Creating a Music Recommendation and Streaming Application for Android

Elliot Jenkins[1] and Yanyan Yang[2]

[1,2]School of Engineering, University of Portsmouth, Portsmouth, UK

[1]elliot.jenkins@myport.ac.uk
[2]linda.yang@port.ac.uk

Abstract. Music recommendation and streaming services have grown exponentially with the introduction of smartphones. Despite the large number of systems, they all currently face a lot of issues. One issue is a cold start where a user who is new to the system can't be made recommendations until the system learns their tastes. They also lack context awareness to make truly personalised recommendations to the user. This paper introduces a new recommendation and streaming application, individual Personalised Music (iPMusic), for Android which is specifically designed to address the issues. We examine the effectiveness of iPMusic based on real world users' feedback which shows positive results.

Keywords: Music Retrieval · Mobile Computing · Recommendation System · Music Streaming

1 Introduction

With the continual evolution of personal media players from Sony Walkman's, to portable CD players, MP3 players and eventually the Apple iPod it is clear that individuals like to listen to music wherever they are. This is becoming an area that smartphones are largely moving into to help cope with the huge demand for music. The amount of music being released has also been growing at ever larger rates with multiple services such as Google Play Music and Deezer having in excess of 30 million songs [1]. With such huge libraries of music it is possible for users to become lost and struggle to find new music that they like. As new songs are released users may not be informed so it becomes problematic for them.

To help the users, recommendation systems have been created making use of content-based and collaborative algorithms. There are however issues with these current recommendation algorithms, those used by Spotify [2] and Google Music [3] require the user to first of all listen to music before any recommendation can be made to them. The issue with this is if a new user joins then no recommendation can be made to them. Similarly if a new song is added until users listen to it and rate it, collaborative methods would rate the song lowly so it is unlikely to be recommended. A hybrid approach was put forward by Wang et al [4] that makes use of the user's context such as location and listening history and combines this with content based methods to overcome the issue of not providing recommendations to new users. By taking the user's location into account it creates a more personalised recommendation compared to Spotify or Google Music but this can be further improved so that every recommendation is unique to that individual user. This provides an interesting challenge for developers to solve as the system needs to be able to cope with new users being added and to provide them with personalised recommendations from the beginning but also when a new song is added it is considered fairly when making recommendations.

The motivation for this work has been to provide users with an easy way to discover new music. We have developed an algorithm that will provide unique personalised recommendations based on the users' Twitter posts as well as their listening history. The system will pull song lyrics and other fields to generate a list of similar songs when creating the recommendation. Collaborative methods have been developed to find similar users in the system and to enhance the recommendation based on what similar users like. This hybrid approach combining information from a variety of sources will produce a more accurate and personalised recommendation to the user on the individual Personalised Music (iPMusic) App. It will then be possible for the user to stream the songs in the app or to play the music video through YouTube. At the time of writing, we are unaware of any music service that can produce recommendations without any prior knowledge of personal listening history.

The rest of the paper is organised into the following sections; in section 2, we discuss the problem in more detail and related work that has already been carried out. Section 3 introduces the proposed system and real world usage scenarios. In section 4, we describe the architecture and introduce the main components in more detail. In section 5, we discuss the current implementation of the system and evaluate the findings from user feedback. We conclude in section 6.

2 Problem Description and Related work

The current issue with the music industry is there is a huge quantity of songs available to the user. The largest library of music is provided by Apple Music [1] which has in excess of 43 million songs. Work carried out by Nathan et al [5] suggested that the average song length is 226 seconds which means to listen to 43 million songs would take 308 years. This is not possible and many users wouldn't like all of the music available to them so the music services have begun to provide the user with recommendations. Despite that when making recommendations they face an issue known as the cold start. The cold start issue is where a new user is added to the system so has no known tastes meaning a recommendation can't be made to them. So many current services require the user to start searching and playing music that they like before any recommendations can be made to them. A further weakness is that recommendations usually do not include any user context so are not truly personalised to the user. Another issue is when a new song is added the user may not be made aware of this. This is because collaborative methods take into account song ratings and how frequently they have been played, meaning a new song would have nothing for both most likely resulting in it not being recommended to the user.

Su et al [6] propose a prototype music recommendation system that is designed to make use of the user's context and context information mining to offer recommendations that will suit the listener in their current situation. This would provide a personalised recommendation in a way no other system can by making inferences from patterns detected by the context miner. Despite providing better recommendations than existing systems there is still the major issue of a cold start that has not been addressed. Due to the nature of the algorithm a context log is required which describes multiple conditions about the user at different times of the day which would need to be collected prior to making a recommendation.

Narayanan et al [7] presented a collaborative method making use of K-Nearest Neighbourhood (K-NN). This method allows predicting what one user will like based on another user who is similar to him. Although not directly solving any of the issues we are looking to overcome, the idea can be used to further enhance the accuracy of the recommendations as proven by Narayanan et al work.

Adomavicius et al [8] presented many approaches to recommendation systems detailing any advantages or limitations. The main issue highlighted is the cold start issue and their solution to solve it is using a hybrid approach of collaborative and content-based methods. However they discuss different hybrid approaches such as using the two methods separately and carrying forward the recommendation that is the most accurate. This approach can therefore remove any personalisation of the recommendation if just content-based methods are used.

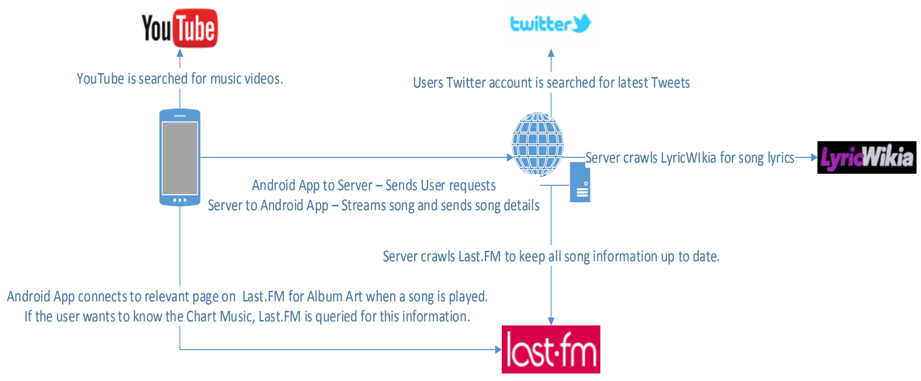
Wheal et al [9] developed CSRecommender which provides recommendations for different cloud based services that are currently available. Wheal's approach uses a hybrid recommender making use of collaborative methods taking into account the user and similar users and combining this with a content-based method that finds a similar service. By making use of both approaches it allows for an accurate recommendation to be made to the user. Despite recommending cloud services a similar approach can be taken to make song recommendations.

Twitter Music [10] is a system that pulls music from iTunes, Spotify, Rdio and Vine and then presents the best new music that is trending on Twitter. The recommendations being made by the service are not personalised and are instead based on the entirety of Twitter users. This is a reliable method to find what the most popular music is and is the only system to alleviate the cold start issue for a new user. However if a new song has been released and it doesn't trend on Twitter then it won't be recommended so it faces the same issue as the other systems.

This research work addresses the aforementioned issues by creating a hybrid approach that will take into account the majority of above methods in a unique recommendation algorithm. By using context information collected from Twitter combined with a K-NN approach the cold start issue can be addressed whilst offering a high level of personalisation.

3 System Overview

The system is based on a client server architecture which both communicate with one another as well as external sources which is shown in Figure 1. It is possible for multiple clients using the iPMusic App to connect to the Server simultaneously and each will be handled by their own thread.



YouTube
YouTube is searched for music videos.
twitter
Users Twitter account is searched for latest Tweets
Server crawls LyricWIkia for song lyrics
LyricWikia
Android App to Server – Sends User requests
Server to Android App – Streams song and sends song details
Server crawls Last.FM to keep all song information up to date.
Android App connects to relevant page on Last.FM for Album Art when a song is played.
If the user wants to know the Chart Music, Last.FM is queried for this information.
last·fm

Fig. 1. Overview of the System

There are two phases that occur on the Server, initially a setup phase runs before users can connect and make requests. The setup phase crawls Lyric Wikia and Last.FM to create an up to date index for the initial library of music. Once the information has been collected the Server exits the setup phase and allows connections from the iPMusic App. The Server now remains in this stage so it is necessary for it to detect new music being added to the system and to obtain the information for new items and to correctly add them to the index.

From the users' point of view, they will start off by downloading iPMusic from the Google Play Store. Once downloaded and installed they can create an account which will then allow them full access to the Application. They will have the ability to see what music is in the current charts, to get a list of recommendations, search for a song and to display their favourite music to play back at any time. The following real world scenarios are an indication of how the system can be used:

- A new user may be wanting to discover new music so use the "Play me something" button. This would present the user with a list of uniquely generated recommendations and the ability to play any of these back. On playing a song and the user providing a rating it further improves the accuracy of future recommendations.
- The system has the capability to generate a unique and personalised recommendation created just-in-time so users may be wishing to take advantage of this feature that is not offered by other systems at such a personalised level.
- Although not the primary purpose, the search feature in the app not only searches song titles, artists and album names but also the lyrics. Therefore meaning if the user knows the lyrics of a song but is unsure what it is called then they can find out via the app.

4 System Architecture and Design

An overview of how all of the components on the server and client side will communicate is detailed below in Figure 2.

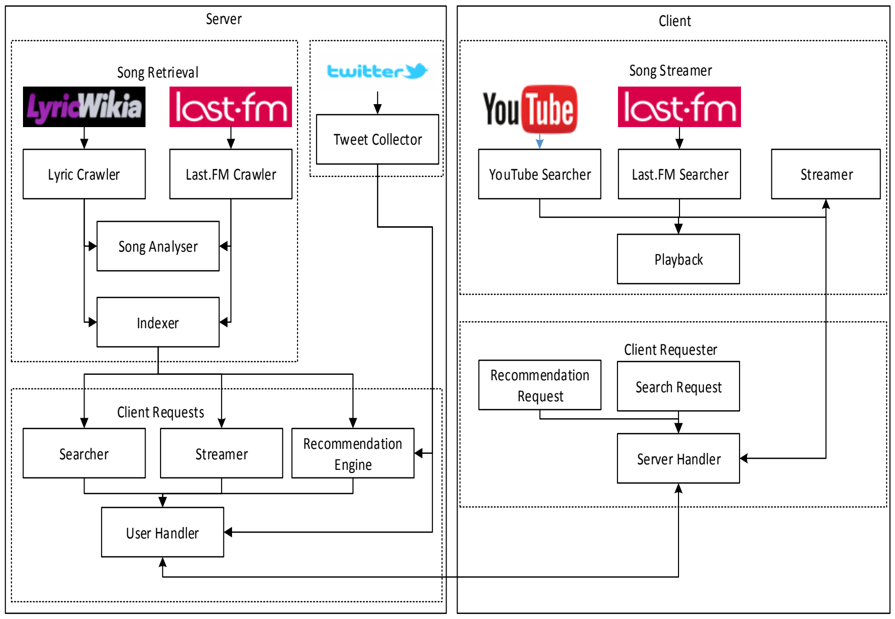


Fig. 2. Block Diagram of Server and Client

4.1 Server Side

Last.FM Crawler. The purpose of the Last.FM crawler on the server side is to maintain an up to date list of all the music files in the system. When reading the tag fields from an .mp3 file it is possible some information is missing or it is wrong due to being entered incorrectly. Last.FM provides an API to access their different services and the crawler will make use of track.getInfo which returns the metadata for any given track. In doing this it will allow the recommendations being made to be more accurate since the tracks will contain more details such as release dates and album that may otherwise be missing.

Lyric Crawler. The main part of the recommendation algorithm is based on the song lyrics so it is necessary to find all of the song lyrics for the tracks in the system. LyricWikia provides lyrics for 1,798,797 different songs, so it is likely that any song searched for produces a result. So the purpose of the lyric crawler is to obtain the lyrics for every song in iPMusic. It is possible a song has no lyrics such as instrumental but it will remain in the system as the song title and album names could still hold relevance to a recommendation. When searching for song lyrics the sitemap will be queried for the current song and if there is a match the crawler will connect to the matching URL. Once connected to the web page the content will be parsed by Jsoup and the lyrics will get extracted and stored in the index. The process is highlighted in algorithm 4.1.1.

Tweet Collector. Twitter has over 320 million monthly users so there is a high probability that the user will have a Twitter account. To get a more personalised recommendation their Tweets can be collected and indexed. The Twitter API provides an easy way to retrieve the Tweets of an individual user given their username and a time based context. This means it is possible to restrict the Tweets to only those that are from the past day as this would more accurately reflect the user's current state of mind. If however the user has not posted any Tweets in the past day then the system will revert back to any Tweets within the past week or month if necessary. When the Tweets are indexed the words are kept in their raw form since stemming the words can totally change the meaning when then being compared to song lyrics. Any numbers in the Tweets will also be indexed as they can be related to song titles or lyrics. If a user does not have a Twitter account then this stage is not possible so there is the extension of integrating iPMusic with Facebook at a later stage which would follow the same process.

Song Analyser. When making a recommendation based on a user's listening history it is necessary to know what songs are similar to those that they have listened to. The Song Analyser determines the similarity between all songs in iPMusic so the most similar songs can be taken into account when producing the recommendation.

A song is made up of the 5 following unique fields; Lyrics, Artist, Album, Title and Release Year. By using different weightings for each field it can be determined how similar one song is to another. The weightings used by the Song Analyser are shown below. These weightings were determined following the use of experimental weightings until the yielded results were liked by a set of users.

- 55% - Lyric Similarity
- 20% - Artist or Album matching
- 20% - Title Similarity
- 5% - Same release year

The process to generate a list of similar songs starts by using cosine similarity to compare the similarity of the song lyrics which is then multiplied by the weighting of 55%. If the Artist or Album match then the score is increased by 20%. The titles of the song being compared against is split into individual terms and the similarity of the two titles is calculated by seeing how many times the individual terms appear in the other title. For every match a counter is incremented and this can then be converted to a percentage for the overall similarity which is then multiplied by a weighting of 20%. The remaining 5% comes from the year that the songs were released as it may have some relevance to the recommendation. The formula to calculate similarity is shown in Equation 2. The list of similar songs is then sorted into descending order and stored in the index.

Cosine $Similarity(A, B) = \frac{A \cdot B}{|A||B|}$ (1)

Song Similarity whereby LS is Lyric Similarity and TS Title Similarity

$$Son\ Similarity = (LS \times 0.55) + if(Album\ or\ Artist\ Match, 0.2, 0) + (TS \ \times 0.2) + if(Years\ Match, 0.05, 0) \quad (2)$$

Indexer. With all of the data gathered by the above methods it is necessary to index the data so it can be quickly queried and provides results to the user in the shortest amount of time. The index will be made up of the following fields:

- The Song and Artist ID's will allow for fast identification of songs without the need for searching for titles and artists and finding a match. Instead the system will be able to go the nth Artist and to that artists nth Song therefore greatly increasing the speed and efficiency of the system.
- The Song Title, Artist and Album names the Term Frequency (*TF*) and Inverse Document Frequency (*IDF*) will be calculated using Equation 3 and 4 respectively. From this the *TF-IDF* can be calculated for each term in the fields and this will be stored as a posting in the Inverted File Index.

$$TF = \frac{Number\ of\ times\ t\ appears\ in\ a\ document}{Total\ number\ of\ terms\ in\ the\ document} \quad (3)$$

$$IDF = \log \frac{Total\ number\ of\ documents}{Number\ of\ documents\ with\ term\ t} \quad (4)$$

- The Release Date too but this can be in a variety of different formats such as an English date of DD/MM/YYYY compared to American format of MM/DD/YYYY. The date may need to be changed so that all of the formats are the same allowing for quicker comparisons when creating a recommendation.
- The Lyrics will be stored using TF-IDF but also the document vectors will be kept which will allow for getting the cosine similarity between two songs.
- The file location will indicate where the song is stored on the iPMusic Server so that the song can be quickly streamed to the smartphone. This allows for scalability as it is possible for songs to be stored in more than one location.
- The Similar Songs List will be generated by the Song Analyser and is stored in descending order to quickly find the most similar song.

Recommendation Engine. The recommendation engine goes through the following series of stages each of which are explained further below.

- K-Nearest Neighbour – 25%
- Twitter – 25%
- Similar Songs – 50%

If the user's listening history is not empty the first stage is to use a collaborative approach using K-Nearest Neighbour. A user is considered a nearest neighbour based on the similarity between the users listening history and the neighbours taking into account their ratings as well as play count. Since nearest neighbours will have similar interests in music it can be assumed that if neighbour N likes song A then user U will also like song A so a recommendation can be made based upon this.

Algorithm 4.1.1 K-Nearest Neighbour

```
Get vector for User U
while there are Users to compare Uc do
  Get vector for Uc
  Calculate Cosine Similarity for U and Uc (Equation 1)
  Set counter and matches to 0
  while there are songs S in Uc Listening History do
    if S is in U listening history then
      Increment matches
      if ratings for songs U and Uc are above 3 then
        Increment counter
      end if
    end if
  end while
  Calculate rating similarity from counter / matches
  if rating similarity > cosine similarity then
    Add user to nearest neighbour list
  end if
end while
Sort the nearest neighbour list
for the top 5 nearest neighbours do
  While there are Songs S in listening history do
    if S rating is 5 then
      Add S to recommendation with score of 25
    else if S rating is 4 then
      Add S to recommendation with score of 20
    else if S rating is 3 then
      Add S to recommendation with score of 15
    end if
  end while
end for
```

The next step is to find songs similar to those that the user has already listened to. In the index the list of similar songs can be used to identify those that are similar to what the user has already listened to. This however creates an issue of how to handle the score if two or more songs have a similar song in common. If the scores were to be added then the results would be skewed and songs that aren't that similar could get higher scores than those that are. Or if the average was taken any outliers would bring the score down. So a method was designed to combine the scores without skewing the results as shown in algorithm 4.1.2 by removing any scores outside the standard deviation of the average.

Algorithm 4.1.2 Similar Songs

```
while there are songs S in listening history do
  for the top 10 similar songs SS do
    Add SS to recommendation list
    Add variance to list of variances in recommendation
    list using equation 5
    Keep running total of variance squared and sum of
    variances for recommendation
  end for
end while
while there are recommendations R do
  Calculate standard deviation from equation 6
  Remove any variances below average minus standard devi-
  ation
  Set score to average variances
  Add weighting onto score from equation 7
end while
```

$$Variance = Similar\ Song\ Score\ - Average\ Song\ Score \tag{5}$$

$$Standard\ Deviation = \sqrt{\frac{\sum Variances^2}{Number\ of\ Variances}} \tag{6}$$

$$Weighting = \frac{100 - Maximum\ Score}{Maximum\ number\ of\ occurrences} \times Number\ of\ occurrences \tag{7}$$

Following this the top terms from the users Tweets will be searched for amongst the song lyrics. Any songs that match the search query are then added to the list of song recommendations with a score determined by their TF-IDF score. The maximum score a song can get from this stage is 25, so the TF-IDF scores are normalised to range from 0 – 100%. As previously mentioned each stage carries a different weighting towards the final score and for this stage it is 25%. This was determined through testing of different weightings until the recommendation best reflected the test users taste. With the normalised score now as a percentage the final score can be calculated by multiplying it by the weighting of 25%.

The final stage of the algorithm is optional and can restrict the songs to only those that have been released within the user's lifetime. The likelihood of the songs being restricted is calculated from the percentage of songs that have been released during the user's lifetime compared to those that have not. This means that the more music a user listens to that has been released during their lifetime the higher the likelihood of the restriction being put in place, if the majority of the music listened to is not in their lifetime there is a small likelihood of the restriction being in place.

Following all of these stages a recommendation can be made to the user and the top 10 are presented to the user for them to then pick which to listen to. If however the user has never listened to anything on the system then two of the stages are used to stop the cold start issue.

The K-NN approach is used again but instead focuses on the user's age. There is a high chance that if one user is the same age as another then they will like similar music. With the nearest 5 neighbours found all of their highest rated songs are added to the recommendation list.

The next stage using the same approach as before with Twitter. If the user does not have a Twitter account then this stage is skipped and the recommendations are given straight to the user. A future addition will be the integration of Facebook as well as other social networking platforms to further enhance the level of personalisation.

This hybrid approach combining K-NN with the content based methods using Twitter alleviates the cold start issue that other systems suffer from. As the system grows and the number of songs listened to increases the algorithm will be able to provide more accurate recommendations. If however there are no other users in the system and the user does not have Twitter then the algorithm would not work. So it will be necessary to populate the system with some default users that fit certain categories.

Searcher. The searcher will provide a fast way to search the index file and return back the most relevant results. The recommendation algorithm will make use of the searcher when searching lyrics and titles for Twitter keywords and will return back the most relevant results. Similarly if the user submits a search from the iPMusic App then the lyrics, artist, title and album will all be queried returning back the most relevant results. The relevancy of a result is calculated from multiplying the term frequency with inverse document frequency from the inverted index to get the TF-IDF weight w. The Vector Space Model is then used to rank the results whereby q is the term being queried in two documents.
Equation 8 - Vector Space Model whereby N is the number of results, $w_{i,j}$ is the weight given to the *ith* word in document j and $w_{i,q}$ the ith word in document q

$$Result = \frac{\sum_{i=1}^{N} w_{i,j} w_{i,q}}{\sqrt{\sum_{i=1}^{N} w_{i,j}^2}\sqrt{\sum_{i=1}^{N} w_{i,q}^2}} \tag{8}$$

User handler. The last component of the server is the user handler and its purpose is to service all of the requests coming from the iPMusic App. The requests will be sent over a TCP socket which then need to be handed to the correct part of the server. As there will be multiple users it will be necessary to first login and once logged in the user can request recommendations or search for songs. The server will then send back to the user the list of recommendations or search results.

4.2 iPMusic Client

YouTube Searcher. The purpose of the YouTube Searcher is to try and identify the correct music video for the song currently being played. This will then allow the user to watch the music video rather than just listening to the song. The searcher will use the YouTube API to create a query made up of the song name and the artist. This is likely to return the correct video but it is impossible to guarantee it.

Last.FM Searcher. When playing music, album art should be displayed which Last.FM hosts for the majority of songs. So the Searcher will be used to get find the album art and display it on the app when playing a song. The album art is then cached locally on the device so that it can be used again without downloading the image.

Streamer. The Streamer is responsible for downloading the songs from the server onto the client so that they can be played back. When a user requests a song to be played that song will be downloaded as well as the songs preceding and following it. This allows the user to either fast forward or rewind songs without having to wait for those songs to be downloaded. Prior to end of the current song being played the next song will be requested and downloaded ensuring it is ready to play with continuous playback.

Playback. The playback component is responsible for displaying the information provided by the Songs MP3 tags, the album art from Last.FM and playing the Songs MP3 file. It also needs to handle playing and pausing of songs as well as fast forwarding and rewinding tracks.

Recommendation Request. The recommendation request will ask the Server to produce a recommendation via the Server Handler, the response will contain a list of recommendations which will then be downloaded and stored on the client. These will then be displayed in panels on the iPMusic client so the user can select what to listen to first.

Search Request. The search request component will get the users search term from the input box and pass this onto the Server Handler. The rest of the component works in the same way as a recommendation request with Search Result in place of Recommendation.

Server Handler. The server handler will be responsible for communicating with the server. It will use TCP sockets to either send or receive messages. Like the user handler any messages will need to be given to the correct component such as Search Request so that it is handled correctly.

5 Implementation and Evaluation

At the time of writing, iPMusic has been released on the Google Play Store as a closed Alpha. Android was the platform of choice as following its introduction 8 years ago [11], it is estimated to have a 46.7% [12] share of 6,931,000,000 [13] active mobile phones, the largest of any platform. The Server and Android Client have both been written in Java and make use of the following open source Java API's (Application Program Interface): Apache Lucene, Jaudiotagger, Jsoup, Twitter4J and lastly YouTube. All of these API's provide services required by either the Server or the Android Application. A library of 1,200 songs has been imported into iPMusic and a small number of seed users have been created. The android app interface is shown below in Figure 3.

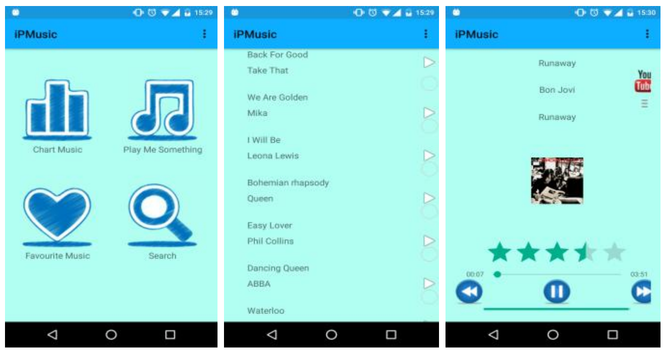


Fig. 3. Welcome Screen, Recommendations, Playback, Lyrics

Prior to being released as an Alpha on the Google Play Store internal testing was carried out by a selected group of users and by drawing comparisons between Last.FM [14] and iPMusic. This testing was very positive with the majority of recommendations being made being accurate and providing new recommendations to the user. When comparing the recommendations to Last.FM suggested songs there were some matches between the two. There are a few cases where the song and artist both matched from both systems but in more cases they both suggested songs from the same artists. A small sample is shown in Table 1. This highlights the fact that the recommendation engine works successfully. However due to the limited library compared to Last.FM a lot of songs suggested by Last.FM are not in the system so this is not a totally fair comparison.

Table 1. Yellow indicates matches of song or artist between both systems, Red shows songs not in our system.

Song Listened To	iPMusic Recommendation	Last.FM Recommendation
I Will Be – Leona Lewis	Better In Time – Leona Lewis The Best You Never Had – Leona Lewis Whatever It Takes – Leona Lewis Come In With The Rain Taylor Swift Here I Am – Leona Lewis	Here I Am – Leona Lewis Yesterday – Leona Lewis I Still Believe – Mariah Carey Beautiful – Christina Aguilera
Easy Lover – Phil Collins	If Leaving Me is Easy – Phil Collins Two Hearts – Phil Collins Wannabe – Spice Girls When You're Gone – Avril Lavigne	Two Hearts – Phil Collins Something Happened On The Way to Heaven – Phil Collins Invisible Touch - Genesis Land Of Confusion - Genesis

To further test the recommendation system users were asked to test the system and rank each song recommended to them and then whether or not the system provides something new and if they like the system or not.

For the first test each user requested N recommendations and would then listen to each of the songs being recommended to them and rank it out of 5 (1 dislike, 5 really like). This data was collected by the server and a sample from the data is displayed in Table 2.

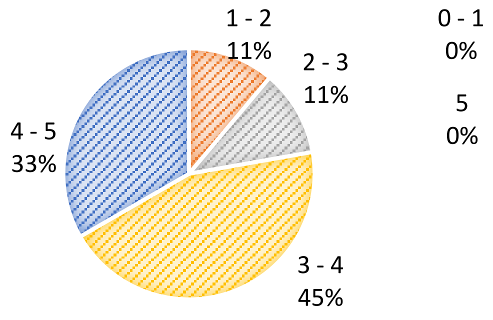
Table 2. Sample Recommendation Test Results

User ID	Number of Recommendations	Average Recommendation Rating	Average Rating
3	6	2, 2.4, 3.5, 4, 4, 3.5	3.23
4	5	0.8, 1.2, 1.3, 2.4, 3	1.74
8	5	3.6, 3.8, 4.3, 4.1, 4.5	4.06

The recommendation test results show that for most users the number of useful results started off low and as the system gained a better understanding of their tastes the recommendations being made to them improved. Despite that for user 4 the average rating was 1.74 which is really low, this could be down to the fact the library of music was limited so there were few songs the user liked.

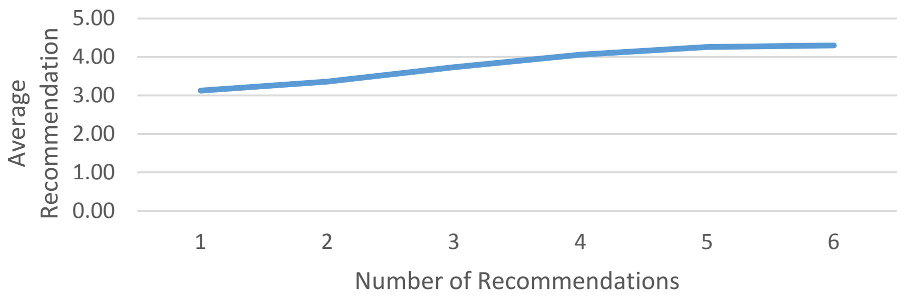
The average recommendation rating from 70 users is 3.4 which means the majority of the songs being recommended are liked by the users. 45% of the ratings ranged between 3 and 4 which is the highest percentage for any score, and was closely followed by 33% of the ratings being between 4 and 5. This shows positive results as the majority of the songs are highly rated by the users. Figure 4 shows a further breakdown of the scores and shows one area for improvement which is the fact that no user rated all 10 recommendations as 5 stars.

Fig. 4. Recommendation Results



However further analysis of the results also show positive findings. iPMusic was designed to use the listening history of its' users when making recommendations and as a result the accuracy should increase over time. This is proven in Figure 5 which shows that initially the songs being recommended are not as relevant as those suggested later during the users experiences. These results also show that the issues concerned with a cold start have been addressed since the average rating for the initial recommendations is 3.1. This is a satisfactory score showing that the users like the songs being recommended even when they are new to the system, if this score was lower, it would suggest the cold start issues had not been addressed.

Fig. 5. Relationship between Ratings and the Number of Recommendations



Following on to the next question, the user was asked whether the system provides something new. 60% said that the system provided something new and one of the positive comments was "The integration with Twitter provides me more personalised recommendations than I get from Spotify, if the library could be increased substantially then it would be a great system." However 40% believed that the system does not offer anything new. However a couple of the related comments stated these users did not have Twitter meaning the system couldn't personalise the recommendations so would be similar to how Spotify or Google Music make recommendations.

The final results show that the 80% of the users that have tested the system liked using it despite the shortcomings from the client being in the Alpha stage of development. One user stated "The app is really simple to use and the real time recommendations come really quickly."

6 Conclusion

The music recommendation system is unique compared to the other music streaming services as it alleviates the cold start issue and provides much more personalised recommendations. With the ever growing amount of music and increasing number of individuals with smartphones there will be a greater need for advanced recommendation algorithms and this satisfies that demand.

The system has become very complex and many improvements can still be made to further improve the recommendations. The immediate goal is to integrate the system with Facebook in addition to Twitter which will provide more contextual information about the user. With a better profile built up about each user the K-Nearest Neighbour algorithm can find better matches further increasing the accuracy of the recommendations. Other immediate goals are to fix any remaining issues within the Android Client and to ensure maximum compatibility across devices.

With a unique approach to identifying similar songs and making recommendations, the project has the potential to become a marketable solution if music licencing laws are taken into account. If the system was migrated to iOS it would provide an easy way for any smartphone user to discover and to listen to new music.

7 References

1. Software Insider, “Compare Music Streaming Services,” Software Insider, [Online]. Available: http://music-streaming-services.softwareinsider.com/#main.
2. Spotify, “Spotify,” Spotify, [Online]. Available: https://www.spotify.com/uk/.
3. Google, “Google Play Music,” Google, [Online]. Available: https://play.google.com/.
4. M. Wang, T. Kawamura, Y. Sei, H. Nakagawa, Y. Tahara and A. Ohsuga, "Context-Aware Music Recommendation with Serendipity Using Semantic Relations," *Lecture Notes in Computer Science,* vol. 8388, pp. 17 - 32, 2014.
5. N. Anisko and E. Anderson, “Average Length of Top 100 Songs on iTunes,” StatCrunch, 2012.
6. J.-H. Su, H.-H. Yeh, P. S. Yu and V. S. Tseng, "Music Recommendation Using Content and Context Information Mining," *IEEE Intelligent Systems,* vol. 25, no. 1, pp. 16 - 26, 2010.
7. S. Narayanan and V. Goswami, “K-Nearest Neighborhood based Music Recommendation System”. [Online]. Available: http://www.cs.ucsb.edu/~sivabalan/cs240a/CS240A_Project_Report.pdf.
8. G. Adomavicius and A. Tuzhilin, “Toward the Next Generation of Recommender Systems: A Survery of the State-of-the-Art and Possible Extension,” *IEEE Transactions on Knowledge and Data Engineering,* pp. 734 - 749, 2005.
9. J. Wheal and Y. Yang, “CSRecommender: A Cloud Service Searching and Recommendation System,” *Journal of Computer and Communications,* pp. 65 - 73, 2015.
10. Twitter, “#music,” Twitter, [Online]. Available: https://music.twitter.com/.
11. GSMArena, “T-Mobile G1,” GSMArena, October 2008. [Online]. Available: http://www.gsmarena.com/t_mobile_g1-2533.php.
12. Canalys, Androdlib, Gartner, “Android Phone Statistics,” Statisticsbrain, 17 March 2015. [Online]. Available: http://www.statisticbrain.com/android-phone-statistics/.
13. World Bank, “Mobile Cellular Subscribers / Global & US,” Statistic Brain, 17 March 2015. [Online]. Available: http://www.statisticbrain.com/mobile-cellular-subscribers-global-us/.
14. Last.FM, “Last.FM,” Last.Fm, 2015. [Online]. Available: http://www.last.fm/.