session. Instead it gives a rating of 1 to songs that the user listened to. We use this normal Funk SVD implementation as a simple **performance baseline**.

The result shows that given 33 test sessions, the hybrid technique has consistently outperformed the baseline performance. This demonstrates that incorporating session information creates a more accurate recommendation if we compare the recommendation to what actually happened with the test user.

The graph in Fig. 4.9 below shows the breakdown of the system performance for individual test sessions by plotting Rank over User ID (which represents each user's test session). Rank refers to the order the hybrid system and the comparison baseline have given to each test song.

As we can see, in nearly all cases, our hybrid recommender system produces a better (speak lower) rank for the particular *test song* that is associated with each user's test session.

	30Music (Subset) + Trial Users
Total Data (sessions,users,songs)	154,102,1007
Test Data (sessions, users, songs)	31,31,334
MRR Score For Our Hybrid Session-Aware RS	**0.08549**
MRR Score For Single Funk SVD RS	0.00533

Figure 4.8: Mean Reciprocal Ranking results for test songs

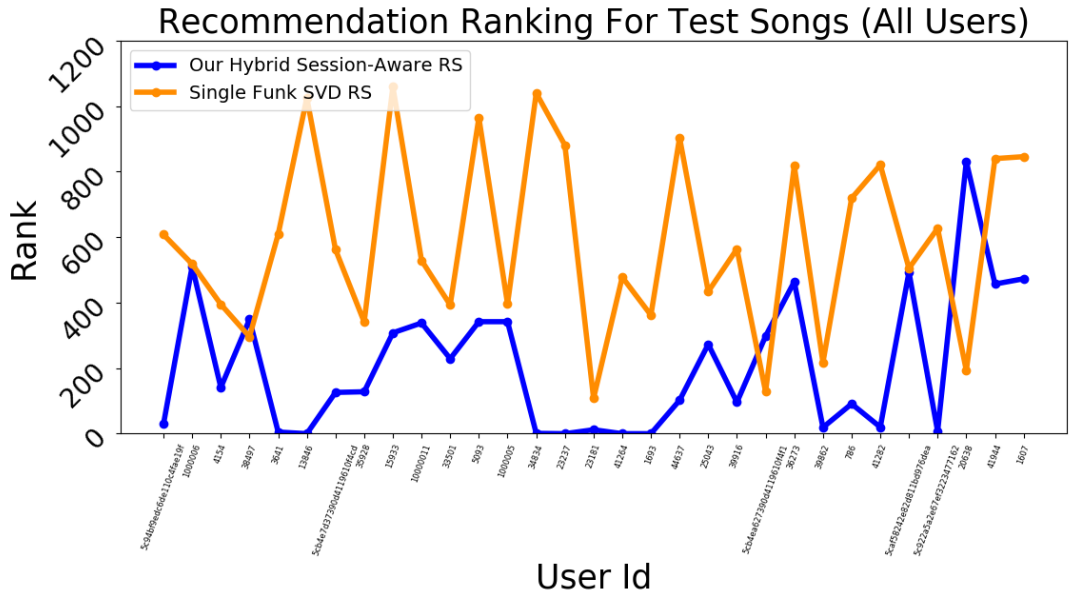

Figure 4.9: Comparison of single Funk SVD (baseline) with our hybrid method

Looking at the actual ranking score, there are cases where the hybrid recommendation system performs very well, producing rankings in the top 100 songs, whilst there are a few other cases where the system doesn't perform that well.

Now, for a subset of our users in the training data, no historical sessions existed because they only contained one session which is considered as their 'current, ongoing session'. The recommender system finds itself in an extreme *cold start situation* for the collaborative component for such users, as no historical session data can be processed. The reason why our hybrid system performs reasonably well even in such cold start situations is because it leverages the content-based component.

Therefore, it may be more suitable to conduct a comparison of performance against the baseline for only those users that possess historical training sessions.

Non-cold start users: possess at least one historical training session.

Cold start users: possess no historical training session.

The graph in Fig. 4.10 shows the rank of the test song for the subset of non-cold start user sessions, for the hybrid system in comparison to the baseline single Funk SVD approach. It shows that the hybrid system performs largely better than the baseline, when we focus on non-cold start users only.

In the chart there is only one test user, for whom the recommendation is ranked worse compared to the baseline approach. Our hypothesis is that due to the large difference in rank, the content-based component was responsible for generating this outlier.

We can further investigate this by analysing what recommendation would result, if we only used the content-based component of our system. Fig. 4.11 shows the recommended rankings for the test songs,

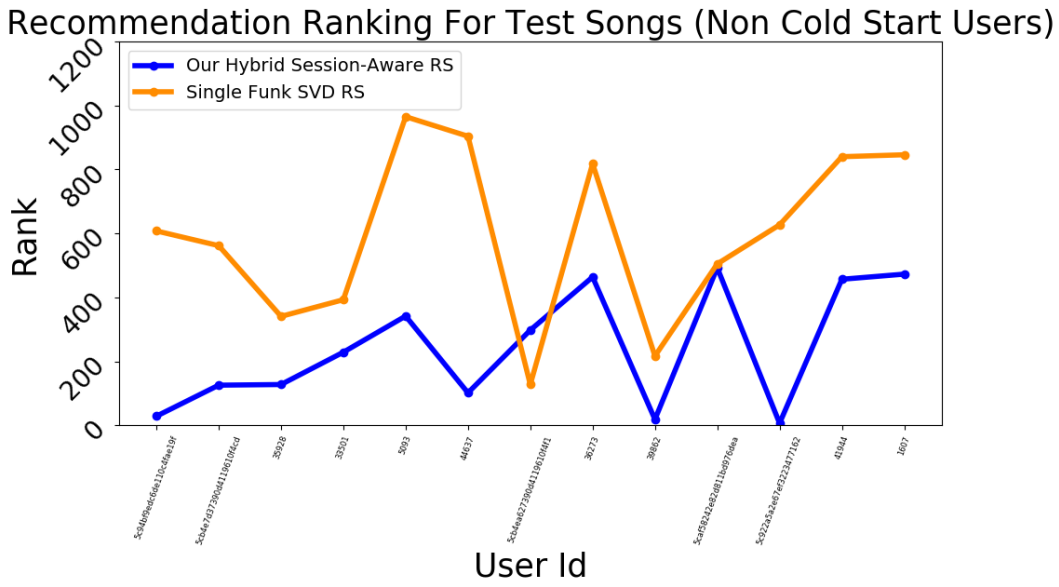

Figure 4.10: Comparison of single Funk SVD (baseline) with our hybrid method for non-cold start users

first as produced by the hybrid system and second as produced by its content-based recommender component only.

This analysis confirms our hypothesis that the content-based part of the system was largely responsible for generating the ranking of that particular test song. Some deeper analysis reveals that the Word2Vec neural network produced a poor similarity rating between test session and test song. Specifically, we find that the most prominent tags are 'alternative', 'indie', 'pop' in the test session and 'ambient', 'indie', and 'soundtrack' in the test song.

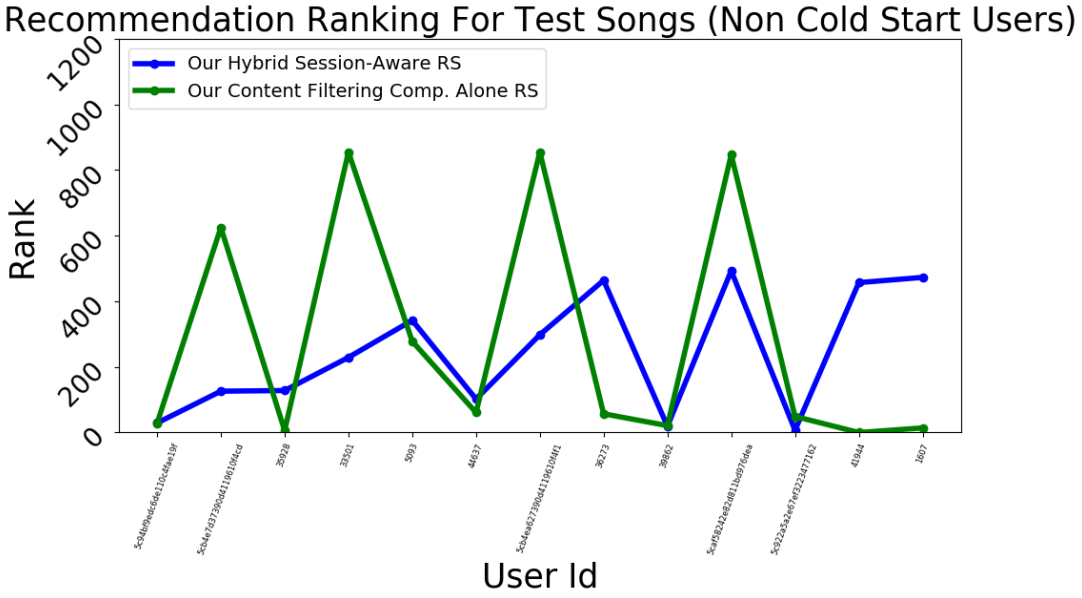

Figure 4.11: Comparison of the content component and our hybrid method for non-cold start test members

By looking at the word embeddings produced from the Word2Vec neural network, we can get some extra information. As our word embeddings are represented in a 100-dimensional latent space, we used the **T-distributed Stochastic Neighbour Embedding (t-SNE)** in order to get a 3-dimensional representation of word embeddings which we can visually inspect.

t-SNE constructs a probability distribution over pairs of high-dimensional objects such that the more similar two objects are in high-dimensional space, the higher the probability will be for the pair.
Then it uses the Kullback-Leibler divergence to find the locations of objects in the lower dimensional space, such that the pairwaise low-dimensionality probability distribution which also measures similarity looks the same as the probability distribution for the higher dimensional space.

For the test session, we find the following tags in Fig. 4.12:

- 'alternative' on the very left side, middle,
- 'indie' on the very very left side, upper corner,
- 'pop' very close to 'indie'.

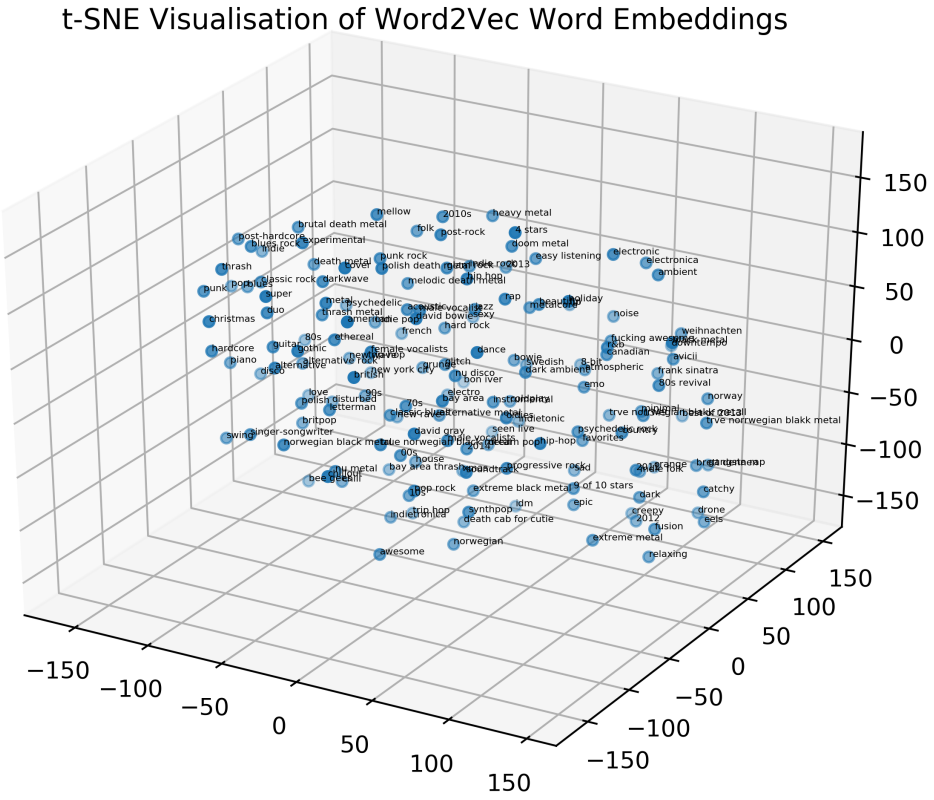

Figure 4.12: t-SNE visualisation of word embeddings

Fig. 4.13 zooms into a portion of the 3D space for the sake of better readability.

Oh the other hand, we find the word embeddings of the tags of the test song as follows:

- 'ambient' in the right upper part,
- 'indie' in the left upper corner,
- 'soundtrack' in the middle lower part.

Averaging the locations of the tags in the test session and in the test song, respectively, leads to two vectors. We find that the cosine similarity between these two vectors is relatively small compared to other songs. Hence, the poor rank of the test song.

The reason behind this is, that the training of the Word2Vec neural network didn't often see the tags of the test song in the same context as the tags of the test session.

We believe that the test song tags 'ambient' and 'soundtrack' aren't a perfect fit anyway to the test session tags. So in this respect, our system appears to perform well.

However, it does raise the following questions:

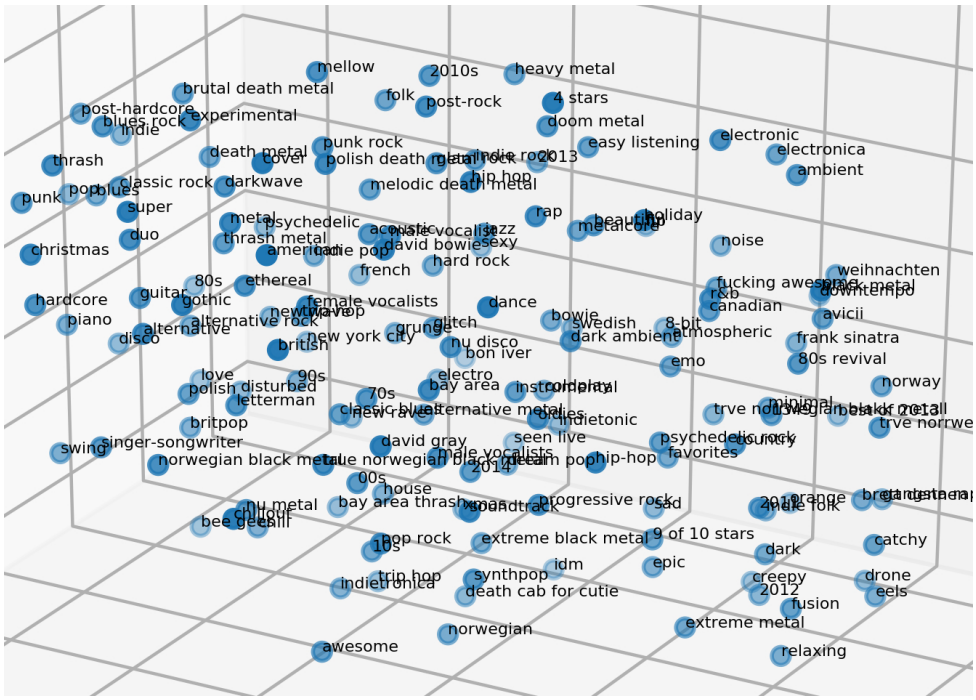

Figure 4.13: t-SNE visualisation of word embeddings (zoomed)

1. We rely heavily on the assumption that the test song 'naturally' follows on from the test session (e.g. a 'indie rock' song follows 5 'classic rock' songs, thus the session is still pretty much 'rock'). This assumption may not be fully true in practice.
2. How accurately are the song tags, which are provided by people, characterising the actual song?
3. Would a greater amount of training data for the Word2Vec neural network help to cover more special situations such as in this case? We think that would indeed be the case.

Next we study the probabilities which were assigned to the test songs by our hybrid solution and compare this to a baseline of randomly choosing a song from the song data set. Our expectation is that our hybrid system performs for the test songs much better than a system that would simply make random recommendations.

Fig. 4.14 plots the probability of the relevant test song getting recommended by our system over the different users' test sessions. Note that the test songs vary across the different users.
The flat line represents the uniformly random choice of picking the test song from the song data set. The blue line shows with which probability the test song would be picked by the hybrid system.
What is clear from the diagram is that the hybrid system performs better than a random selection across all the test users.

The full picture can be seen when we show the probabilities of the whole universe of songs for a particular user's test session as in Fig. 4.15. It focuses on a particular test user and plots the probability with which each song from the song data set would be recommended to this particular test user, either by using our hybrid recommender system or by making a uniform random choice. The graph shows that a large portion of the songs in the database would have had probabilities less than the one associated with the uniform random selection. As the hybrid solution has recommended the test songs with a higher probability as we have seen before, it illustrates that the hybrid system is functioning well as intended.

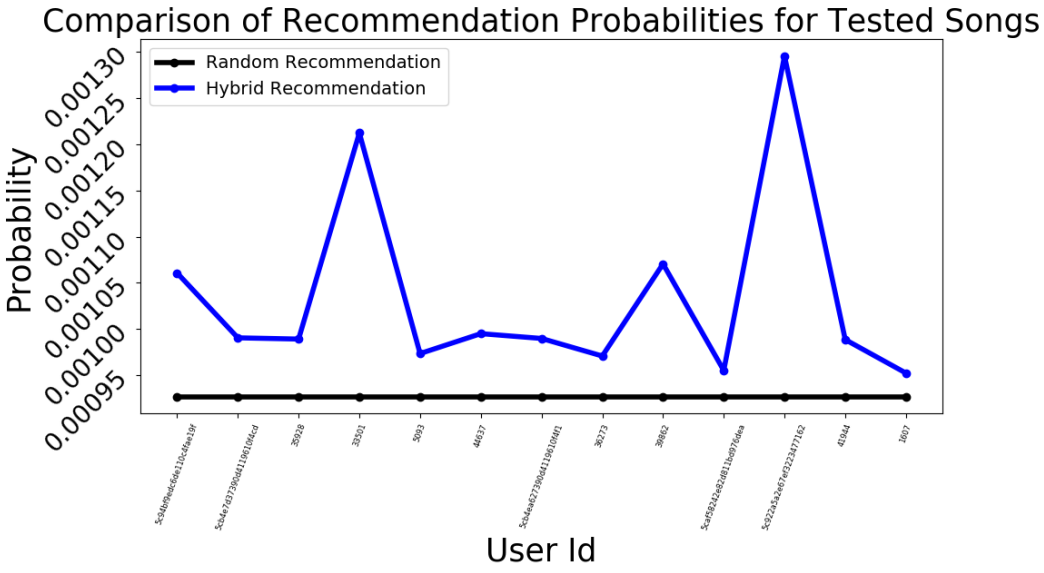

Figure 4.14: Comparison of hybrid recommendation versus random recommendation

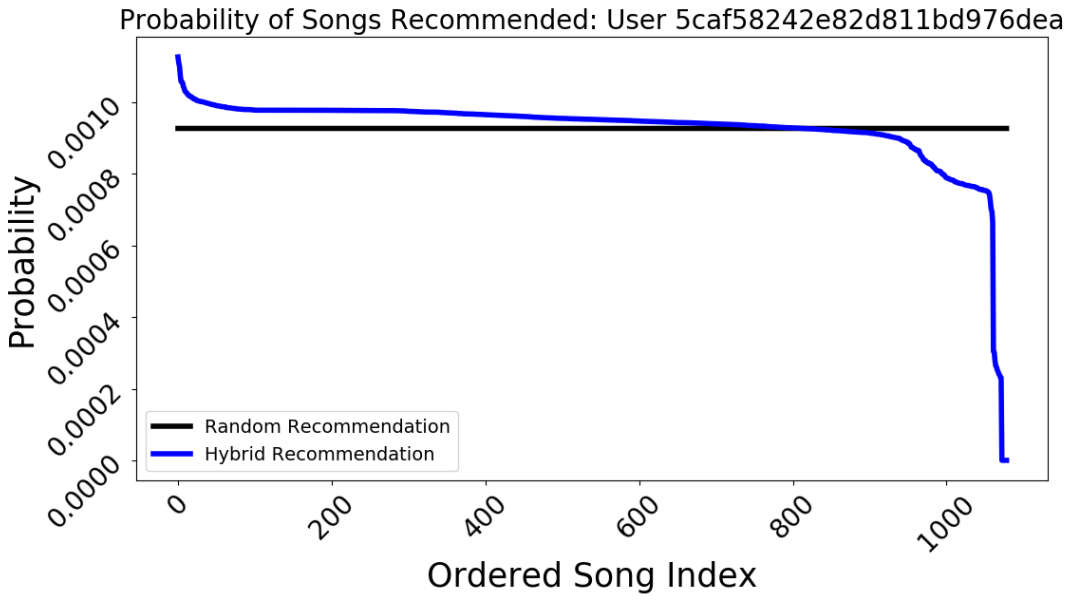

Figure 4.15: Probability distribution of songs recommended given test session of specific user

Finally, we can try to understand how much the two different parts of the hybrid recommender system contribute to the final result, namely the collaborative-based and the content-based components. This provides some intuition to which extent both system components are actually used.

Fig. 4.16 plots the rating score for the top-ranked song for each test session across the different users. It shows the proportions that the collaborative-based and content-based recommender parts have contributed to the final rating score. We can see that in all cases apart from one, both components were present.

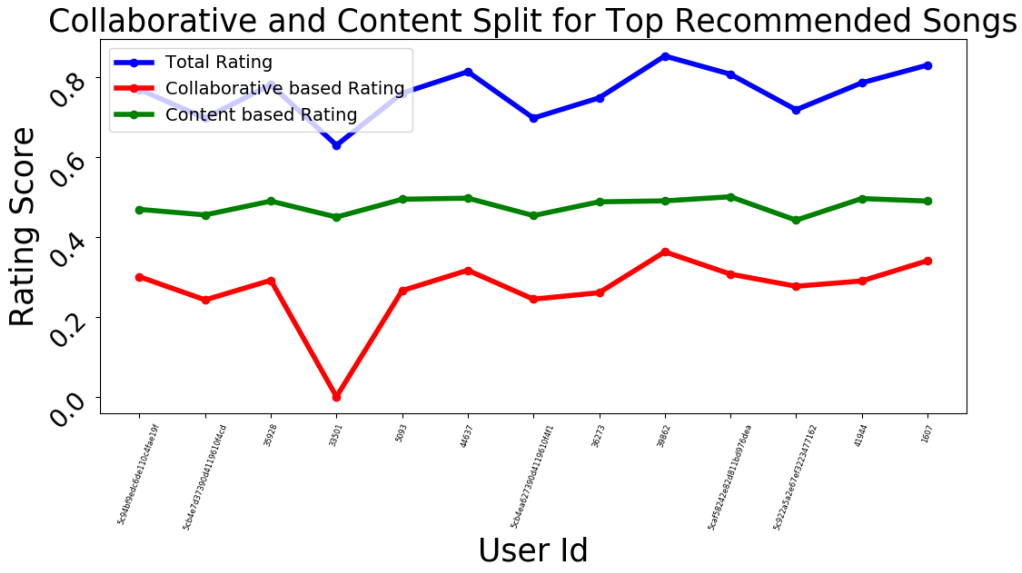

Figure 4.16: Attribution of total rating score for the top-recommended song to collaborative-based and content-based parts of the solution.

4.3 Functional Testing and Failure Cases

Our implementation of the music streaming service works in normal circumstances.

Failure cases:

1. In case the Spotify API is not live, users of the service are not able to access any songs. Our solution is currently not mitigating against this possibility.
2. An unexpected downtime of the Last.fm API doesn't impact the user of our streaming service, however, it could temporarily impact any further, ongoing development of the recommendation system.
3. The streaming service is not robust against unorderly user behaviour in regards to submitting completed music listening sessions to our streaming service. Sessions submitted by a user are added to the historical data, which in turn is used for model training. Thus, effects of spam or inappropriate use of the system (e.g. adding completely random songs to an ongoing session and submitting the session) could negatively affect the data set and have a negative impact on the recommendations the system produces.
4. Insufficient testing has be done so far related to the front-end system to ensure that all errors that might occur in the asynchronous communication with the backend system would be captured and handled.

4.4 Evaluation of Options and Decisions Within the Project

In hindsight, we can comment on various decisions made in the project, which relate to different options that were available.

1. Selecting the dimensions for the latent factors of the Funk SVD model
   There is no correct dimension per se, however, there will be different ways to fine-tune this parameter. In our implementation, we arrived at the value K=10 through observing the quality of the recommendations produced, e.g. through studying the types of graphs mentioned above. However, we were unable to do more extensive testing, which might have changed the value for a possibly

more optimal representation. Care would have to be taken to avoid overfitting of the Funk SVD model.

2. Selecting the algorithms for the content-based recommender part
   We built the content-based component of the hybrid recommender system by using the Word2Vec neural network as a key component. We arrived at this decision, as Word2Vec is a common technique to compute semantic similarities, by using vector space models. However, other alternative methods could be used. Word2Vec also works with a latent space, and we decided to use a dimension of 100, as this is the standard/default value in the libraries we used. This parameter would require more extensive testing to explore its impact on the overall recommendations.

3. Selecting the algorithms for the collaborative-based recommender part
   Earlier in the project, we explored the use of Hierarchical Dirichlet Process (HDP), in order to replace (the finally selected) Latent Dirichlet Allocation. HDP doesn't require to set the number of topics as a hyperparameter. As the topics in our music streaming service are relatively abstract, HDP would have produced the optimal number of different topics by itself. However, HDP didn't perform well. Nevertheless, this option could be explored more thoroughly in the future.

4. Selecting the number of latent topics in the Latent Dirichlet Allocation (LDA)
   We chose 16 latent topics, based on visual inspection of the tags within each topic after performing a coherence measurement. We haven't undertaken a brute force search for a potentially more optimal number of latent topics for LDA.

5. Decisions related to the system architecture
   We had the option to do more or less of the model training online or offline. For example, in the current implementation, the matrix factorisation will not be rerun every time the user asks for a new recommendation during an ongoing session. This implies, that the current session is not incorporated into the rating matrix factorisation within the collaborative-based part of the recommender system. The reason is that we felt the expected time required to calculate the matrix factorisation would be too long for a good user experience in particular in case of large data sets (lots of users or a large universe of songs). However, we could have used Amazon AWS services like SageMaker to potentially solve that challenge.

6. Decision related to the sourcing of music songs
   Initial development involved the creation of a database that contained actual mp3 audio files stored on Amazon AWS S3. This decision was later revised and we changed over to using the Spotify API. The upside was that the Spotify API offered a very large potential song library, which means that our system could now be easily expanded. However, the downside for now is the restriction to 30 seconds song previews instead of full-length songs, at least in the case of free use of the API.

7. Decision related to how to extract information from music songs
   Instead of using music song tags as from Last.fm and a mechanism like LDA to determine the latent topics, a potentially more precise method could be used through the extraction of multiple audio features from songs, where the topics could then be defined through an unsupervised classification task. We believe this would remove uncertainties, inaccuracies and inconsistencies related to the tags which are created by people. Audio features extracted from songs could include information about instruments, audio spectrum, pitch etc. We found that such information is rarely found in crowd-sourced tags as available from Last.fm. On the other hand, human/user-created tags sometimes provide information about user emotions and this is equally valuable. The problem of automatically describing music and songs in terms of emotions is an active area of research, called Music Emotion Recognition. Therefore, a mixture of techniques to extract information from music songs could have signficiant potential.

8. Decisions related to privacy Due to the resource and time constraints in this project, the decision was made to leave out features that would otherwise cater for privacy related regulation. The music streaming service implementation holds private information about users. First, the system enables them to create an account. Second, the system analyses their music listening history and their current session to provide a personalised recommendation. A regulator may consider such a recommendation as private information.On 25 May 2018, the General Data Protection Regulation (GDPR) came into force in the EU [2]. As a commercial service operated from the UK, the session-aware music streaming service would have to be compliant with this regulation. Some of the key impacts would be as follows:

- The streaming service would have to ask users for consent to process personal data. If the data processed were considered sensitive personal data, a mechanism for requesting explicit consent from users would have to be implemented.
- Minimum requirements would apply to how the request for consent issued to users is articulated (e.g. clear, plain language, intelligible, easily accessible). In addition, the system would have to let users easily withdraw their consent.
- Privacy by design, which advocates the inclusion of data protection from the start into the design of systems, rather than as an afterthought, is becoming a legal requirement as part of GDPR. Thus, the solution design would have to cater for privacy right from the outset.
- The music streaming service would have to provide users (data subjects) with a mechanism to access a raw copy of their personal data and additional information like who has received such data.

Adhering to GDPR and any similar regulation in other regions or countries will be important for two reasons: First, to avoid significant penalties and fines for GDPR non-compliance and second, to avoid reputational damage. As recent complaints about Spotify, Apple, YouTube and Amazon demonstrate, achieving full compliance may not be that simple [9].

Chapter 5

Conclusion

5.1 Main Contribution and Achievements

The aim of this project was to implement a prototype music streaming service that includes a novel recommendation system for music songs. In particular, our intent has been to design and build a recommender system that allows for interactivity with users and that is session-aware in contrast to just session-based. Currently there is a growing research interest in exploring session-based/aware recommender systems in general to provide online users with more relevant results.

The main achievements are as follows:

1. We have built a fully functioning music streaming service hosted on Amazon cloud infrastructure. This service allows for the playback of any songs (in preview form) found in its database and the discovery of songs through both search and recommendation functionality. Users can consume this service through a web browser. The web application is intuitive, responsive and professionally designed.

2. As a core building block of the music streaming service, we have designed and implemented a hybrid session-aware recommender system. This is a system that uses historical past online sessions where a user consumed music songs in one sitting. The system uses this past information about many users to make relevant recommendations for a user's current online session. The hybrid system consists of two components and combines them in an innovative way. The first one is a collaborative-fitering-based recommender part and the second one is a content-filtering-based recommender part. This hybrid mechanism has been subjected to behavioural testing and analysis with the goal to maximise the performance of the recommender system.

3. We have devised a solution to create a collaborative-filtering-based recommender component that is aware of users' historical music sessions. We subject all past user listening sessions with their songs and specifically the song tags to a process that identifies a number of latent topics present within this universe of tags. For a user, who is currently engaged in an active, ongoing music session, we identify similarities between the historical music listening sessions of this user and those of all other users covered by the system's database. The collaborative-filtering-based component makes a prediction about how a user would have rated a song, based on how the user's past music sessions relate to music sessions of other users.
   To realise the discovery of a fixed number of latent topics across all music song tags, we create a Latent Dirichlet Allocation model. To perform collaborative filtering, we have implemented a Funk SVD matrix factorisation approach.

4. In addition, we have designed and implemented a content-filtering-based recommender component. This approach takes as input the song tags of the current, ongoing music session of a user, and the tags of any new candidate song that could be recommended. We calculate the degree of similarity between the session tags and a candidate song's tags in a latent space which we in turn create from the tags of historical music sessions through a Word2Vec neural network.

5. We realised a degree of interaction between users and the recommendation system part within the

music streaming service with the vision to further improve the performance of the system based on user feedback. Song tags are assigned to songs within the current session which users can both observe and remove. This gives them a greater understanding and level of interaction with the system.

6. From the evaluation stage, we can conclude that the overall implementation has some potential, as it produced promising recommendations. This is derived from quantitative analysis involving tests and from qualitative user feedback.

5.2 Current Status of Project

The project has been completed in the time frame with all of the objectives being addressed to a good degree. From the initially considered and planned capabilities of the system, we believe that about 90% of the intended features have been implemented.

However there are a few elements which could not be fully explored as initially envisaged.

A shortcoming is the lack of training data on which the recommendation system models were trained on. Unfortunately during the development process, the Last.fm topTag API broke globally for around 3-4 weeks. This had some impact on the testing process and the final results produced. With a much larger amount of session data, we expect that the LDA model and Word2Vec model would produce more refined results. For instance, the number of "session topics" could be in the number of hundreds. We believe this could produce more focused recommendations.

The other shortcoming relates to the amount of user interaction currently possible through the system. The initial plan was for users to edit, modify and add any specific tags to their recommendation search to get an even more refined result. For instance, if a user wanted to carry on the session "vibe" but wanted more of "guitar", they could do this through a form of interaction with the system.

Overall, the implemented system is a good platform through which other music recommendation systems and algorithms could be explored in the future.

5.3 Open Issues and Future Work

Based on the results from the evaluation phase, future work could address the following areas.

Regarding the actual streaming service, one area requiring improvements would be the process of how users discover songs themselves. Currently they scroll a list of songs available. This is not a very time-efficient means of discovering new songs in particular if the library of songs gets very large. Instead, some categorisation such as genres or popular playlists could be constructed and added to the user interface.

An area that would require focus is the general architecture of the system. As has currently been implemented, both the online and offline parts of the recommendation system would not scale well to millions of users. Improvements could involve hosting Apache Spark on a Hadoop cluster manager. Using the distributed MLlib framework would produce a more scalable solution. This in combination with Cassandra as the distributed storage could be a more optimal solution.

Working on this project has emphasised the question of whether song tags from Last.fm are both accurate and descriptive enough to allow creation of good recommendations. Last.fm tags may not be explicit enough to enable separation or clustering of songs. To illustrate an extreme case: If 300 songs out of 1000 had nearly identical tags assigned to them, differentiation amongst those songs would be nearly impossible. This issue is down to the open-source nature in which the tags tend to be created. Our recommender system's performance is very much at the mercy of the quality of the song tags. Due to only a small subset of users and songs being tested, further exploration using these tags could be worthwhile. This might be a reason why commercial music streaming services like Pandora employ music experts to manually curate the descriptive song tags.

The recommendation system has currently some static nature to it that could be addressed. Assume a user uses the recommendation functionality and adds the top recommended song to their session. If the

user then asks for a further recommendation, the top results could largely remain unchanged. In this case, there might be too much exploitation and too little exploration.

One solution to this could be that the user gives instant feedback to the recommendations such that active learning can occur over time and the degree of exploration can be adjusted. A reinforcement learning technique could seek to eliminate this problem [34]. An alternative is that the system automatically increases exploration itself, if it has a particular objective. Such an objective could be to help the discovery of unknown artists. Another example of such a system objective could be to encourage user exposure to different music genres or to related artists.

Other future studies could involve further expansion using contextual factors that may affect the user's choice such as location, time, the weather or their current activities and integrate these into the system.

In summary, this project has seen very encouraging results for using session-aware recommender algorithms in music streaming services. It demonstrates viability to enhance the user experience and suggests suitability for using such techniques in future streaming services.

Bibliography

[1] 30music dataset. http://recsys.deib.polimi.it/datasets/.

[2] Gdpr. https://eugdpr.org/.

[3] Gensim python package. https://radimrehurek.com/gensim/index.html.

[4] Lastfm api. https://www.last.fm/api.

[5] Million song dataset. https://labrosa.ee.columbia.edu/millionsong/.

[6] Patent: Collaborative recommendations using item-to-item similarity. https://patentimages.storage.googleapis.com/41/80/fb/07d4d9e61e7431/US6266649.pdf.

[7] Surprise python package. https://surprise.readthedocs.io/en/stable/index.html.

[8] Word2vec tutorial part i: The skip-gram model. http://alexminnaar.com/2015/04/12/word2vec-tutorial-skipgram.html.

[9] Major streaming services violating gdpr, says privacy ngo. https://globaldatareview.com/article/1179393/major-streaming-services-violating-gdpr-says-privacy-ngo, 2019.

[10] Charu C. Aggarwal. *Recommender Systems*. 2006.

[11] Adam Berenzweig, Beth Logan, Daniel P.W. Ellis, and Brian Whitman. A large-scale evaluation of acoustic and subjective music similarity measures. 2004.

[12] John Brooke. Sus - a quick and dirty usability scale. 1986.

[13] Keunwoo Choi, Gyrgy Fazekas, and Kyunghyun Cho. A tutorial on deep learning for music information retrieval. 2018.

[14] Ricardo Dias and Manuel J. Fonseca. Improving music recommendation in session-based collaborative filtering by using temporal context. 2013.

[15] Andres Ferraro, Dmitry Bogdanov, and Jisang Yoon. Automatic playlist continuation using a hybrid recommender system combining features from text and audio. 2018.

[16] Chase Geigle. Inference methods for latent dirichlet allocation. 2016.

[17] Carlos A. Gomez-Uribe and Neil Hunt. The netflix recommender system: Algorithms, business value,and innovation. 2015.

[18] Balazs Hidasi, Linas Baltrunas, Alexandros Karatzoglou, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. 2016.

[19] Shyong K Tony Lam, Dan Frankowski, and John Riedl. Do you trust your recommendations? an exploration of security and privacy issues in recommender systems. 2006.

[20] Xuan Nhat Lam, Thuc Vu, Trong Duc Le, and Duong Anh Duc. Addressing cold-start problem in recommendation systems. 2008.

[21] Jin Ha Lee. How similar is too similar?: Exploring users' perceptions of similarity in playlist evaluation. 2011.

[22] T. Adilakshmi M. Jaya Sunitha. Session aware music recommendation system with user-based and item-based collaborative filtering method. 2014.

[23] Sung Eun Park, Sangkeun Lee, and Sang goo Lee. Session-based collaborative filtering for predicting the next song. 2011.

[24] Martin Pichl, Eva Zangerle, and Gunther Specht. Towards a context-aware music recommendation approach: What is hidden in the playlist name? 2015.

[25] Auste Piliponyte, Francesco Ricci, and Julian Koschwitz. Sequential music recommendations for groups by balancing user satisfaction. 2013.

[26] Quentin Plepl. Topic coherence to evaluate topic models. http://qpleple.com/topic-coherence-to-evaluate-topic-models/, 2013.

[27] Massimo Quadrana, Paolo Cremonesi, and Dietmar Jannach. Sequence-aware recommender systems. comput. 2018.

[28] Markus Schedl, Hamed Zamani, Ching-Wei Chen, Yashar Deldjoo, and Mehdi Elahi. Current challenges and visions in music recommender systems research. 2017.

[29] Jan Schlter and Christian Osendorfer. Music similarity estimation with the mean-covariance restricted boltzmann machine. 2011.

[30] Carson Sievert and Kenny Shirley. pyldavis. https://github.com/bmabey/pyLDAvis, 2014.

[31] Yee Whye Teh, Michael I. Jordan, Matthew J. Beal, and David M. Blei. Hierarchical dirichlet processes. 2005.

[32] Nava Tintarev, Christoph Lofi, and Cynthia C. S. Liem. Sequences of diverse song recommendations. 2017.

[33] Shoujin Wang, Longbing Cao, and Yan Wang. A survey on session-based recommender systems. 2018.

[34] Xinxi Wang, Yi Wang, David Hsu, and Ye Wang. Exploration in interactive personalized music recommendation: A reinforcement learning approach. 2014.

[35] Bo Zhang, Na Wang, and Hongxia Jin. Privacy concerns in online recommender systems: Influences of control and user data input. 2014.