2014

A Hybrid Approach to Music Recommendation: Exploiting Collaborative Music Tags and Acoustic Features

Jaime C. Kaufman
University of North Florida, datrumpet5@gmail.com

A HYBRID APPROACH TO MUSIC RECOMMENDATION: EXPLOITING COLLABORATIVE MUSIC TAGS AND ACOUSTIC FEATURES

by

Jaime C. Kaufman

A thesis submitted to the
School of Computing
in partial fulfillment of the requirements for the degree of

Master of Science in Computer and Information Sciences

UNIVERSITY OF NORTH FLORIDA
SCHOOL OF COMPUTING

December 2014

The thesis, "A Hybrid Approach to Music Recommendation: Exploiting Collaborative Music Tags and Acoustic Features," submitted by Jaime C. Kaufman in partial fulfillment of the requirements for the degree of Master of Science in Computer and Information Sciences has been

Approved by the thesis committee: Date

_____ _____

Dr. Ching-Hua Chuan
Thesis Advisor and Committee Chairperson

_____ _____

Dr. Sherif Elfayoumy

_____ _____

Dr. Karthikeyan Umapathy

Accepted for the School of Computing:

_____ _____

Dr. Asai Asaithambi
Director of the School

Accepted for the College of Computing, Engineering, and Construction:

_____ _____

Dr. Mark Tumeo
Dean of the College

Accepted for the University:

_____ _____

Dr. John Kantner
Dean of the Graduate School

ACKNOWLEDGEMENTS

I express my deepest gratitude to my advisor, Dr. Ching-Hua Chuan for her exceptional guidance, technical expertise in the area of music informatics, patience, and considerable amount of time dedicated throughout this process. I would not have been able to realize the completion of this study without her direction. I would also like to thank my thesis committee members, Dr. Sherif Elfayoumy and Dr. Karthikeyan Umapathy for their support and expertise. Thank you to the University of North Florida School of Computing department for providing a server to host the web application I designed for this project. Special thanks to Scott Young, station manager for UNF Spinnaker Radio, for providing a large dataset of diverse music for use in my academic research.

I express warm appreciation to my friend and mentor, Leslie Blumenfeld, for his guidance, professional experience, compassion, and time. Additionally, I would like to thank Cliff Baumen, Bob Blackman, Aleksey Charapko and Joel Gottlieb, for their technical knowledge and time.

CONTENTS

FIGURES

TABLES

ABSTRACT

Recommendation systems make it easier for an individual to navigate through large datasets by recommending information relevant to the user. Companies such as Facebook, LinkedIn, Twitter, Netflix, Amazon, Pandora, and others utilize these types of systems in order to increase revenue by providing personalized recommendations. Recommendation systems generally use one of the two techniques: collaborative filtering (i.e., collective intelligence) and content-based filtering.

Systems using collaborative filtering recommend items based on a community of users, their preferences, and their browsing or shopping behavior. Examples include Netflix, Amazon shopping, and Last.fm. This approach has been proven effective due to increased popularity, and its accuracy improves as its pool of users expands. However, the weakness with this approach is the Cold Start problem. It is difficult to recommend items that are either brand new or have no user activity.

Systems that use content-based filtering recommend items based on extracted information from the actual content. A popular example of this approach is Pandora Internet Radio. This approach overcomes the Cold Start problem. However, the main issue with this approach is its heavy demand on computational power. Also, the semantic meaning of an item may not be taken into account when producing recommendations.

In this thesis, a hybrid approach is proposed by utilizing the strengths of both collaborative and content-based filtering techniques. As proof-of-concept, a hybrid

music recommendation system was developed and evaluated by users. The results show that this system effectively tackles the Cold Start problem and provides more variation on what is recommended.

Chapter 1

INTRODUCTION

1.1 Motivation

Thanks to the power of the Internet, recommendation systems have become part of everyday life for many users. If an individual has used Facebook, LinkedIn, or even Netflix, he or she has experienced a system that recommends new items based on various criteria. Amazon.com employs a system that recommends merchandise based on the user's browsing/purchase history and products purchased by other users with similar taste. Pandora and Last.fm are popular online systems that recommend music to users. These and other web-based music applications create revenue not previously in existence and are helping companies expand into other markets. Music recommendation systems help to fuel the digital music economy by assisting users in discovering music. According to the International Federation of the Phonographic Industry (IFPI), digital revenues for record companies in 2012 increased by an estimated 9 percent compared to 2011. This accounts for 34% of the total industry revenues [IFPI13].

1.2 Objectives

The two main methods used today in music recommendation systems are collaborative filtering (also referred to as collective intelligence) and content-based filtering. Systems that use collaborative filtering recommend music based on a community of users, their

preferences/tastes, and their browsing behavior. A popular example of this is Last.fm. The most common issue with systems using strictly collaborative filtering is the Cold Start. This problem arises when music is either brand new or has not been reviewed or rated by users in a system.

Systems that use content-based filtering recommend music based on extracted content-related information (i.e., acoustic features) from the music pieces. The most popular example of this is Pandora Internet Radio. The main challenge with systems using primarily content-based filtering is processing time. Extracting content-related information is a time consuming process, either done by manual annotations or automatic feature extractions. Another limitation is that semantic meaning of each item analyzed is not always taken into account when producing recommendations. For example, a user may find a particular piece of music to be relaxing, which would not be directly considered in the extracted features.

In this thesis, a hybrid approach to music recommendation is developed by utilizing the strengths of both collaborative filtering and content-based filtering. Taking advantage of both techniques will effectively tackle the Cold Start problem prevalent in systems using collaborative filtering, and, in addition, it will address the lack of consideration for semantic meaning that is in systems using content-based filtering. Furthermore, this approach provides more variation in recommendations and thus more opportunity for discovery by the user.

Chapter 2

BACKGROUND AND LITERATURE REVIEW

2.1 Recommendation Systems

As the Internet is in widespread use today, the vast majority of computer, tablet, and smartphone users have encountered one or more recommendation systems. For example, imagine visiting a favorite online store to browse for a particular item of interest. After finding it and clicking the direct link, there may be a section on the page titled, "Customers who bought this item also bought." These items are listed as potentially interesting items based on the product in review. For registered users, a personalized list of recommendations will be automatically displayed upon logging into the website. The software used to provide recommendations is a recommendation system.

Personalized recommendations require a system to obtain some knowledge about each user. In other words, a recommendation system must develop and maintain a user profile that contains each user's preferences. These user preferences can be acquired explicitly by asking the user to rate a particular item or implicitly by monitoring user behavior [Jannach10].

2.1.1 Collaborative Filtering

What type of information should be used to determine a list of personalized recommendations? The most popular approach is to utilize the parameters related to behavior, opinions, and tastes collected from a large community of users. This approach is classified as collaborative filtering. The concept is that users who previously shared the same interests will be likely to be interested in similar items in the future.

Systems that take advantage of implicit user collaboration are common today. This approach does not require any actual knowledge specifically about the items being recommended [Jannach10]. An obvious strength to this approach is that complex data does not need to be included nor maintained in the system. Consequently, there will not be a large amount of overhead involved. However, not taking into account data specifically about the content may render the system less accurate in its recommendations. Additionally, it will be difficult to recommend items new to the system or items that have yet to be discovered. Another disadvantage is the potential for spam (advertisements, etc.) in user accounts that are automatically run by electronic messaging systems [Levy09].

A number of methods have been used to successfully recommend items based on collaborative filtering [Ekstrand10, Hameed12, Sachan13, Schafer07, Su09]. For example, in the paper, "A Survey of Collaborative Filtering Techniques," the authors summarized different types of collaborative filtering approaches [Su09]. Generally, the collaborative filtering recommendation systems can be divided into three groups:

memory-based systems, model-based systems, and hybrid systems (combining models with a database of user profiles). The authors also discussed the challenges of such approaches, including data sparsity, scalability, synonymy (many of the same item with different names), and shilling attacks (individuals providing positive ratings to their own product and negative ratings to items of competitors). Additionally, various distance measures were discussed for identifying similarity between items in collaborative filtering, including Pearson correlation, vector cosine distance, Euclidean distance, and others.

In the paper, "Exploring Social Annotations for the Semantic Web," the authors explored a collaborative approach that focuses on social annotations made manually by web users without a predefined formal ontology. Although these annotations were informal and vague, they were easily accessible. A sample of Del.icio.us data was collected by crawling its website. The dataset consisted of 2,879,614 tags from 10,109 users. The initial evaluation showed that this method can effectively discover semantically related web bookmarks [Wu06].

In the paper, "Usage Patterns of Collaborative Tagging Systems," the authors analyzed the structure of collaborative tagging systems and their dynamic aspects. It was observed that tags varied in frequency of use and in what they described. A significant amount of tagging was done for personal use; however, this information still could benefit other users [Golder06]. The evaluation of recommendation systems using collaborative filtering is also discussed in various publications [Breese98, Cacheda11, Herlocker04, Huang06, Lee12]. In [Breese98], the authors compared six collaborative filtering

algorithms using two evaluation metrics: average accuracy of predicted ranking for individual items and the utility of a ranked list of suggested items. In [Cacheda11], the authors proposed evaluation metrics such as coverage (the percentage of items the system is able to recommend), prediction accuracy in terms of mean absolute error and root mean square error, and classification and rank accuracy (precision, recall, ROC curves, and half-life utility). In [Lee12], the authors tested the prediction accuracy (mean absolute error and root mean square error) of 15 algorithms using the Netflix dataset in different experimental contexts to study the performance in relation to data size, density, user/item count, and dependency.

2.1.2 Content-based Filtering

There are several different reasons to make use of a recommendation system. For one, it can encourage users to take an action like purchasing a particular product or previewing a particular television show. Secondly, it can assist users in discovering new items they may not have otherwise found. Another reason is to solve the problem of 'information overload.' In other words, the system may filter through a large dataset in order to select the most interesting items for a user. This deals with technology commonly referred to as information retrieval and information filtering. However, the main focus is to distinguish between relevant and irrelevant data to the user. Taking advantage of information obtained from the contents of items in order to rank them is commonly referred to as content-based recommendation [Jannach10].

Essentially, the content-based approach utilizes the descriptions of items (gathered manually or extracted automatically) and a user profile, consisting of a level of importance with respect to these attributes. A user profile can simply be a query song, or it can be the descriptive preferences of a user. The content-based approach attempts to recommend items similar to those the respective user has previously found enjoyable. The basic process is matching attributes/preferences in a user profile with attributes described for a content item. The goal is to recommend new interesting items to that user [Lops11]. For example, characteristics of a piece of music may include the genre, the tempo, the key, the style, the timbre, the instrumentation, etc. Relevant information about a query track can be gathered manually or automatically by analyzing the acoustic features of the track.

In comparison with pure collaborative filtering, content-based recommendation has several advantages. It does not require large user groups in order to function with reliable accuracy. Also, there is no need to wait until an item has gained traffic from users. New items can be recommended as soon as item attributes are accessible. One disadvantage is the increased overhead due to the preprocessing required. In addition, performance based on automatic extraction of features may be less accurate due to the lack of semantic meaning [Jannach10].

In the paper, "Recommendation as Classification: Using Social and Content-based Information in Recommendation," the authors used IMDb (the Internet Movie Database) to gather features to describe the content of a movie. They conducted an experiment using 45,000 movie ratings collected from a community of over 250 users. The result

indicates that combining the content-related features outperforms the pure collaborative-filtering approach [Basu98]. In the paper, "Content-based Book Recommending Using Learning for Text Categorization," the authors propose a book recommendation system that uses the Bayesian learning algorithm to categorize the information obtained from the Web about a book based on user feedback [Mooney00]. In the paper, "Content-based Recommendation Systems," the authors indicated that the performance of a content-based filtering system depends on the richness of the information to separate preferable items from unwanted items [Pazzani07]. A more detailed survey on content-based recommendation systems can be found at [Lops11].

2.2 Music Recommendation Systems

Due to the enormous amount of music available via the Internet, one challenge for music lovers is to be able to discover music they find interesting without having to attempt to sift through it all. Music recommendation systems are evolving to solve this problem.

2.2.1 Academic Research

In the area of academia, a number of systems based on collaborative filtering have been proposed for music recommendation. In the paper, "Web-collaborative Filtering: Recommending Music by Crawling the Web," a web crawler was used to collect semantically related entities from the Web for collaborative filtering. Specifically, the crawler uses heuristics to collect lists of musical artists that can be used to supplement or replace user ratings in a collaborative filtering system. The experiment was conducted

using a dataset of 5,095 downloads for the test set and 23,438 downloads for the training set to include 981 artists associated with these downloads. The results showed promising use of the web crawler for collaborative filtering music recommendation [Cohen00].

In the paper, "Automatic Generation of Social Tags for Music Recommendation," the authors proposed an auto-tagging algorithm using supervised machine learning to generate social tags automatically based on acoustic features extracted from MP3 files. Acoustic features were extracted, including 20 Mel-Frequency Cepstral Coefficients (MFCCs) and 85 spectrogram coefficients sampled by constant-Q transform. Experiments were conducted using the 60 most popular tags from the Last.fm crawl data using a dataset consisting of 89,924 songs from 1,277 artists [Eck08].

A number of methods based on content-based filtering have been employed to successfully recommend music. In the paper, "A Music Recommendation System Based on Music Data Grouping and User Interests," songs were represented using perceptual properties of music objects, including pitch, duration, and volume (loudness). User preferences were expressed as access histories recorded in a profile. Based on the access histories, users were divided into groups based on their interests in the feature space. Three recommendation methods were tested to discover similar users for music recommendations [Chen01].

In the paper, "A Music Search Engine Built upon Audio-based and Web-based Similarity Measures," the authors used a symmetrized Kullback-Leibler divergence calculated on the means and covariance matrices based on MFCCs for musical similarity. The dataset

consisted of 12,601 tracks. The goal was to create a search engine for large music collections that could be queried by natural language text input. Last.fm's track specific tag information was used as a ground truth [Knees07].

In the paper, "Evaluation of Distance Measures between Gaussian Mixture Models of MFCCs," the authors compared the measures, Kullback-Leibler, the earth mover's distance, and the normalized L2 distance, all based on the MFCCs. Distance measures were evaluated based on the MIREX 2004 genre classification contest, made up of 729 songs from 6 genres. The results showed that all three distance measures perform similarly [Jenson07].

In the paper, "Song-level Features and Support Vector Machines for Music Classification," the authors compared Kullback Leibler divergence and Mahalanobis distance based on MFCCs. The distance measures were tested with respect to artist identification and performed comparably on a dataset of 1,200 pop songs performed by 18 artists [Mandel05].

In the paper, "Lightweight Measures for Timbral Similarity of Musical Audio," the authors proposed lightweight similarity measures based on MFCCs that scale well for large collections. The distance measures compared were Kullback Leibler divergence and Mahalanobis distance. These measures were evaluated with genre classification between several datasets based on a 1-nearest neighbor classifier. The datasets included one from the ISMIR 2004 genre classification contest, an in-house set of 3600 tracks, and

a larger in-house set to emulate more of a medium-sized personal collection. The results confirmed that both distance measures perform similarly [Levy06].

2.2.2 Commercial Systems

Two of the most popular applications currently being used are Last.fm and Pandora. Last.fm is a music service that assists users in discovering new music they like based on the music they currently listen to. Last.fm was founded in the United Kingdom in 2002 and was acquired by CBS Interactive in 2007. On April 28, 2014, Last.fm terminated its streaming music service; however, its recommendation technology is still used to integrate with other streaming services. The Last.fm website contains a plethora of information about artists, albums, and music tracks. It also provides information about music similar to a selected artist and music tags created solely by registered users. Users have the ability to read about artists they are interested in, view/play their most popular tracks, browse their albums, and learn about similar artists. They can also learn about upcoming events, listen to streamed music, and watch videos.

What sets Last.fm apart from other music recommenders is the participation of their community. It truly is a collaborative-based approach. Users can stream music from a channel they create, comment on a track they like or dislike (like YouTube or Facebook), provide music tags to a song, express ‘love’ for a song, or ‘ban’ a song from being played via their account.

When a new user registers a free account with Last.fm, a music profile will be created. It can be filled with implicit data gathered from logs of the user's listening habits, which tracks he or she 'loves,' which tracks are 'banned,' etc. Additional data can be gathered explicitly such as when the system asks the user for favorite artists. Logs can be gathered from media player software on the computer, as well, from Last.fm's free software, Scrobbler, but it must be installed and running in the background while listening to music. This will enable Last.fm to keep track of a user's listening habits even when not listening to music directly on their website.

Figure 1 illustrates the Last.fm web-based application to stream music. At the top, the current station/channel is present as 'Poncho Sanchez Radio.' On the top right, a user can click the heart to express he or she loves a particular track. One can also click the universal 'no' symbol to never play the current track again. The artist, song title, and album are provided on the lower left. Album art and other artist photos automatically cycle as a slideshow background. Information about the artist current being played is displayed towards the bottom. Similar artists are also shown. The option to add music tags is presented on the bottom right. There are also links to purchase the song or the entire album via an online music store such as Amazon.com. The channel was created with the salsa/Latin jazz musician, Poncho Sanchez. Illustrated is a track being played by a similar artist, Ray Baretto. The most popular user-created music tags are shown: "salsa, latin, latin jazz, jazz, and boogaloo."

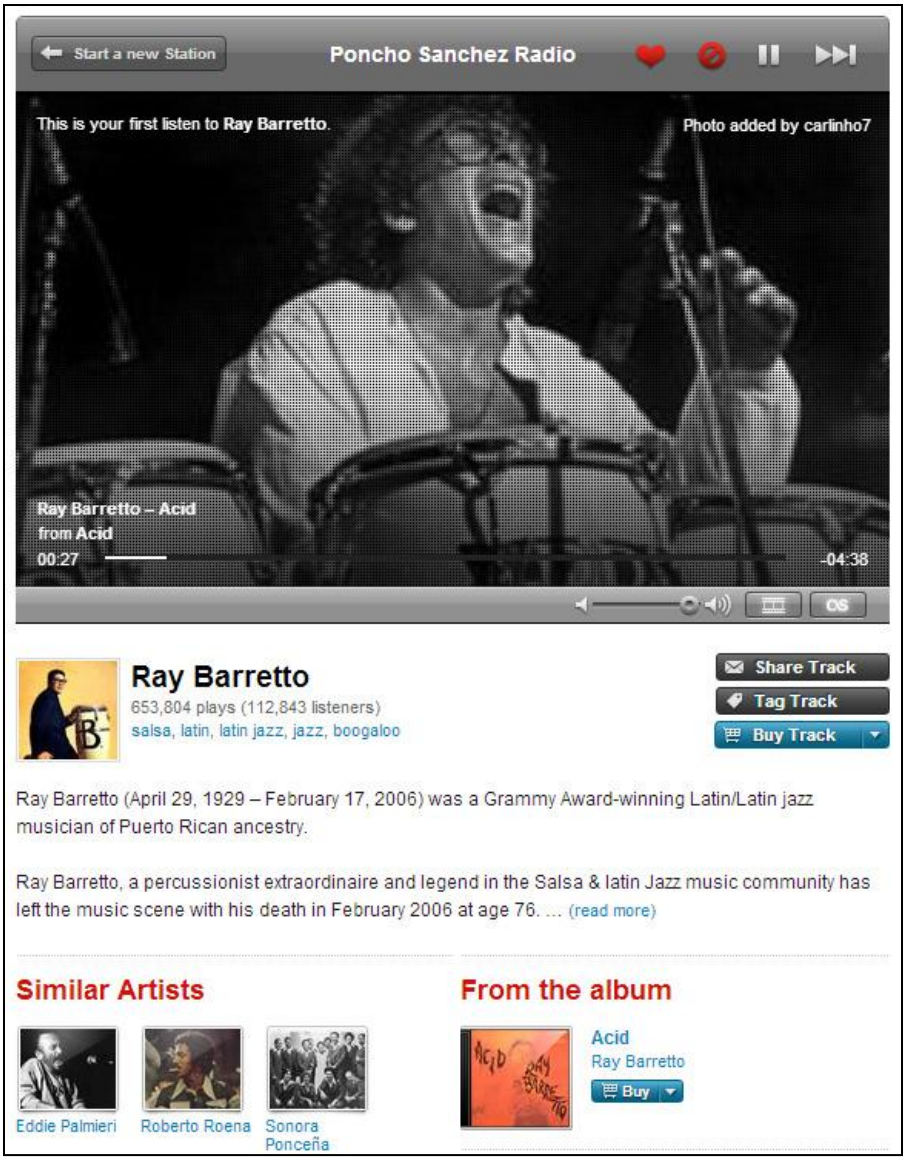


Figure 1: Last.fm Web-based Music Player

Figure 2 displays a user's favorite artists and also includes the number of times a song by each artist has been played. A new user will be explicitly prompted by Last.fm to provide favorite artists. After each favorite artist has been selected, a list of similar artists is shown to recommend more favorites to be chosen. When a user clicks the heart to

'love' a song currently being played, that artist will be automatically added to the favorite list [Last.fm14].

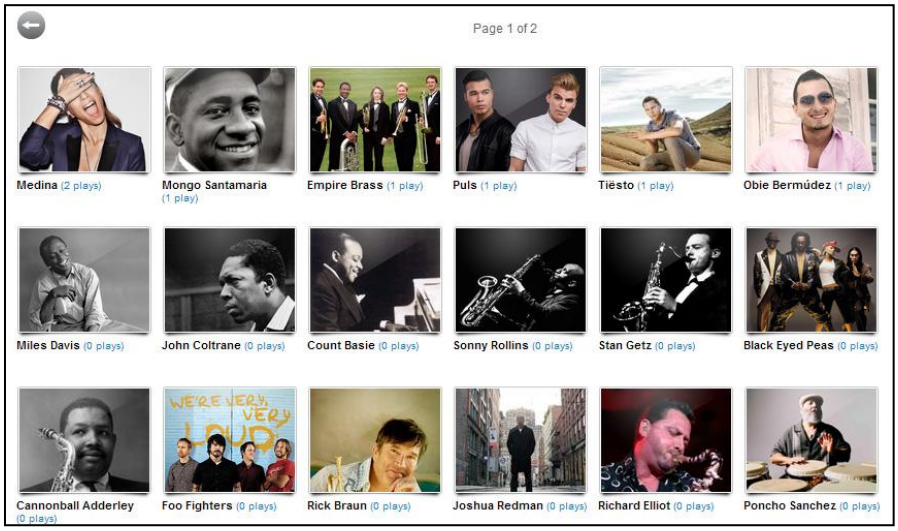


Figure 2: Last.fm User Taste

The extremely popular music recommendation system based on the Music Genome Project is known as Pandora. The Music Genome Project was started in 2000 with the goal of gathering detailed technical and qualitative features of music tracks in order to gain a broad and deep understanding of music. The project is comprised of over ten years of analysis by professional musicologists and encompasses everything from early music (including Renaissance and Baroque music) to the new music of today.

Through the Music Genome Project, each song/track is analyzed by a professional music analyst using up to 450 distinct musical attributes. In addition to capturing the musical identify of a piece of music, these features also characterize important qualities related to understanding the musical tastes of listeners.

Pandora utilizes a content-based filtering approach based on manually extracted features. For registered users, Pandora maintains user profiles based on user logs including when a user clicks the 'thumb up' button, 'thumb down' button, or skips a track. In order to create a music channel, a user must provide an artist or song title. Pandora customizes this channel's recommended music by the unique characteristics of the channel name (artist or song title) along with the user logs. Pandora does not currently make use of automated acoustic data extraction.

Pandora is available as an application on various devices. A web-based app can be run directly from a web browser, and a desktop app is available to paid users without having to open a browser. Figure 3 illustrates the Pandora One computer application.

Figure 3: Pandora One App for PC

At the top, the current station/channel is displayed as the artist, 'Deadmau5 Radio.' The song title, artist, album, and album art are shown next. On the bottom, a check box is visible on the thumb up button indicating that the thumb up was previously selected for this track. Directly to the left of the thumb up button is an up arrow icon. Upon clicking it, more options are made available including 'Why this Song,' 'Buy,' and 'About this station.' Clicking 'Why this Song,' on this sample shows that the song was recommended because it "features house roots, four-on-the-floor beats, electronica influences, beats made for dancing, and many other similarities identified in the Music Genome Project." Putting the mouse over the 'Buy' option provides hyperlinks to purchase this track via iTunes or Amazon. Clicking 'About this station,' displays detailed information about this channel in association with the current user (channel creation date, thumbed-up tracks, and thumbed down tracks) [Pandora14].

Chapter 3

METHODOLOGY

3.1 Hybrid Approach to Recommendation

There are several drawbacks to relying solely on collaborative filtering to recommend music. The biggest problem is the "Cold Start." Music tracks are only tagged as often as listeners are discovering or listening to them. In other words, there are little or no available 'tags' to describe new music or music that has not been discovered yet. Additionally, listeners are more willing to supply tags for songs they enjoy most than for songs they mildly enjoy or do not enjoy at all. Because of this, it is difficult for a system using collaborative filtering to provide accurate recommendations when there are not a sufficient amount of music tags available for a music track.

Content-based recommendation systems relying primarily on automatic extraction of acoustic features require longer processing time and a higher amount of resources. Systems using manual extraction of music features will encounter problems with scalability. As more and more music becomes produced and becomes widely available (in stores and online), more resources are required to analyze the new music [Levy09].

A hybrid approach is proposed that will utilize the benefits of user-supplied music tags (collaborative filtering) and automatic extraction of acoustic features (content-based

filtering). This system will improve on the weaknesses of systems primarily using one or the other.

3.1.1 User-supplied Music Tags

Collaborative tags (i.e., social tags) are brief descriptions provided by a community of Internet users. These tags assist in navigation through large collections of media [Golder06, Wu06]. Although users can enter any text as a tag, it is often beneficial to select tags already created by users in order to generate a usable navigation system. These tags can then be utilized by tag-based search interfaces to display the most popular tags for the page or item to be described [Levy09]. The system developed in this thesis makes use of music tags from Last.fm. Tags were retrieved from Last.fm's system in order to recommend music based on those tags.

3.1.2 Acoustic Features from Audio Recording

Mel-Frequency Cepstral Coefficients (MFCCs) are the common features extracted from an audio recording used extensively in speech recognition and music analysis [Chuan13, Levy09]. In the system developed, we extracted MFCC data using Sonic Annotator with the Queen Mary Vamp plug-ins. Sonic Annotator [Sonic Annotator14] is an open source command line program for batch extraction of audio features from multiple audio files. In order to visualize how this works manually, we utilized Sonic Visualiser [Sonic Visualiser14], an open source application for viewing and analyzing the contents of music audio files. This software is available for Linux, OS/X, and Windows and was

developed at the Centre for Digital Music in Queen Mary, University of London. Let us go into further detail on how Sonic Visualiser can extract the acoustic features from an audio recording.

After the software is successfully installed, we run Sonic Visualiser and open a music file. High quality .mp3 format will work, as well. Next, using Queen Mary's plug-ins, we extracted MFCC's from the audio track as illustrated in Figure 4. We used the default settings as shown in Figure 5.

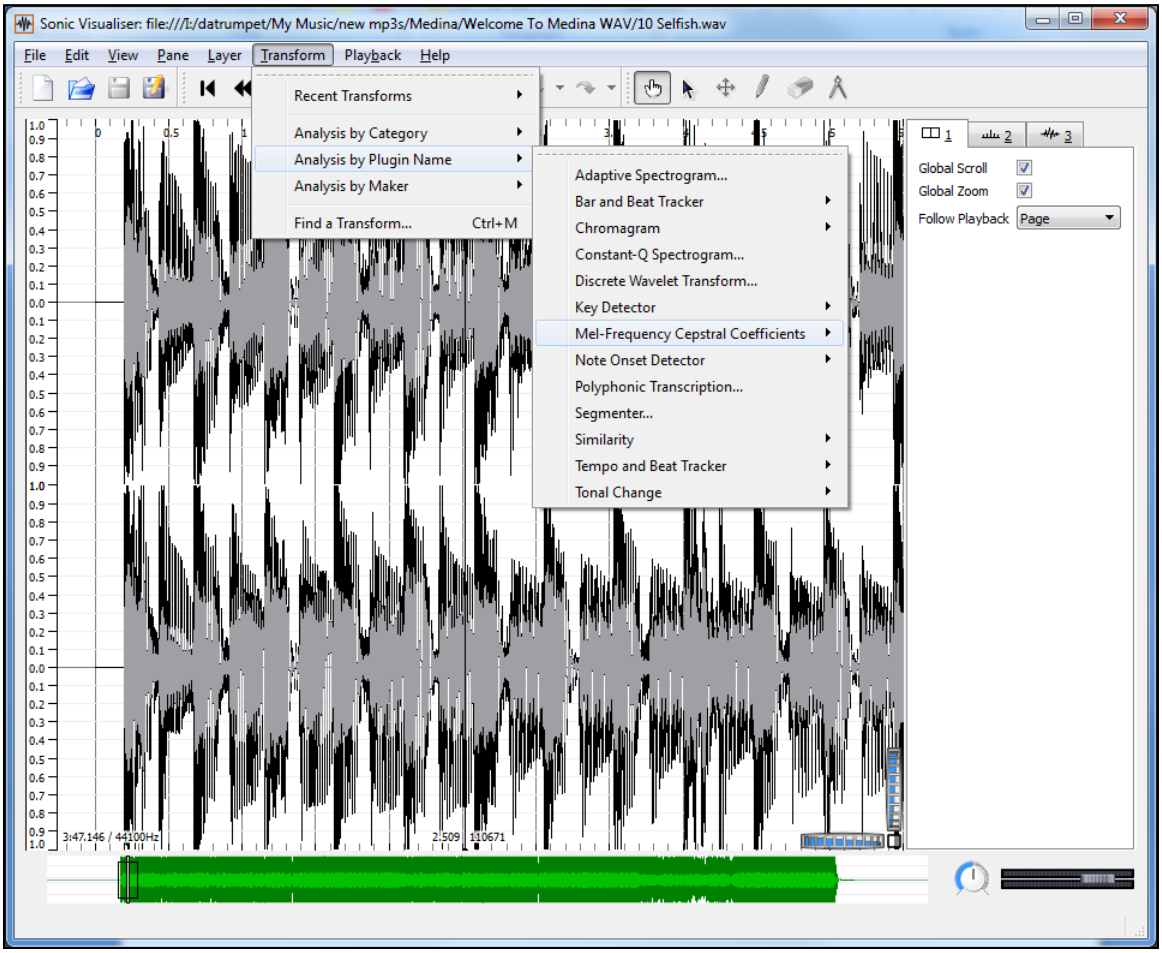


Figure 4: Calculating MFCC Using Sonic Visualiser

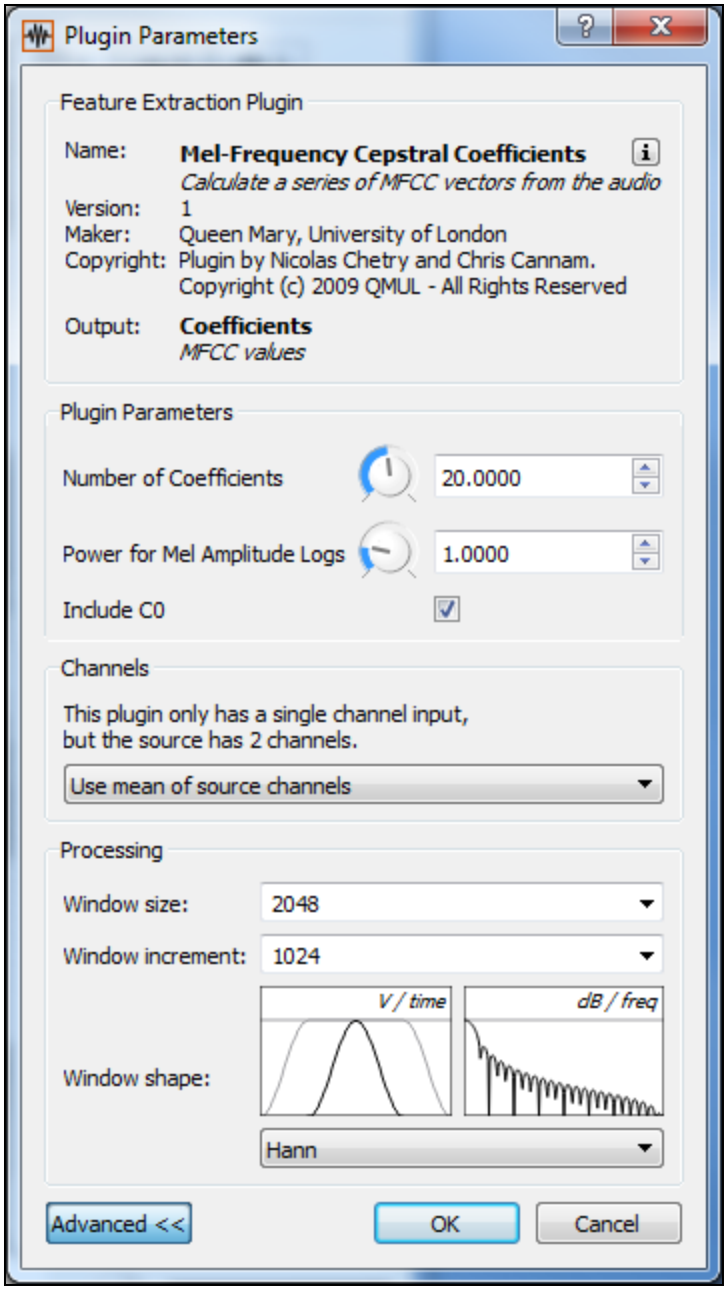
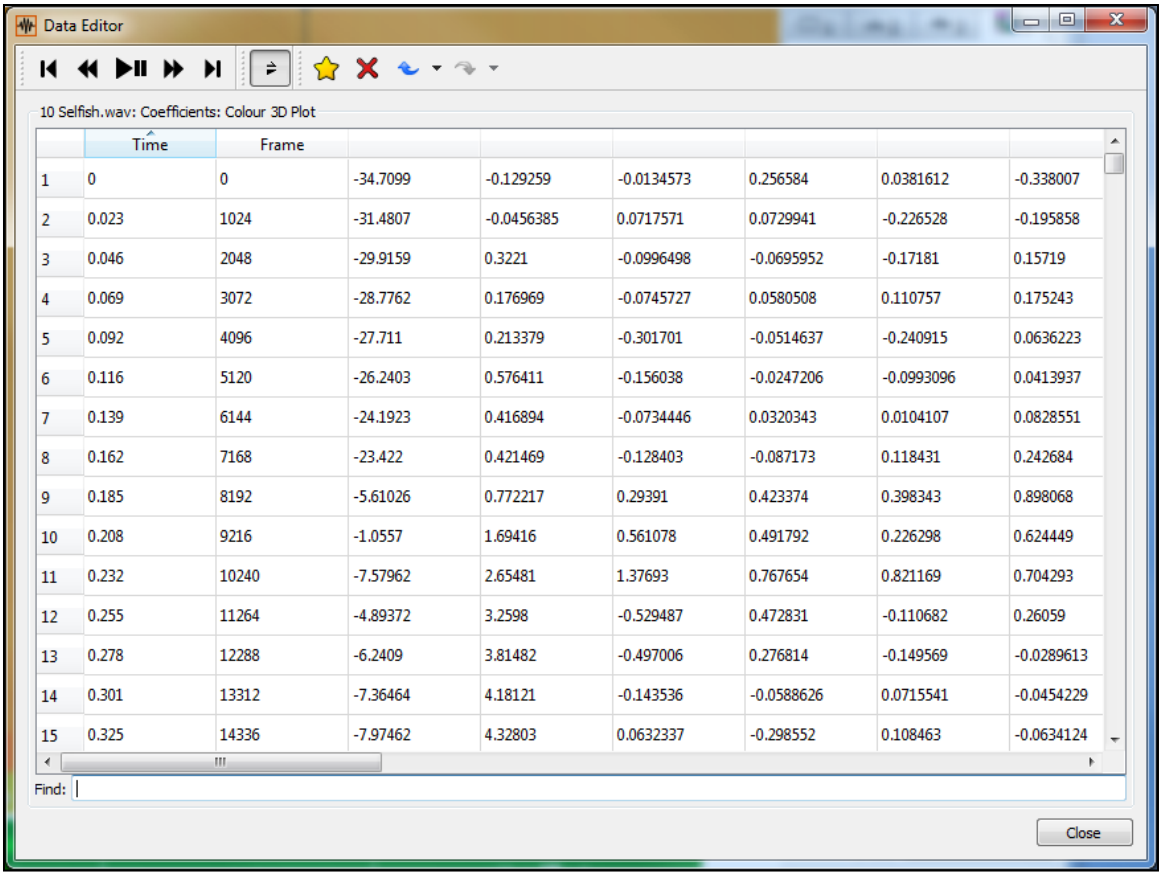


Figure 5: Sonic Visualiser MFCC Default Settings

Next, we selected Layer and Edit Layer Data in order to show the data as in Figure 6. We then extracted the mean of MFCC data into a CSV file. Each row has twenty values, each of which represents an MFCC coefficient [Sonic Visualiser14].



	Time	Frame						
1	0	0	-34.7099	-0.129259	-0.0134573	0.256584	0.0381612	-0.338007
2	0.023	1024	-31.4807	-0.0456385	0.0717571	0.0729941	-0.226528	-0.195858
3	0.046	2048	-29.9159	0.3221	-0.0996498	-0.0695952	-0.17181	0.15719
4	0.069	3072	-28.7762	0.176969	-0.0745727	0.0580508	0.110757	0.175243
5	0.092	4096	-27.711	0.213379	-0.301701	-0.0514637	-0.240915	0.0636223
6	0.116	5120	-26.2403	0.576411	-0.156038	-0.0247206	-0.0993096	0.0413937
7	0.139	6144	-24.1923	0.416894	-0.0734446	0.0320343	0.0104107	0.0828551
8	0.162	7168	-23.422	0.421469	-0.128403	-0.087173	0.118431	0.242684
9	0.185	8192	-5.61026	0.772217	0.29391	0.423374	0.398343	0.898068
10	0.208	9216	-1.0557	1.69416	0.561078	0.491792	0.226298	0.624449
11	0.232	10240	-7.57962	2.65481	1.37693	0.767654	0.821169	0.704293
12	0.255	11264	-4.89372	3.2598	-0.529487	0.472831	-0.110682	0.26059
13	0.278	12288	-6.2409	3.81482	-0.497006	0.276814	-0.149569	-0.0289613
14	0.301	13312	-7.36464	4.18121	-0.143536	-0.0588626	0.0715541	-0.0454229
15	0.325	14336	-7.97462	4.32803	0.0632337	-0.298552	0.108463	-0.0634124

Figure 6: Sonic Visualiser MFCC Data

3.2 Similarity Metrics

In order to present appropriate music recommendations based on a music track chosen by the user, a recommendation system must compare the similarity between the input track and a large dataset of tracks.

3.2.1 Similarity Metric for Music Tags

Let us discuss utilizing a collaborative-based approach. As discussed earlier, Last.fm hosts a large community of users. This service boasts an open source API to allow

developers to access their database. The method, *track.getTopTags*, provides the most prevalent user-supplied 'tags' for a music track and also the number of users who provided that tag. The track name and artist name are required input parameters, and the output will include top tags ordered by the tag count descending [Last.fm14]. Last.fm pre-normalized the tag counts based on the top tag for each track having a value of 100.

In this thesis, calculating collaborative-based similarity between two music tracks was accomplished by retrieving the top ten tags from Last.fm for each music track and using a weighted Euclidian distance in order to measure the distance between the songs with respect to the collaborative music tags. Euclidian distance was chosen because it is a relatively simple approach. Based on preliminary results from this thesis, it was determined that this approach performs very well. A weight was employed to account for distance of tags with varying tag counts. In other words, a tag submitted by many users has a greater impact on the distance than a tag submitted by very few users.

The weighted Euclidian distance $d_{x,y}(\boldsymbol{p}, \boldsymbol{q})$ is the distance between two music tracks x and y represented as tag vectors $(\boldsymbol{p}, \boldsymbol{q})$ given by the Pythagorean formula:

$$d_{x,y}(\boldsymbol{p}, \boldsymbol{q}) = \sqrt{w_1(p_1 - q_1)^2 + w_2(p_2 - q_2)^2 + \cdots + w_n(p_n - q_n)^2)}, \quad (1)$$

where w_i is the weight for tag i calculated in equation (2):

$$w_i = p_i + q_i. \quad (2)$$

The normalized tag count p_i is the tag count of tag i for the query track x as shown in equation (3):

$$p_i = \frac{r_i}{\sum_{i=1}^{n} r_i}, \quad (3)$$

where r_i is the original tag count of tag i obtained from Last.fm. Likewise, the normalized tag count q_i is the tag count of tag i for the compared track y. $\boldsymbol{p}$ and $\boldsymbol{q}$ are the normalized tag count vectors associated with two tracks, the query track and compared track, respectively. Each track is represented by their combined top n music tags from Last.fm in this study. Each value is weighted by multiplying by number of supplied tags for the most popular tag of each set of songs. The set with the lowest distance value includes the song from the dataset that is the most similar to the query song [Deza09].

For a specific example of the similarity metric based on Last.fm top tags, let us use the query track, *Blue Train* by John Coltrane (id 73514), and the compared track, *Ornithology* by Charlie Parker (id 74709).

Table 1 shows the top ten tags extracted from Last.fm for (a) *Blue Train* and (b) *Ornithology* with shared tags highlighted in light blue. The normalized tag count p_i and q_i can be calculated using equation (3). For example, the normalized tag count for the tag "jazz" for *Blue Train* is approximately 0.510, calculated by its original the tag count, 100, divided by the sum of tag counts, 196. The sum of normalized tag counts for each track will equal 1.

Once the normalized tag counts are calculated for the two tracks, the weight for an individual tag can be calculated using the normalized tag counts as in equation (2). For example, the weight for the tag "jazz" is the sum of the normalized tag counts for *Blue Train* (≈ 0.510) and *Ornithology* (0.450): approximately 0.960.

Query Track		
Blue Train by John Coltrane		
Tag Name	**Tag Count** (r_i)	**Normalized Tag Count** (p_i)
Jazz	100	0.5102041
saxophone	29	0.1479592
john coltrane	20	0.1020408
Bebop	18	0.0918367
Hard Bop	12	0.0612245
instrumental	5	0.0255102
free jazz	4	0.0204082
Blues	3	0.0153061
Coltrane	3	0.0153061
classic jazz	2	0.0102041
Tag Count Sum	**196**	

(a)

Compared Track		
Ornithology by Charlie Parker		
Tag Name	**Tag Count** (r_i)	**Normalized Tag Count** (q_i)
jazz	100	0.4504505
bebop	49	0.2207207
saxophone	27	0.1216216
charlie parker	16	0.0720721
instrumental	9	0.0405405
bop	5	0.0225225
jazzysmalejazz	4	0.0180180
Good Stuff	4	0.0180180
piebald composers	4	0.0180180
milk is getting warm	4	0.0180180
Tag Count Sum	**222**	

(b)

Table 1: Tags for Query Track (a) *Blue Train* and Compared Track (b) *Ornithology*

Table 2 lists the normalized tag counts for the two tracks and the calculated weights for each unique tag with the shared tags highlighted in blue.

Unique Tag	Normalized Tag Count for *Blue Train* (p_i)	Normalized Tag Count for *Ornithology* (q_i)	Tag Weight (w_i)
Jazz	0.510204082	0.450450450	0.960654532
Bebop	0.091836735	0.220720721	0.312557455
saxophone	0.147959184	0.121621622	0.269580805
john coltrane	0.102040816	0	0.102040816
charlie parker	0.072072072	0	0.072072072
instrumental	0.025510204	0.040540541	0.066050745
Hard Bop	0.061224490	0	0.061224490
Bop	0.022522523	0	0.022522523
free jazz	0.020408163	0	0.020408163
jazzysmalejazz	0.018018018	0	0.018018018
Good Stuff	0.018018018	0	0.018018018
piebald composers	0.018018018	0	0.018018018
milk is getting warm	0.018018018	0	0.018018018
Blues	0.015306122	0	0.015306122
Coltrane	0.015306122	0	0.015306122
classic jazz	0.010204082	0	0.010204082

Table 2: Normalized Tag Count and Weight for *Blue Train* and *Ornithology*

Next, we calculate the weighted Euclidean distance. Based on the normalized tag counts and tag weights in Figure 8, the weighted Euclidean distance can be calculated using equation (1) as follows:

$$d_{x,y}(\boldsymbol{p},\boldsymbol{q})$$

$$= \sqrt{0.961(0.510-0.450)^2 + 0.313(0.092-0.221)^2 + \cdots + 0.010(0.010-0)^2}$$

$$\cong 0.103.$$

For another example of the similarity metric based on Last.fm top tags, let us use the same query track, *Blue Train* by John Coltrane (id 73514), and another compared track, *Freak Out* by 311 (id 63869).

Query Track		
Blue Train by John Coltrane		
Tag Name	**Tag Count (r_i)**	**Normalized Tag Count (p_i)**
Jazz	100	0.510204082
saxophone	29	0.147959184
john coltrane	20	0.102040816
Bebop	18	0.091836735
Hard Bop	12	0.061224490
instrumental	5	0.025510204
free jazz	4	0.020408163
Blues	3	0.015306122
Coltrane	3	0.015306122
classic jazz	2	0.010204082
Tag Count Sum	**196**	

(a)

Compared Track		
Freak Out by 311		
Tag Name	**Tag Count (r_i)**	**Normalized Tag Count (q_i)**
alternative rock	100	0.304878049
rock	66	0.201219512
ska	33	0.100609756
reggae	33	0.100609756
running songs	16	0.048780488
Genre-meld	16	0.048780488
non-select	16	0.048780488
Omaha	16	0.048780488
311	16	0.048780488
white boy rap	16	0.048780488
Tag Count Sum	**328**	

(b)

Table 3: Tags for Query Track (a) *Blue Train* and Compared Track (b) *Freak Out*

Table 3 shows the top ten tags extracted from Last.fm for (a) *Blue Train* and (b) *Freak Out*. One can observe that there are no shared tags between these two songs. The normalized tag count is calculated using equation (3).

After calculation of normalized tag counts, the weight for each tag is calculated using the normalized tag counts per equation (2). Table 4 lists the normalized tag counts for the two tracks and the calculated weights for each unique tag.

Unique Tag	Normalized Tag Count for *Blue Train* (p_i)	Normalized Tag Count for *Ornithology* (q_i)	Tag Weight (w_i)
jazz	0.510204082		0.510204082
alternative rock		0.304878049	0.304878049
rock		0.201219512	0.201219512
saxophone	0.147959184		0.147959184
john coltrane	0.102040816		0.102040816
Ska		0.100609756	0.100609756
reggae		0.100609756	0.100609756
bebop	0.091836735		0.091836735
Hard Bop	0.061224490		0.061224490
running songs		0.048780488	0.048780488
Genre-meld		0.048780488	0.048780488
non-select		0.048780488	0.048780488
Omaha		0.048780488	0.048780488
311		0.048780488	0.048780488
white boy rap		0.048780488	0.048780488
instrumental	0.025510204		0.025510204
free jazz	0.020408163		0.020408163
blues	0.015306122		0.015306122
Coltrane	0.015306122		0.015306122
classic jazz	0.010204082		0.010204082

Table 4: Normalized Tag Count and Weight for *Blue Train* and *Freak Out*

Next, we calculate the weighted Euclidean distance. Based on normalized tag counts and tag weights in Table 4, the weighted Euclidean distance is calculated using equation (1):

$$d_{x,y}(\boldsymbol{p},\boldsymbol{q}) = \sqrt{0.51(0.51-0)^2 + 0.305(0-0.305)^2 + \cdots + 0.01(0.01-0)^2)}$$

$$\cong 0.421.$$

Given that the weighted Euclidean distance between *Blue Train* and *Ornithology* is about 0.103 and the weighted Euclidean distance between *Blue Train* and *Freak Out* is about

0.421, it can be observed that *Ornithology is* considered to be more similar to *Blue Train* than *Freak Out* is because of the lower distance value based on collaborative filtering.

3.2.2 Similarity Metric for Audio Signal

In this thesis, calculating acoustic similarity between two music tracks was accomplished by extracting mean and variance values from the Mel-Frequency Cepstral Coefficients (MFCC) of each audio recording and using Mahalanobis distance in order to measure the distance between the songs with respect to acoustic features. This method was chosen because it was the most straightforward approach that also performs comparably well to other measures according to the academic research previously discussed [Levy06, Mandel05].

The Mahalanobis distance $D_{x,y}$ is the distance between two music tracks x and y represented by their MFCC-based vectors ($\boldsymbol{u}$, $\boldsymbol{v}$) respectively:

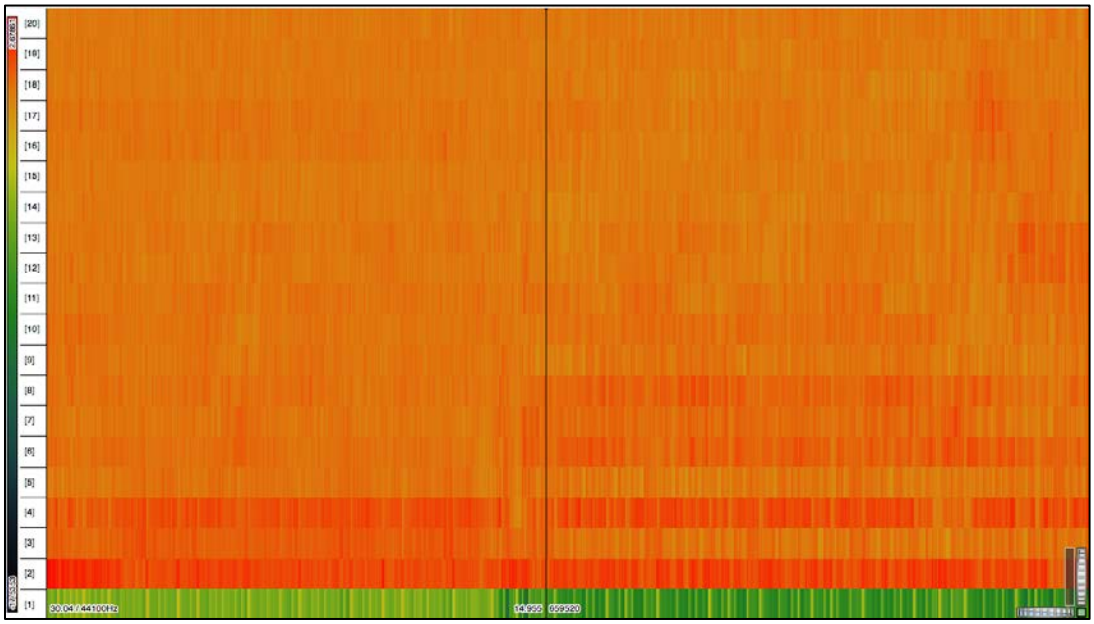
$$D_{x,y}(\boldsymbol{u},\boldsymbol{v}) = (\boldsymbol{u}-\boldsymbol{v})^T \sum^{-1} (\boldsymbol{u}-\boldsymbol{v}), \quad (4)$$

where Σ is the covariance matrix of acoustic features across the entire dataset of music tracks, which is approximated as a diagonal matrix of the individual feature's variances. $\boldsymbol{u}$ and $\boldsymbol{v}$ are vectors containing MFCC average and variance values associated with two tracks, the query track x and compared track y, respectively [Mandel05].

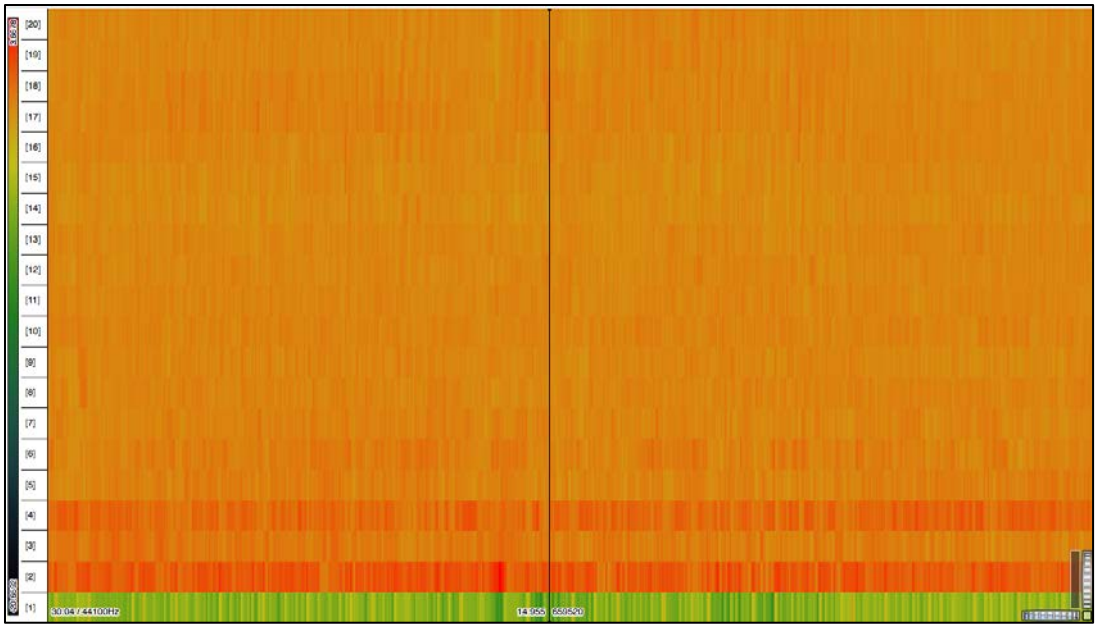
In this thesis, the first 20 MFCCs were extracted from the audio recording using Sonic Annotator. Each audio recording was processed frame-by-frame with a frame size of 2048 samples (46.4 ms if the recording is sampled at 44.1 kHz) with a hop size of 1024 samples. Therefore, a three minute audio recording results in more than 7,750 frames, and each frame is represented using 20 MFCCs.

The MFCCs generated for each frame were then summarized together by calculating the average and variance of the first 20 MFCCs across all frames. As a result, the acoustic characteristics of each audio recording are represented by 40 numerical values: 20 averages and 20 variances of the first 20 MFCCs.

For an example of similarity based on MFCC data, let us use the query track, *Beautiful Disaster* by 311 (id 69134), and the compared track, *Freak Out* by 311 (id 63869).



(a) *Beautiful Disaster*



(b) *Freak Out*

Figure 7: Visualization of the First 20 MFCCs
for (a) *Beautiful Disaster* and (b) *Freak out*

In Figure 7, the horizontal axis represents time, while the vertical axis indicates the 20 MFCCs with colors representing the value for a particular MFCC.

Table 5 is the frame-by-frame 20 MFCCs for (a) *Beautiful Disaster* and (b) *Freak Out*, including average and variance values extracted automatically using Sonic Annotator. Let us organize each set as a vector of 40 values. The vector $\boldsymbol{u}$ in equation (4) represents the 20 average values and 20 variance values for *Beautiful* Disaster, while $\boldsymbol{v}$ represents the 20 average values and 20 variance values for *Freak Out*.

Query Track *x*		
Beautiful Disaster by 311		
MFCCs	**Average**	**Variance**
1	-5.3793800	2.2327500
2	1.6020600	0.3104200
3	0.2570150	0.5509020
4	0.7835540	0.3876730
5	0.0926931	0.1414710
6	0.3427570	0.1217340
7	0.1379940	0.0936205
8	0.2328870	0.0817499
9	0.0910813	0.0637536
10	0.1667890	0.0487892
11	0.1155130	0.0532833
12	0.0402561	0.0515344
13	0.1021760	0.0509366
14	-0.0190406	0.0467154
15	0.0112838	0.0384650
16	0.0244702	0.0420153
17	0.1065370	0.0438976
18	0.0024070	0.0425422
19	0.0034893	0.0444777
20	0.0128852	0.0463218

(a)

Compared Track *y*		
Freak Out by 311		
MFCCs	**Average**	**Variance**
1	-4.0285000	1.4615900
2	1.3949400	0.2908530
3	0.1768860	0.5063070
4	0.8119710	0.3577290
5	0.0907603	0.1624580
6	0.2416660	0.0918268
7	0.0486814	0.0602763
8	0.0890771	0.0491793
9	0.0122621	0.0558450
10	0.1347060	0.0414332
11	0.0604939	0.0412303
12	-0.0055282	0.0392996
13	0.0395426	0.0401052
14	-0.0897435	0.0402974
15	-0.0408359	0.0359550
16	0.0385006	0.0391745
17	0.0788703	0.0446318
18	0.0399256	0.0413123
19	0.0013677	0.0405976
20	0.0219407	0.0405980

(b)

Table 5: Average and Variances of the First 20 Mel-Frequency Cepstral Coefficients for (a) *Beautiful Disaster* and (b) *Freak Out*

The term $(\boldsymbol{u} - \boldsymbol{v})$, in Table 6, is the difference (a) between the corresponding values in the *Beautiful Disaster* vector and the *Freak Out* vector. The covariance matrix Σ is a 40 x 40 matrix calculated previously based on the entire dataset. The second and third terms of the Mahalanobis distance, $\sum^{-1}(\boldsymbol{u} - \boldsymbol{v})$, are the product of (b) the inverse covariance matrix and (a) the difference matrix.

Difference Matrix $(u - v)$	
1	-1.3508800
2	0.2071200
…	…
39	0.0038801
40	0.0057238

(a)

	Inverse Covariance Matrix $\sum^{-1}$				
	1	**2**	**…**	**39**	**40**
1	0.513781932	0.432328116	…	1.368893943	-1.077001276
2	0.432328116	2.874080794	…	-6.381114659	-3.589329695
…	…	…	…	…	…
39	1.368893943	-6.381114659	…	9057.864846	-4671.120479
40	-1.077001276	-3.589329695	…	-4671.120479	7997.792215

(b)

Table 6: (a) Difference Matrix and (b) Inverse Covariance Matrix

Difference Matrix Transposed $(u - v)^T$				
1	**2**	**…**	**39**	**40**
-1.3508800	0.2071200	…	0.0038801	0.0057238

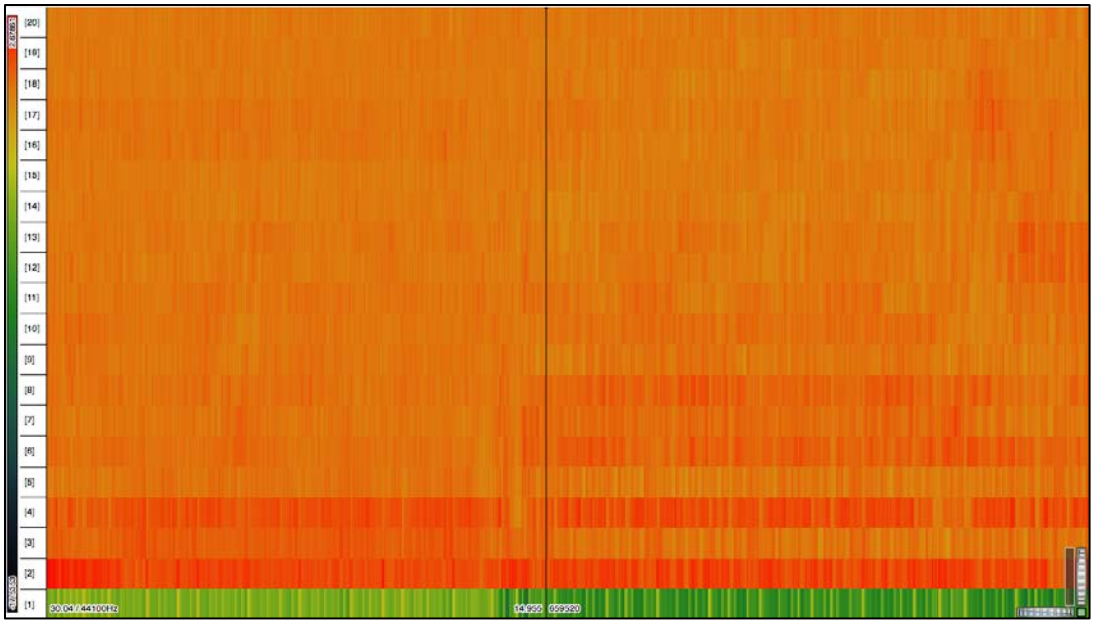
(a)

Product Matrix $\sum^{-1}$ $(u - v)$	
1	-0.618028378
2	0.672041948
…	…
3	-8.080172915
4	17.79374393

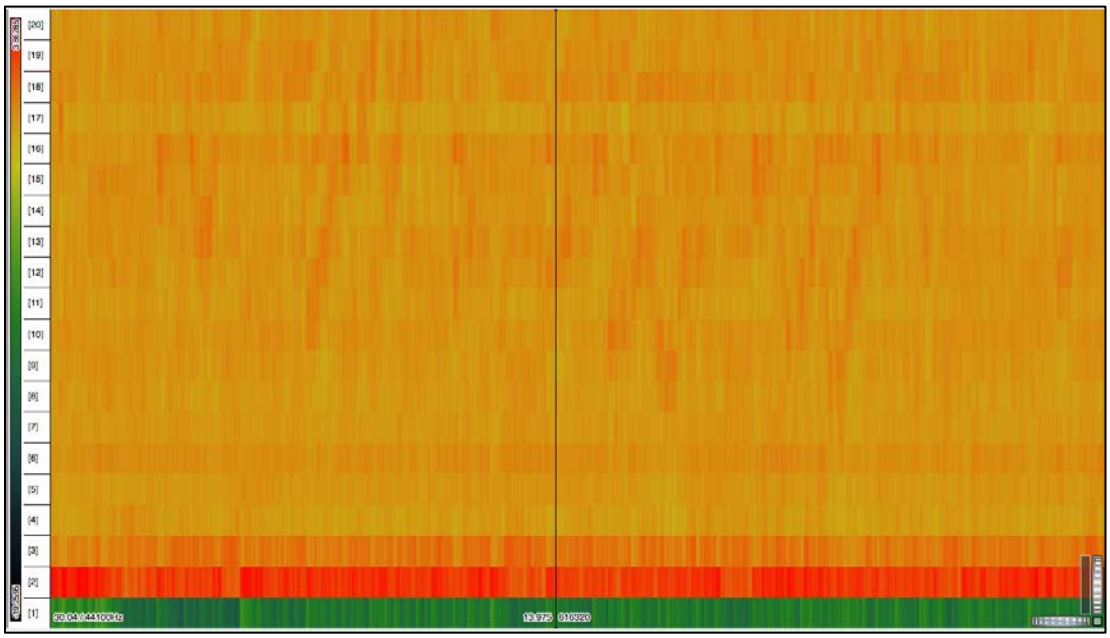
(b)

Table 7: (a) Difference Matrix Transposed (b) Product Matrix

Next, we calculate the product of $(u - v)$ transposed (a) and the product matrix (b) as shown in Table 7. The result, the Mahalanobis distance, is approximately 11.518.



(a) *Beautiful Disaster*



(b) *Ornithology*

Figure 8: Visualization of the First 20 MFCCs
for (a) *Beautiful Disaster* and (b) *Ornithology*

For another example of MFCC-based similarity, we use the query track, *Beautiful Disaster* by 311 (id 69134), and the compared track, *Ornithology* by Charlie Parker (id 74709). Figure 8 shows a visual of the first 20 MFCCs of the tracks respectively.

Query Track ***u***		
Beautiful Disaster by 311		
MFCCs	**Average**	**Variance**
1	-5.3793800	2.2327500
2	1.6020600	0.3104200
3	0.2570150	0.5509020
4	0.7835540	0.3876730
5	0.0926931	0.1414710
6	0.3427570	0.1217340
7	0.1379940	0.0936205
8	0.2328870	0.0817499
9	0.0910813	0.0637536
10	0.1667890	0.0487892
11	0.1155130	0.0532833
12	0.0402561	0.0515344
13	0.1021760	0.0509366
14	-0.0190406	0.0467154
15	0.0112838	0.0384650
16	0.0244702	0.0420153
17	0.1065370	0.0438976
18	0.0024070	0.0425422
19	0.0034893	0.0444777
20	0.0128852	0.0463218

(a)

Compared Track ***v***		
Ornithology by Charlie Parker		
MFCCs	**Average**	**Variance**
1	-8.2900900	4.9917900
2	2.6329000	0.4329740
3	0.5234710	0.2693300
4	-0.2288010	0.0728657
5	-0.1355770	0.0516473
6	0.1354510	0.0616032
7	-0.2065490	0.0594943
8	-0.1241680	0.0552533
9	-0.1276950	0.0801893
10	0.0193225	0.0906240
11	-0.1866130	0.1265750
12	-0.0214313	0.1345910
13	0.0577351	0.1505030
14	-0.0316441	0.1426060
15	0.0620594	0.1334630
16	0.0739553	0.1429660
17	-0.1546360	0.1413060
18	0.1198520	0.1239080
19	0.0394791	0.1189840
20	-0.0282063	0.0741014

(b)

Table 8: Average and Variances of the First 20 Mel-Frequency Cepstral Coefficients for (a) *Beautiful Disaster* and (b) *Ornithology*

Again, vector ***u*** in equation (4) represents the 20 average values and 20 variance values for (a) *Beautiful Disaster*, while ***v*** represents the 20 average values and 20 variance values for (b) Ornithology.

The term ***(u – v)*** is the difference between the corresponding values in the *Beautiful Disaster* vector and the *Ornithology* vector as displayed in Table 9.

Difference Matrix *(u – v)*	
1	2.9107100
2	-1.0308400
…	…
39	-0.0745063
40	-0.0277796

(a)

	Inverse Covariance Matrix $\sum^{-1}$				
	1	**2**	**…**	**39**	**40**
1	0.513781932	0.432328116	…	1.368893943	-1.077001276
2	0.432328116	2.874080794	…	-6.381114659	-3.589329695
…	…	…	…	…	…
39	1.368893943	-6.381114659	…	9057.864846	-4671.120479
40	-1.077001276	-3.589329695	…	-4671.120479	7997.792215

(b)

Table 9: (a) Difference Matrix (b) Inverse Covariance Matrix

Once more, the second and third parts of the Mahalanobis distance, $\sum^{-1}(u - v)$, are the product of (b) the inverse covariance matrix and (a) the difference matrix. The product can be observed in Table 10.

Next, we calculate the product of *(u – v)* transposed (a) and the product matrix (b). The result, the Mahalanobis distance, is approximately 116.139.

Difference Matrix Transposed $(u-v)^T$				
1	**2**	**…**	**39**	**40**
2.9107100	-1.0308400	…	-0.0745063	-0.0277796

(a)

Product Matrix $\sum^{-1}(u-v)$	
1	0.682875573
2	0.606882658
…	…
3	-167.8867041
4	183.6881682

(b)

Table 10: (a) Difference Matrix Transposed and (b) Product Matrix

Given that the Mahalanobis distance between *Beautiful Disaster* and *Freak Out* is about 11.518 and the Mahalanobis distance between *Beautiful Disaster* and *Ornithology* is about 116.139, it can be observed that *Freak Out is* considered to be more similar to *Beautiful Disaster* than *Ornithology* is because of the lower distance value based on content-based filtering.

3.3 Combining Tags with Acoustic Features

The proposed hybrid system utilizes similarity based on user-supplied music tags and similarity based on automatic extraction of acoustic features. As illustrated in previous examples, the proposed system uses a combined/hybrid distance score derived from preprocessed values: the weighted Euclidean distance $d_{x,y}$ based on collaborative music tags from Last.fm and the Mahalanobis distance $D_{x,y}$ based on MFCC data extracted from

each audio file via Sonic Annotator. Due to the fact that the scales for tag distance and for MFCC distance differ greatly, distance values are normalized before being combined.

The normalized tag (weighted Euclidean) distance $\overline{d_{x,y}}$ is calculated using decimal scaling normalization by dividing the weighted Euclidean distance by the maximum weighted Euclidean distance:

$$\overline{d_{x,y}} = \frac{d_{x,y}}{\max\{d_{x_i,y_j}\}}, \quad (5)$$

where $d_{x,y}$ is the weighted Euclidean distance between tracks *x* and *y* calculated in equation (1) and d_{x_i,y_j} represents the weighted Euclidean distance between all pairs of tracks in the dataset.

Similarly, the normalized MFCC (Mahalanobis) distance $\overline{D_{x,y}}$ can be calculated using decimal scaling normalization as follows:

$$\overline{D_{x,y}} = \frac{D_{x,y}}{\max\{D_{x_i,y_j}\}}. \quad (6)$$

The combined distance *C (x,y)* between two music tracks *x* and *y* can be then calculated by combining the two distance measures based on collaborative tags and acoustic features:

$$C(x,y) = \frac{1}{2}\left(\frac{\overline{d_{x,y}} - \overline{d_{x,1}}}{\overline{d_{x,20}} - \overline{d_{x,1}}} + \frac{\overline{D_{x,y}} - \overline{D_{x,1}}}{\overline{D_{x,20}} - \overline{D_{x,1}}}\right). \quad (7)$$

where $\overline{d_{x,y}}$ is the normalized tag (weighted Euclidian) distance between query track *x* and compared track *y*, $\overline{d_{x,1}}$ is the lowest normalized tag distance value between query track *x* and all music tracks in the dataset, and $\overline{d_{x,20}}$ is the 20th lowest normalized tag distance value between query track *x* and all music tracks in the dataset. Likewise, $\overline{D_{x,y}}$ is the normalized MFCC (Mahalanobis) distance between music tracks *x* and *y*.

For cases when the tag (weighted Euclidian) distance between two tracks does not exist because one or both tracks do not have any top tags from Last.fm, the combined distance C*(x,y)* between two music tracks *x* and *y* is modified so that it only depends on the MFCC distance:

$$C(x,y) = \frac{\overline{D_{x,y}} - \overline{D_{x,1}}}{\overline{D_{x,20}} - \overline{D_{x,1}}}. \quad (8)$$

Finally, the proposed hybrid system will output several recommendations, each as text (track title and artist). Recommendations will be prioritized by highest level of similarity; lower distance correlates to higher similarity.

For an example of similarity based on a combination of Last.fm top tags and MFCCs, let us use the query track, *Ornithology* by Charlie Parker (id 74709), and the compared track, *Blue Train* by John Coltrane (id 73514).

As calculated in section 3.2.1, we know that the tag (weighted Euclidean) distance between *Ornithology* and *Blue Train* is about 0.103. Referring to section 3.2.2, the MFCC (Mahalanobis) distance between these two tracks is about 84.712.

Query Track *(x)*	**Compared Track** *(y)*	**tag distance** $d_{(x,y)}$	**max tag Distance**	**normalized tag distance** $\overline{d_{x,y}}$
Ornithology	*Blue Train*	0.102673063	1.414213569	0.072600819

Query Track *(x)*	**Compared Track** *(y)*	**MFCC distance** $D_{(x,y)}$	**max MFCC Distance**	**normalized MFCC distance** $\overline{D_{x,y}}$
Ornithology	*Blue Train*	84.71246317	8075.253131	0.010490379

Table 11: Normalized Distance between *Ornithology* and *Blue Train*

The maximum tag distance value (a constant) in the dataset can be retrieved from the database and is approximately 1.414. Likewise, the maximum MFCC distance value (a constant) in the dataset is approximately 8,075.253. As referenced in equation (5), the normalized tag distance between *Ornithology* and *Blue Train* $d_{x,y}$ is approximately 0.073. Similarly, as per equation (6), the normalized MFCC distance between these tracks $\overline{D_{x,y}}$ is approximately 0.010.

By querying the database, we can retrieve the lowest normalized tag distance value $d_{x,1}$ and the twentieth lowest normalized tag distance value $d_{x,20}$ between a query track and every other music track in the dataset. In this example, the lowest normalized tag distance is approximately 0.073 (which happens to the normalized tag distance between *Ornithology* and *Blue Train*), and the twentieth lowest value is approximately 0.127.

Also via database query, the lowest normalized MFCC distance is approximately 0.007, and the twentieth lowest value is approximately 0.008.

Query Track (x)	**Compared Track** (y)	**normalized tag distance** $\overline{d_{x,y}}$	**lowest normalized tag distance** $\overline{d_{x,1}}$	**twentieth lowest normalized tag distance** $\overline{d_{x,20}}$
Ornithology	*Blue Train*	0.072600819	0.072600819	0.127092709

Query Track (x)	**Compared Track** (y)	**normalized MFCC distance** $\overline{D_{x,y}}$	**lowest normalized MFCC distance** $\overline{D_{x,1}}$	**twentieth lowest normalized MFCC distance** $\overline{D_{x,20}}$
Ornithology	*Blue Train*	0.010490379	0.006545963	0.008033416

Table 12: Lowest Distance Values between *Ornithology* and *Blue Train*

At this point, we can calculate the combined/hybrid distance between *Ornithology* and *Blue Train* using equation (7):

$$C(x,y) = \frac{1}{2}\left(\frac{0.073-0.073}{0.127-0.073} + \frac{0.01-0.007}{0.008-0.007}\right) \cong 1.326.$$

For another example of similarity based on a combination of Last.fm top tags and MFCCs, let us use the query track, *Ornithology* by Charlie Parker (id 74709), and the compared track, *Beautiful Disaster* by 311 (id 69134).

Referring again to section 3.2.1, the tag (weighted Euclidean) distance between *Ornithology* and *Beautiful Disaster* is about 0.355. As calculated in section 3.2.2, we know that the MFCC (Mahalanobis) distance between these two tracks is about 116.139.

Query Track (x)	Compared Track (y)	tag distance $d_{(x,y)}$	max tag Distance	normalized tag distance $\overline{d_{x,y}}$
Ornithology	*Beautiful Disaster*	0.354525013	1.414213569	0.250687039

Query Track (x)	Compared Track (y)	MFCC distance $D(x,y)$	max MFCC Distance	normalized MFCC distance $\overline{D_{x,y}}$
Ornithology	*Beautiful Disaster*	116.1388483	8075.253131	0.014382069

Table 13: Normalized Distance between *Ornithology* and *Beautiful Disaster*

As in the previous example, the maximum tag distance value is approximately 1.414, and the maximum MFCC distance value is approximately 8,075.253. Per equation (5), the normalized tag distance between *Ornithology* and *Beautiful Disaster* is approximately 0.251. Similarly, referencing equation (6), the normalized MFCC distance between these tracks is approximately 0.014.

Query Track (x)	Compared Track (y)	normalized tag distance $\overline{d_{x,y}}$	lowest normalized tag distance $\overline{d_{x,1}}$	twentieth lowest normalized tag distance $\overline{d_{x,20}}$
Ornithology	*Beautiful Disaster*	0.250687039	0.072600819	0.127092709

Query Track (x)	Compared Track (y)	normalized MFCC distance $\overline{D_{x,y}}$	lowest normalized MFCC distance $\overline{D_{x,1}}$	twentieth lowest normalized MFCC distance $\overline{D_{x,20}}$
Ornithology	*Beautiful Disaster*	0.014382069	0.006545963	0.008033416

Table 14: Lowest Distance Values between *Ornithology* and *Beautiful Disaster*

As with the same query track in the previous example, the lowest normalized tag distance is approximately 0.073 and the twentieth lowest value is approximately 0.127. The

lowest normalized MFCC distance is approximately 0.007 and the twentieth lowest value is approximately 0.008.

At this point, we can calculate the combined/hybrid distance between *Ornithology* and *Blue Train* using equation (7):

$$C(x,y) = \frac{1}{2}\left(\frac{0.251-0.073}{0.127-0.073} + \frac{0.014-0.007}{0.008-0.007}\right) \cong 4.268.$$

Given that the combined distance between *Ornithology* and *Blue Train* is about 1.326 and the combined distance between *Ornithology* and *Beautiful Disaster* is about 4.268, it can be observed that *Blue Train is* considered to be more similar to *Ornithology* than B*eautiful Disaster* is because of the lower distance value based on a combination of collaborative and content-based filtering.

Chapter 4

SYSTEM DESIGN AND EXPERIMENT

Figure 9 illustrates the basic data flow of the hybrid music recommendation system. The database contains information on more than 15,000 music tracks: mp3 audio files and their metadata (track name, artist, and file name). The track name and artist are used to extract community tags (top tags) via Last.fm's API method, *track.getTopTags*. The audio file is used to extract MFCC data (acoustic features) via Sonic Annotator.

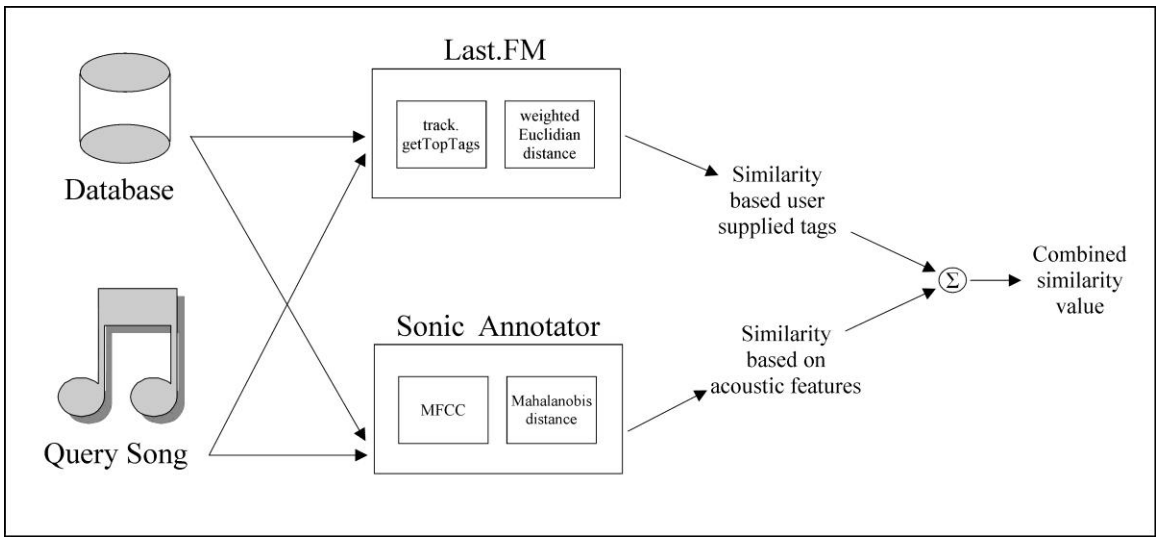


Figure 9: Flowchart of Proposed System

The query song is selected by the user. For each of tags and MFCC, a similarity value is calculated by comparing data of the query song to data of each song in the database. A combined, weighted similarity value is calculated based on tag similarity value and

MFCC similarity value. A list of recommended music tracks is provided by the system based the selected query track.

In order to evaluate the proposed hybrid approach, the system is broken up into three versions: subsystem 1 utilizes only Last.fm collaborative music tags (collaborative filtering), subsystem 2 utilizes only acoustic features (content-based filtering), and subsystem 3 (hybrid approach) utilizes a combination of both music tags and acoustic features. The three subsystems were evaluated based on user opinions on the level of similarity between the query track and the recommendations provided by the system.

4.1 System Design

In order to successfully build the hybrid music recommendation system, the job was broken up into many tasks as shown in Table 15. First, a large dataset of diverse music was acquired from University of North Florida's radio station, Spinnaker Radio. XAMPP, an open source Apache distribution package was used to design a MySQL relational database.

Figure 10 gives an idea of the structure of the MySQL relational database utilized in this project. The main table*, music track*, is where metadata from mp3 audio tracks was cleaned up and imported. Data relating to Last.fm top tags is stored in the *top tag* table. Mel-frequency Cepstral Coefficients extracted from the audio are stored in the *MFCC data* table. Distance values used for similarity based on collaborative filtering, content-based filtering, and the hybrid approach are stored in the *similarity* table. User ratings

collected from participants who used the system were stored anonymously in the session table for further analysis. The database was tuned to perform well with a large dataset.

Task Required	Details of Completion
Legally acquire large dataset of diverse mp3 music tracks	Acquired over 30,000 mp3s from UNF Spinnaker Radio
Design relational database	Designed a MySQL database with XAMPP; configured database to perform well with large amount of data
Obtain metadata from tracks	Removed duplicates, cleaned metadata, and imported data about each mp3 into database
Obtain collaborative tags	Developed PHP script that connects to Last.fm API, retrieves top ten tags for each song, and inserts them into database
Obtain data about acoustic features	Customized a batch file that uses Sonic Annotator to extract MFCC data and export to CSV.
Calculate similarity between all tracks based on collaborative data	Developed complex script that calculates distance using weighted Euclidian Distance algorithm based on top tags.
Calculate similarity between all tracks based on acoustic features	Developed complex script that calculates distance using Mahalanobis Distance algorithm based on MFCC data.
Calculate hybrid similarity between all tracks based on previous two	Developed complex script that calculates combined distance based on normalized top tag and MFCC distance values
Design website allowing participants to use system	Developed front-end using HTML5, CSS, JavaScript, and jQuery (DataTables)
Conduct experiment(s) to gather feedback and to fine tune system	Adjusted hybrid approach to recommendation based on user ratings from first experiments

Table 15: System Design Tasks

Figure 10: Database Schema for Hybrid Recommendation System

Next, scripts were developed to load the database with Last.fm top tags, and MFCC acoustic data. Additional scripts were developed to calculate similarity based on each set of data and also based on a combination of both. Refer to Appendix C to view the implemented functionality, mostly in PHP.

A web-based application was designed from the ground up to allow users to access and use the hybrid recommendation system.

4.1.1 The Web Application

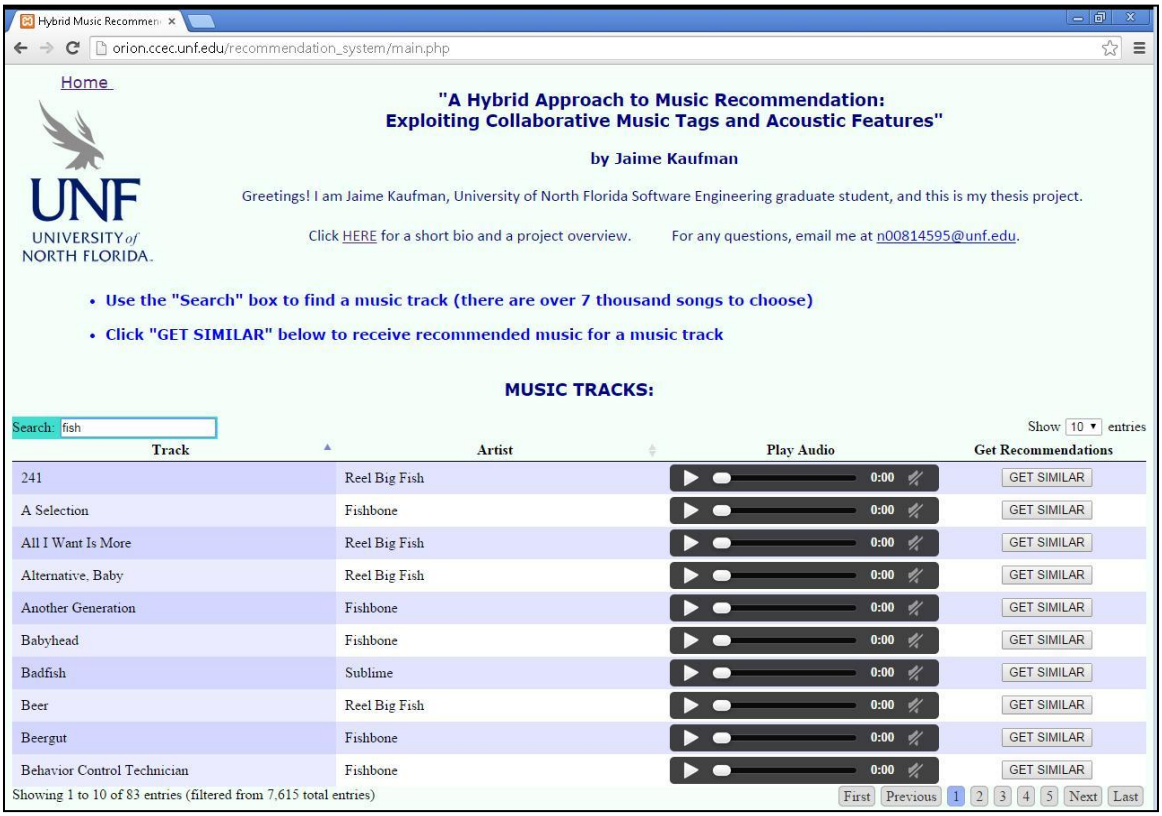


Figure 11: Web Application Main Page

Figure 11 displays the homepage of the web application. The page includes basic information and a link to a project overview and bio page. There are basic directions on what is available on this page. A user has the ability to search on the fly by music track or artist name, sort ascending or descending by track or artist, specify the number of entries to display, and navigate to additional pages of audio tracks. One can listen to an audio sample of any of thousands of tracks available. A user clicks the *GET SIMILAR* button next to a track to choose what is referred to as the query track.

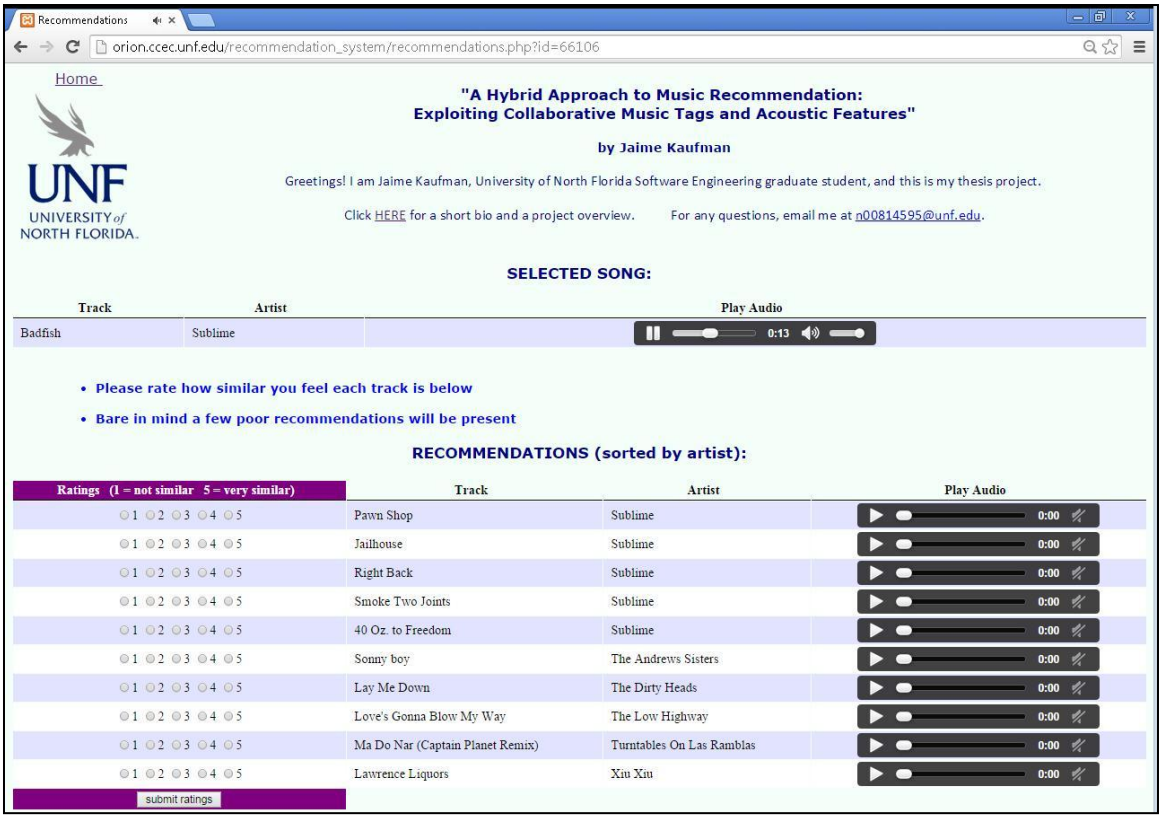


Figure 12: Web Application Recommendations Page

Based on the music track a user selects, the system provides a list of music recommendations as shown in Figure 12. Of the supplied recommendations, three are based on Last.fm top tags, three are based on extracted MFCCs, three are based on the hybrid approach, and three are intentionally poor recommendations based on the hybrid approach. Given that there may be music tracks recommended from more than one subsystem, any duplicates are tracked and consolidated. It can be observed that users were not able to identify which tracks were recommended based on which approach. The recommendations page includes directions on how to provide ratings for each of the recommended tracks. The user has the ability to listen to a sample of each recommendation, as well as the selected query track at the top. After listening to the

audio, a user will rate each recommended track on a scale of 1 to 5, where 5 means the track is very similar to the query track selected and 1 means the track is very dissimilar to the query track.

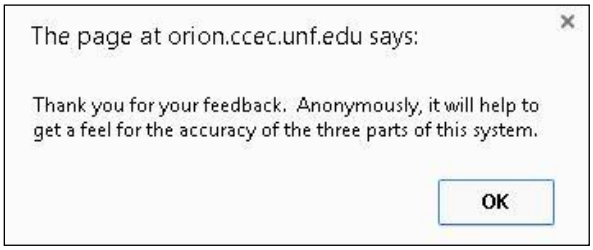


Figure 13: Web Application User Ratings Received

Once a value is selected for each recommendation, the *submit ratings* button is clicked to record the feedback anonymously to the database. As shown in Figure 16, a message will be displayed indicating the ratings were successfully received and recorded in the database.

4.2 Experiment: Assessing System Recommendations

In order to validate that the recommendations from the system provide a high level of satisfaction in regard to similarity, an experiment was performed. All University of North Florida staff and students had an opportunity to participate. The University of North Florida Institutional Review Board (IRB) reviewed this project and declared it "not research involving human subjects." Refer to Appendix B to view the instructions supplied to the potential participants. Each user accessed the web application and selected a query track from the dataset. Based on that selection, the system provided

three recommendations based on collaborative filtering, three on content-based filtering, three on the hybrid approach (a combination of the two), and three intentionally bad recommendations.

4.2.1 Different Approaches Employed

The experiment was conducted several times in order to fine tune the hybrid subsystem. For the first set, the hybrid distance values were calculated based on an average of the normalized tag and MFCC distance. This heavily favored the MFCC distance due to the distributions of values being lower for MFCC. For the next set, the hybrid distance values were calculated based on a maximum and minimum value from the lowest/best twenty normalized tag and MFCC distances. Unfortunately, this also favored the MFCC distance. For the final run, the previous method was used, and two additional requirements were included. Two of the three hybrid recommended tracks must be in the top twenty for tag distance, and one track must be in the top twenty for MFCC distance and not have tags. This is to ensure quality of recommendations while providing a higher chance of musical discovery.

4.2.2 Data Collection

Several requests were made urging individuals to participate in an experiment. Requests were made to the University of North Florida School of Computing, to the School of Music, and ultimately to the entire faculty and student body at UNF. Several experiments were carried out. For each experiment, 100 sets of user ratings were collected. The web

application was made available for data collection for a total of approximately three months. All user ratings provided were stored anonymously in the database.

Based on the query track selected by the user, the system provided three recommendations based on tag distance (collaborative filtering), three based on MFCC distance (content-based filtering), three based on a combination of the two (hybrid approach), and three intentionally poor recommendations. A participant rated each recommended track on a scale of 1 to 5, where, in his or her opinion, 5 is very similar to the query song and 1 is very dissimilar.

Chapter 5

RESULTS AND EXPLANATION

5.1 Summary of Results

Figure 14 is a summary of results from the final experiment. Refer to the Appendix A to view the table that details a breakdown of all user ratings from the final experiment.

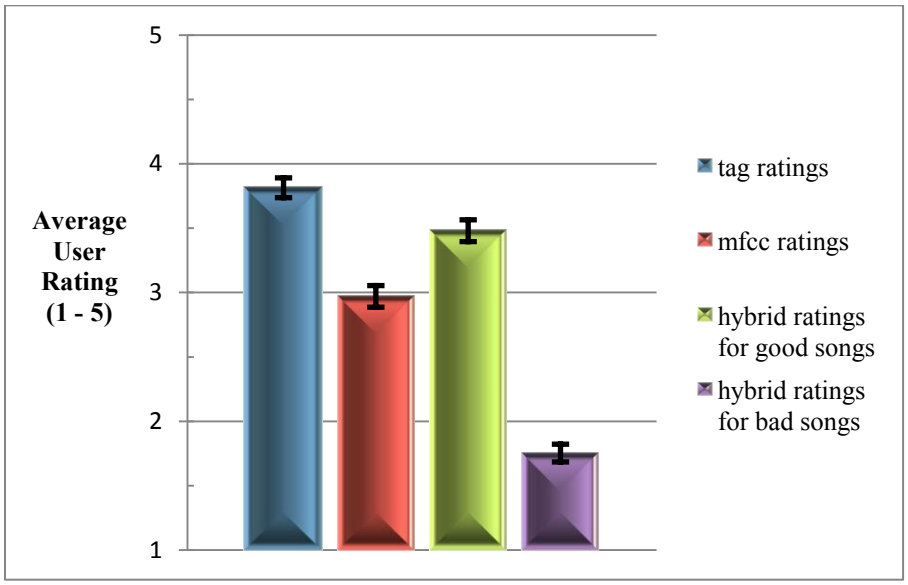


Figure 14: Results of the Four Systems in Average User Rating with Standard Errors

With an average score of about 3.81 out of 5, collaborative-based recommendations received the most favorable user feedback with respect to similarity. The proposed hybrid approach of recommendations based on collaborative filtering and content-based filtering was next with an average score of 3.48 (p-value $\approx 0.004 < 0.05$, i.e., the

difference between the average rating of the tag-based and hybrid-based systems is statistically significant). Recommended music based on acoustic features was third, receiving an average score of 2.97 out of 5 (p-value ≈ $2.176E^{-5}$ when comparing the hybrid-based and MFCC-based systems).

Recommendations intentionally given with poor hybrid similarity values received an average score of about 1.75 out of 5 (p-value ≈ $1.315E^{-26}$ when comparing the MFCC-based and intentionally poor hybrid systems).

5.2 Usefulness of Top Tags

	tagName	related to	average rating	# times tag listed
1	rock	genre	3.7799	159
2	pop	genre	4.1250	104
3	hard rock	genre	3.7215	79
4	classic rock	genre	3.6753	77
5	alternative	genre	3.7536	69
6	female vocalists	feature of artist	4.1379	58
7	soul	genre	4.1481	54
8	80s	genre	4.0556	54
9	alternative rock	genre	3.8462	52
10	70s	genre	3.6136	44
11	oldies	genre	3.4750	40
12	dance	genre	4.4103	39
13	british	genre	3.7500	32
14	00s	genre	4.2581	31
15	male vocalists	feature of artist	4.4138	29
16	90s	genre	4.2069	29
17	american	genre	4.6071	28
18	singer-songwriter	feature of artist	4.3214	28
19	jazz	genre	4.0370	27
20	heavy metal	genre	4.0769	26

Table 16: Breakdown of Last.fm Top Tags

Table 16 illustrates the usefulness of the top tags supplied by a large community of users at Last.fm. The top tags are sorted by how often a tag was used to calculate similarity in the experiment. The most popular top tags look to be related to the genre or style of a music track.

Based on the results, it appears that the commonality of a tag is not directly related to how favorable the response is from a user. Interestingly, some top tags from the experiment resembled the music track, artist, or album. This may have given recommendations based on top tags an unfair advantage over recommendations based on the acoustic features.

5.3 Usefulness of the Hybrid Approach

Included in Table 17 is a summary associated with all hybrid recommendations that were not also recommendations based on tags or acoustic features. Given that the average rating for this subgroup is comparable and even higher than the average rating for all recommendations based on the hybrid approach, it is evident that the proposed hybrid approach to music recommendation utilizing both top tags and extracted features shows much promise. However, the difference is not statistically significant (p-value ≈ 0.374).

	average rating	total tracks	standard error
hybrid recommended tracks not included in top three based on either tags or MFCCs	3.6198	121	0.130991646
all hybrid recommended tracks	3.4800	300	0.084658143

Table 17: Summary of Hybrid Recommended Tracks

Table 18 consists of details of five examples from the experiment to highlight the benefits of the proposed hybrid approach to music recommendation. Of special note, all five were provided as top three recommendations by the hybrid approach but not as top three by the other individual approaches.

query track chosen by user		track recommended by hybrid approach		user rating provided	reason music recommendation based on hybrid approach is appropriate	shared Last.fm top tags
track	**Artist**	**track**	**artist**			
Until the End of Time	Justin Timberlake	Twenty Foreplay	Janet Jackson	5	similar style/genre (slow R&B), same key (B minor), similar chord progressions, same timbre (instrumentation/vocals)	pop, r&b, sexy, soul
Just Got Lucky	Dokken	Mama Weer All Crazee Now	Quiet Riot	5	similar style/genre (hard rock), same tempo (quarter note = 138), similar key (C# major and D major), similar timbre (lead guitar sound, etc.)	hard rock, heavy metal, hair metal, 80s
Baby One More Time	Britney Spears	Crash	Gwen Stefani	4	similar style/genre (pop), similar drum beat, similar timbre (bass, vocals)	pop, dance, female vocalists, sexy
Autumn Leaves	Cannonball Adderley Feat. Miles Davis	Song for My Father	Horace Silver	4	similar style/genre (cool jazz), similar timbre (instrumentation)	jazz (query track only has one top tag)
Rumour Has It	Adele	Rolling in the Deep	Adele	5	similar style/genre (soul / pop), same artist/band (Adele), similar drum beat, similar timbre (instrumentation/vocals)	soul, pop, Adele, female vocalists, British, singer-songwriter

Table 18: Five Examples of Recommendations Explained

In the first example, *Until the End of Time* by Justin Timberlake was the query track, and *Twenty Foreplay* by Janet Jackson was recommended. When analyzing the two songs manually, it is apparent that they are both in the same style/genre, are in the same key, have very similar chord progressions, and have very similar timbre (quality of sound). The Janet Jackson track was recommended by the hybrid approach for two reasons: both

tracks share the top tags, *pop*, *r&b*, *sexy*, and *soul*, and both tracks share similar acoustic features (MFCC data).

For example two, *Just Got Lucky* by Dokken was selected by the user, and *Mama Weer All Crazee Now* by Quiet Riot was recommended. After manual analysis, it is evident they are both in the same style/genre, have the same tempo, are within a half step of the same key, and have similar timbre. The Quiet Riot song was recommended by the hybrid approach because both tracks share the top tags, *hard rock*, *heavy metal*, *hair metal*, and *80s*, and both tracks share similar acoustic features based on MFCCs.

In example three, *Baby One More Time* by Britney Spears was the input track, and *Crash* by Gwen Stefani was a recommended track. After listening and comparing the two tracks, it is clear both are in the same style/genre, have a similar drum beat, and have similar timbre. The track by Gwen Stefani was recommended by the hybrid approach because both tracks share the top tags, *pop*, *dance*, *female vocalists*, and *sexy*, and both tracks share similar acoustic features.

In the fourth example, *Autumn Leaves* featuring Miles Davis was selected, and *Song for My Father* performed by Horace Silver was recommended. When analyzing these songs, one can come to the conclusion that they are both in the same style/genre and have similar instrumentation/timbre. Given that the *Autumn Leaves* only has one top tag, *jazz*, it is obvious that this song has not had much discovery or activity on Last.fm. As a result, the collaborative-based approach would not have performed well with this query

track. The hybrid approach, however, did an excellent job in this instance of recommending another jazz song and thus effectively tackled the Cold Start problem.

In example five, *Rumour Has It* by Adele was the query track, and *Rolling in the Deep* also by Adele was a recommendation. After manual analysis, it is observed that both tracks are in the same style/genre, share the same artist/band, have a similar drum beat, and have similar timbre. *Rolling in the Deep* was recommended by the hybrid approach because both tracks share the top tags, *soul*, *pop*, *Adele*, *female vocalists*, *British*, and *singer-songwriter*, and both tracks share similar acoustic features.

Overall when using the hybrid approach, it appears there is a higher chance that a recommendation is a new discovery due to the fact that it is not exclusively dependent on music that has already been heard or discovered.

Chapter 6

DISCUSSION AND FUTURE WORK

6.1 Discussion

Based on overall analysis of the data from the experiment, the method using only the collaborative filtering approach with Last.fm top tags was the most accurate in regard to similarity. This is for a number of reasons. Firstly, Last.fm has millions of registered users. This is helpful because performance improves as the amount of active users increases. However, at times, it was observed that tags were related to the track name, artist name, or album. This most likely affected the user ratings, given that different songs by the same artist are frequently considered to be similar.

The proposed hybrid approach to music recommendation utilizing collaborative filtering and content-based filtering techniques was successful and a step in the right direction. However, recommendations were not always accurate due to the dataset totaling only about 15 thousand audio tracks and lacking more diversity. This issue can be observed in the figure 15. In this example, the selected query track is a jazz, medium-swing Christmas song performed by Ella Fitzgerald. This is the only song by Ella Fitzgerald in the dataset. In addition, there are not many jazz, medium-swing, vocal tracks in the dataset. In fact, only about 3.821% of all music tracks in the dataset with tags have a "jazz" tag. Consequently, the recommendations provided were not as accurate.

Figure 15: Example of Lack of Diversity

For example, the hybrid recommendation, *What Christmas Means to Me* by Hanson, is more of a pop rock song. Recommendations based on each individual approach also suffered from the same problem. A recommendation based on tags, *Christmas Time* by the Backstreet Boys, is in the pop rock genre. This track was recommended because it is also a Christmas song. A recommendation based on MFCCs, *Know You Now* by Amy Winehouse, is more of a soul or R&B track. This track was recommended perhaps because her voice is similar in pitch range and sound quality to that of Ella Fitzgerald. However, these recommendations are not very similar to the query track. More specifically, they are in entirely different styles and tempos.

6.2 Future Work

Using a much larger and more diverse collection of music tracks will be helpful in future in this area. It should help to achieve a more accurate measure of the accuracy of various recommendation systems.

With regard to the collaborative approach, it would be beneficial in future work to develop a method to filter out these tags (i.e., noise) that closely resemble the track name or artist name, etc. This would give a more accurate measure to compare this approach to other approaches.

In general, the content-based approach with automatic extraction of MFCCs performed well. However, it was the least accurate of the three subsystems tested. This is to be expected being that this approach deals only with the frequencies of sounds. The future for music recommendation based on automatic extraction of acoustic features is indeed bright. Other musical features like instrumentation and rhythmic patterns should be explored further. Due to scalability issues, the automatic approach may even eventually replace Pandora's current method: music recommendation based on manual extraction of acoustic features. However, future work in fine tuning this approach shows promise. Incorporating machine learning is an additional method that could prove very useful in improving the accuracy of this type of approach. For example, the system could have the capability of adjusting recommendations based on ratings supplied by the user. This would make the recommendation system more personalized to each user.

REFERENCES

Print Publications:

[Basu98]
Basu, C., Hirsh, H., and Cohen, W., "Recommendation as Classification: Using Social and Content-based Information in Recommendation," *in Proceedings of AAAI (Association for the Advancement for Artificial Intelligence)*, 1998.

[Breese98]
Breese, J. S., Heckerman, D., and Kadie, C., "Empirical Analysis of Predictive Algorithm for Collaborative Filtering," *Uncertainty in Artificial Intelligence, 14th Conference on*, pp. 43-52, 1998.

[Cacheda11]
Cacheda, F., Carneiro V., Fernandez, D., Formoso, V., "Comparison of Collaborative Filtering Algorithms: Limitations of Current Techniques and Proposals for Scalable, High-Performance Recommender Systems," *the Web, ACM Transactions on*, vol. 5, issue 1, no. 2, 2011.

[Chuan13]
Chuan, C. H., "Audio Classification and Retrieval Using Wavelets and Gaussian Mixture Models," *Multimedia Engineering and Management, International Journal of*, vol. 4, issue 1, 2013.

[Chen01]
Chen, H. C., and Chen, A. L., "A Music Recommendation System Based on Music Data Grouping and User Interests," *Information and Knowledge Management, 10th ACM International Conference on*, pp. 231-238, 2001.

[Cohen00]
Cohen, W. W., and Fan, W., "Web-collaborative Filtering: Recommending Music by Crawling the Web," *Computer Networks*, vol. 33, no. 1, pp. 685-698, 2000.

[Deza09]
Deza, M. M. and Dez, E., Encyclopedia of Distances, Springer, New York, pp. 94, 2009.

[Eck08]
Eck, D., Lamere, P., Bertin-Mahieux, T., and Green, S., "Automatic Generation of Social Tags for Music Recommendation," *Advances in Neural Information Processing Systems*, pp. 385-392, 2008.

[Ekstrand10]
Ekstrand, M. D., Riedl, J. T., and Konstan, J. A., "Collaborative Filtering Recommender System," *Foundations and Trends in Human-Computer Interaction*, vol. 4, no. 2, pp. 81 – 173, 2010.

[Golder06]
Golder, S. A. and Huberman, B. A., "Usage Patterns of Collaborative Tagging Systems," *Information Science, Journal of*, vol. 32, pp. 198–208, 2006.

[Hameed12]
Hameed, M. A., Jadaan, O. A., and Ramachandram, S. "Collaborative Filtering Based Recommendation System: A Survey," *Computer Science and Engineering, International Journal on*, vol. 4, no. 5, 2012.

[Herlocker04]
Herlocker, J. L., Konstan, J. A., Terveen, L. G. and Riedl, J. T., "Evaluating Collaborative Filtering Recommender System," *Information Systems, ACM Transactions on*, vol. 22, issue 1, pp. 5-53, 2004.

[Huang06]
Huang, Z., Zeng, D., and Chen, H., "A Comparative Study of Recommendation Algorithms for E-Commerce Applications," *IEEE Intelligent Systems*, 2006.

[IFPI13]
IFPI, "IFPI Digital Music Report 2013: Engine of a Digital World," http://www.ifpi.org/content/library/DMR2013.pdf, pp. 6 and 17, 2013.

[Jannach10]
Jannach, D., Zanker, M., Felfernig, A., and Friedrich, G., Recommender Systems: an Introduction, Campbridge University Press, New York, 2010.

[Jenson07]
Jenson J., Ellis, D., Christensen, M., and Jensen, S., "Evaluation of Distance Measures between Gaussian Mixture Models of MFCCs," *Music Information Retrieval, 8th International Conference on*, pp. 107-108, 2007.

[Knees07]
Knees, P., Pohle, T., Schedl, M., and Widmer, G., "A Music Search Engine Built upon Audio-based and Web-based Similarity Measures," *Research and Development in Information Retrieval, International ACM SIGIR Conference on*, pp. 447-454, 2007.

[Lee12]
Lee, J., Sun, M., and Lebanon, G., "A Comparative Study of Collaborative Filtering Algorithms," *ArXiv Report 1205.3193*, 2012.

[Levy06]
Levy, M. and Sandler, M., "Lightweight Measures for Timbral Similarity of Musical Audio," *Audio and Music Computing Multimedia, 1st ACM Workshop on*, 2006.

[Levy09]
Levy, M. and Sandler, M., "Music Information Retrieval Using Social Tags and Audio," *Multimedia, IEEE Transactions on*, vol. 11, no. 3, pp. 383-395, 2009.

[Lops11]
Lops, P., De Gemmis, M., and Semeraro, G., "Content-based Recommender Systems: State of the Art and Trends," Ricci, F., L. Rokach, Shapira, B., and Kantor, P. (Eds.), Recommender Systems Handbook, Springer, New York, 2011.

[Mandel05]
Mandel, M. and Ellis, D., "Song-Level Features and Support Vector Machines for Music Classification," *Music Information Retrieval, 6th International Conference on*, pp. 594-599, 2005.

[Mooney00]
Mooney, R. J., and Roy, J., "Content-based Book Recommending Using Learning for Text Categorization," *Digital Libraries, 5th ACM Conference on*, pp. 195 – 204, 2000.

[Pazzani07]
Pazzani, M. J. and Billsus, D. "Content-based Recommendation Systems," The Adaptive Web, Springer, Berlin Heidelberg, 2007.

[Sachan13]
Sachan A. and Richariya, V. "A Survey on Recommender Systems Based on Collaborative Filtering Technique," *Innovations in Engineering and Technology, International Journal of*, vol. 2, issue 2, 2013.

[Schafer07]
Schafer, J. B., Frankowski, D., Herlocker, J., and Sen, S. "Collaborative Filtering Recommender Systems," The Adaptive Web, Springer, Berlin Heidelberg, 2007.

[Su09]
Su, X. and Khoshgoftaar, T., "A Survey of Collaborative Filtering Techniques," *Advances in Artificial Intelligence*, 2009.

[Wu06]
Wu, X., Zhang, L., and Yu, Y., "Exploring Social Annotations for the Semantic Web," *World Wide Web, 15th International Conference on*, pp. 417-426, 2006.

Electronic Sources:

[Last.fm14]
Last.fm, "Last.fm," http://www.Last.fm/, 2014, last accessed October 25, 2014.

[Pandora14]
Pandora, "Pandora Internet Radio," http://www.pandora.com, 2014, last accessed October 25, 2014.

[Sonic Annotator14]
Sonic Annotator, "Sonic Annotator," http://www.vamp-plugins.org/sonic-annotator/, 2014, last accessed October 25, 2014.

[Sonic Visualiser14]
Sonic Visualiser, "Sonic Visualiser," http://www.sonicvisualiser.org/, 2014, last accessed October 25, 2014.

Appendix A

DETAILED BREAKDOWN OF 100 USER RATINGS FROM FINAL EXPERIMENT

Session	Query Track	Track Name	Artist Name	Tag Ratings			MFCC Ratings			Good Hybrid Ratings			Bad Hybrid Ratings		
1	69908	(You Drive Me) Crazy	Britney Spears	3	4	4	4	3	1	4	5	3	1	1	1
2	72458	Autumn Leaves	Cannonball Adderley Feat. Miles Davis	3	5	4	3	1	4	3	4	4	1	1	1
3	69939	You And Your Heart	Jack Johnson	4	4	4	3	3	1	4	3	1	3	1	1
4	67687	Government Plates	Death Grips	3	5	5	2	4	2	5	5	2	1	1	1
5	65818	Secret	Maroon 5	4	5	5	3	4	4	4	5	4	1	3	4
6	76308	Far Away Places	Bing Crosby	4	4	2	3	3	2	4	2	1	2	1	1
7	62430	Woodchoppers Ball	Woody Herman	4	3	3	1	2	1	4	3	1	1	2	1
8	76598	Just A Simple Melody	The Andrews Sisters	3	3	3	3	3	3	3	3	3	2	2	1
9	64613	Here Without You	3 Doors Down	3	4	4	4	3	3	3	4	3	3	2	3
10	76214	Wishing You Were Somehow Here Again	Andrew Lloyd Webber	5	4	5	4	4	4	4	5	4	4	4	4
11	63005	The Way I Am	Eminem	4	4	4	4	2	3	4	4	4	1	4	3
12	75040	Little Saint Nick	The Beach Boys	3	3	3	2	4	4	4	5	4	1	1	2
13	73644	Love Song	311	2	1	4	4	3	2	3	3	4	1	1	4
14	72871	What Baby Wants	Alice Cooper	4	5	4	4	4	4	5	4	4	1	2	1
15	72689	Honolulu Baby	Anton LaVey	5	2	4	2	1	5	4	1	3	4	1	1
16	76783	Forever In Blue Jeans	Neil Diamond	5	5	5	2	2	4	2	5	5	2	3	3
17	64613	Here Without You	3 Doors Down	5	4	3	2	4	1	5	3	1	1	1	1
18	66599	Without Me	Eminem	1	5	5	4	4	1	5	5	4	3	1	1
19	62045	Heartbreak Hotel	Elvis Presley	5	5	5	2	3	2	2	5	5	4	3	1
20	65322	Just Got Lucky	Dokken	5	5	5	5	5	5	5	5	5	1	5	1
21	63426	She Will Be Loved	Maroon 5	5	5	5	1	5	4	1	5	5	1	4	4
22	62592	Back In Black	AC/DC	4	4	4	5	5	4	5	4	4	1	1	1
23	63889	Guy What Takes His Time	Christina Aguilera	3	2	5	3	1	1	3	2	1	2	1	1
24	76207	You Oughta Know	Alanis Morisette	1	1	1	4	1	3	1	3	1	1	1	1
25	61965	Down	311	1	4	3	4	3	3	3	4	4	1	1	1
26	68311	Recover	Chvrches	1	1	5	1	1	3	1	1	3	2	1	1
27	76404	Let Go	Frou Frou	5	2	1	2	1	1	1	2	1	5	1	1
28	76969	By the Way	Red Hot Chili Peppers	5	5	5	1	2	1	5	5	1	1	1	1
29	68716	Rumour Has It	Adele	5	5	5	3	2	2	5	5	2	3	2	1

Session	Query Track	Track Name	Artist Name	Tag Ratings			MFCC Ratings			Good Hybrid Ratings			Bad Hybrid Ratings		
30	65793	Rainy Day Women #12 & 35	Bob Dylan	4	5	5	4	2	3	4	4	3	3	2	4
31	69078	Tighten Up	The Black Keys	5	3	5	1	2	4	2	3	5	1	4	3
32	76170	Rag Doll	Aerosmith	1	3	2	5	3	2	2	5	1	1	2	1
33	65218	Everlasting Love	U2	1	2	3	3	2	4	3	2	1	3	2	1
34	64613	Here Without You	3 Doors Down	4	4	3	3	5	3	4	3	3	1	1	3
35	63322	Metal Thrashing Mad	Anthrax	5	5	5	4	3	2	5	5	2	1	3	2
36	66501	Tell Her About It	Billy Joel	3	5	2	1	5	3	5	3	4	1	1	1
37	67425	Rock and Roll Never Forgets	Bob Seger	1	5	4	4	3	3	1	4	4	1	1	3
38	74582	Sleeping Bag	ZZ Top	4	5	3	3	3	4	3	4	3	1	1	1
39	72987	Ballad Of Dwight Fry	Alice Cooper	4	2	4	4	2	1	2	1	2	3	3	3
40	62440	You Should Be Dancing	Bee Gees	5	3	5	2	1	1	4	4	2	3	1	1
41	75199	The Audience Is Listening	Steve Vai	3	5	5	4	2	5	5	5	5	1	1	2
42	75778	I Believe	Elvis Presley	4	2	1	5	5	4	5	2	1	2	2	3
43	74450	Let Me Down Easy	2AM Club	1	1	3	4	4	2	1	2	4	1	1	2
44	76498	Buy Me A Mercedes Benz	Janis Joplin	2	2	1	3	2	3	2	2	1	1	3	3
45	76034	Hello, I Love You	The Doors	3	3	4	1	1	1	1	3	4	1	2	4
46	65405	Make Me Smile	Chicago	3	4	4	2	3	1	3	4	3	1	2	1
47	76181	How Can You Mend a Broken Heart?	Al Green	1	1	2	1	1	2	3	2	2	1	1	1
48	76706	Let's Get It On	Marvin Gaye	1	1	1	1	2	1	1	1	2	1	3	1
49	72385	There Goes My Everything	Elvis Presley	4	4	4	2	2	1	2	4	4	1	1	1
50	64588	Green Eyed Lady	Sugarloaf	1	3	1	3	1	1	1	1	1	1	1	1
51	76275	White Christmas	Bing Crosby	1	3	1	2	1	1	1	1	1	1	1	1
52	76238	Just the way you are	Barry White	3	3	3	1	1	1	1	2	1	1	1	1
53	67159	Young Americans	David Bowie	3	2	2	1	4	1	1	3	3	3	4	1
54	69892	You Make Me Real	The Doors	4	4	5	5	4	5	3	5	5	1	1	1
55	72454	Year Zero	30 Seconds To Mars	5	5	5	2	2	1	5	5	1	5	4	3
56	65720	Make You Feel My Love	Adele	5	5	3	5	4	4	5	5	4	1	1	1
57	65330	Master Of Puppets	Metallica	3	3	3	4	3	3	4	3	3	1	4	1
58	68145	Daft Punk Is Playing At My House	LCD Soundsystem	4	5	3	4	2	2	5	4	4	1	1	1
59	61890	Bohemian Rhapsody	Queen	4	5	4	2	4	1	2	5	5	1	3	4
60	72923	Thunder	Matisyahu	2	1	3	3	2	2	3	3	3	3	4	2
61	72020	Dirty Diana	Michael Jackson	1	5	5	2	2	4	2	5	5	1	1	1
62	72603	Black or White	Michael Jackson	5	3	5	3	5	4	5	3	4	1	1	1
63	64243	Thriller	Michael Jackson	5	5	5	2	4	4	2	5	4	1	1	1
64	62961	Hound Dog	Elvis Presley	5	5	4	1	1	2	1	5	4	1	4	1
65	75948	Live Wire	AC-DC	3	3	4	1	1	1	3	1	4	1	1	1

Session	Query Track	Track Name	Artist Name	Tag Ratings			MFCC Ratings			Good Hybrid Ratings			Bad Hybrid Ratings		
66	64613	Here Without You	3 Doors Down	4	2	2	3	4	2	4	2	2	1	2	2
67	68731	Shades of Truth	Bad Religion	5	5	5	5	5	3	5	5	5	1	1	1
68	64613	Here Without You	3 Doors Down	4	4	3	1	4	2	4	3	2	1	1	1
69	64613	Here Without You	3 Doors Down	4	2	3	1	3	1	4	3	1	1	1	2
70	76341	No Rain	Blind Melon	5	5	5	5	2	1	5	5	2	3	2	1
71	64613	Here Without You	3 Doors Down	5	4	4	3	2	2	5	4	2	1	2	2
72	74450	Let Me Down Easy	2AM Club	4	4	4	1	5	5	4	5	5	1	1	1
73	75212	The Mission	30 Seconds To Mars	5	5	5	5	5	5	5	1	5	1	1	1
74	73394	Madalaine	Winger	1	5	5	5	5	5	5	5	5	1	5	1
75	62592	Back In Black	AC/DC	5	5	5	5	5	5	5	5	5	1	1	1
76	75514	I Thank You	ZZ Top	5	5	5	5	5	5	5	5	5	1	1	1
77	73952	Arrested For Driving While Blind	ZZ Top	5	5	4	5	5	5	5	2	5	1	1	1
78	66332	Lovesong	Adele	5	5	5	5	5	2	5	5	1	5	1	1
79	71968	Black Cat	Janet Jackson	5	5	5	4	4	1	5	4	1	1	1	3
80	73328	Come Back to Me	Janet Jackson	5	5	5	1	4	1	5	5	4	4	4	1
81	64133	Set Fire to the Rain	Adele	4	5	4	3	4	4	4	4	3	1	1	1
82	61968	Dream	Michael Buble	5	5	4	5	5	5	5	4	5	1	1	1
83	66820	Obvious	Christina Aguilera	5	5	5	4	5	2	5	5	5	1	5	4
84	66570	Until the End of Time	Justin Timberlake	5	5	4	4	4	1	4	5	5	1	1	4
85	69908	(You Drive Me) Crazy	Britney Spears	5	5	5	5	5	1	5	4	5	1	1	1
86	64243	Thriller	Michael Jackson	5	5	5	4	4	4	4	5	5	1	1	1
87	68632	O Come All Ye Faithful	Amy Grant	4	4	4	4	4	5	4	5	1	4	1	1
88	69896	You Upset Me Baby	B.B. King	5	4	5	4	1	1	5	5	1	4	1	5
89	69548	Nasty	Janet Jackson	5	5	5	4	5	4	1	5	5	1	4	1
90	68975	...Baby One More Time	Britney Spears	4	5	5	5	1	4	5	4	4	1	1	4
91	75594	Jump Jive And Wail	Brian Setzer Orchestra	5	5	4	1	1	4	5	5	1	4	1	4
92	75740	Antelope	The Dirty Heads	5	1	5	4	5	5	5	5	5	4	4	1
93	68716	Rumour Has It	Adele	5	5	5	5	1	1	5	5	1	5	5	1
94	61989	Fever	Michael Buble	5	5	4	1	5	4	1	5	4	1	5	1
95	71975	Brave New Girl	Britney Spears	5	5	5	5	4	1	5	5	5	1	1	1
96	74787	I Do Not Want This	Nine Inch Nails	2	2	2	3	2	1	2	2	1	1	2	1
97	74748	Simple Man	Lynyrd Skynyrd	2	2	2	2	2	1	2	2	2	1	1	3
98	65547	Enter Sandman	Metallica	5	5	5	5	5	5	5	5	5	1	1	1
99	69967	Attitude Dance	Tower of Power	5	3	3	4	4	5	5	3	4	1	1	1
100	62265	Rolling in the Deep	Adele	3	4	2	3	2	2	3	4	2	1	1	3

Average	**3.81333333**	**2.97**	**3.48**	**1.75333333**
Standard Deviation	**1.33332219**	**1.4523997**	**1.46632204**	**1.1932736**
Count	**300**	**300**	**300**	**300**
Standard Error	**0.07697939**	**0.08385434**	**0.08465814**	**0.06889368**

Appendix B

EXPERIMENT INSTRUCTIONS TO PARTICIPANTS

UNF student or staff member:

My name is Jaime Kaufman, and I am a graduate student in the School of Computing currently working on a Master's thesis titled "A Hybrid Approach to Recommendation: Exploiting Collaborative Music Tags and Acoustic Features." I request a small amount of your time to assist me in the evaluation of the system I have developed.

The system allows you to search by song or artist to select a music track. You then click GET SIMILAR, in response to which the system will provide several recommendations. You will then be asked to submit ratings (score of 1-5) for each song recommended. Feedback will be stored anonymously.

The more feedback that can be gathered in this experiment, the more accurate the results will be. The process takes less than five minutes to complete.

I greatly appreciate your assistance in evaluating and validating my system. Thank you so much for your participation!

Appendix C

IMPLEMENTED CODE SNIPPETS

Retrieve Recommendations – PHP/MySQL Script from Web Application:

```
<?php

   $queryTrackInfo = mysql_query
      (
      "SELECT * FROM music_track WHERE trackID = " . $queryTrackID . ";"
      )
      or die('ERROR SELECTING TABLE<br>queryTrackID = ' . $queryTrackID . '<br>' .
mysql_error());

   // retrieve entire row of data AND artist name separately from query track selected by user
   $row = mysql_fetch_array( $queryTrackInfo );
   $queryArtist = $row['artistName'];

   // retrieve top 3 recommendations (excluding identical song) based on tags, mfcc, hybrid, and bad ones
   // remove duplicates but keep track of them ! ! ! =)

   //
**********************************************************************************
**********************************************
   $recommendations_SQL_statement =
      "SELECT trackID, trackName, artistName, fileName, SUM(whichSystem) AS whichSystems,
distance FROM
      (
         (
            -- ****************************************
            -- best 3 based on top tags              *
            -- ****************************************
            SELECT trackID, trackName, artistName, fileName, 1 AS whichSystem, tagDistance AS
distance
            FROM music_track
            INNER JOIN similarity ON trackID = comparedTrackID
            WHERE queryTrackID = " . $queryTrackID . " AND tagDistance > 0
            ORDER BY tagDistance LIMIT 3
         )
         UNION
         (
            -- ****************************************
            -- best 3 based on mfcc's                *
            -- ****************************************
            SELECT trackID, trackName, artistName, fileName, 10 AS whichSystem, mfccDistance AS
distance
            FROM music_track
            INNER JOIN similarity ON trackID = comparedTrackID
```

```
            WHERE queryTrackID = " . $queryTrackID . " AND mfccDistance > 0
            ORDER BY mfccDistance LIMIT 3
        )
        UNION
        (
            -- ****************************************
            -- best 3 based on tags and mfcc's    *
            -- USE ONE WITH NO TAGS               *
            -- ****************************************
            SELECT trackID, trackName, artistName, fileName, 100 AS whichSystem, combinedDistance
                AS distance FROM music_track
            INNER JOIN similarity ON trackID = comparedTrackID
            WHERE queryTrackID = " . $queryTrackID . " AND combinedDistance >= 0 AND tagDistance
IS NULL
                AND mfccDistance <=
                (
                    SELECT mfccDistance FROM similarity
                    WHERE queryTrackID = " . $queryTrackID . " AND mfccDistance > 0
                    ORDER BY mfccDistance LIMIT 9,1
                )
                -- AND artistName NOT LIKE '%" . mysql_escape_string($queryArtist) . "%'
            ORDER BY combinedDistance LIMIT 1
        )
        UNION
        (
            -- ****************************************
            -- best 3 based on tags and mfcc's    *
            -- USE TWO WITH TAGS                  *
            -- ****************************************
            SELECT trackID, trackName, artistName, fileName, 100 AS whichSystem, combinedDistance
                AS distance FROM music_track
            INNER JOIN similarity ON trackID = comparedTrackID
            WHERE queryTrackID = " . $queryTrackID . " AND combinedDistance >= 0 AND tagDistance
                    IS NOT NULL
                AND tagDistance <=
                (
                    SELECT tagDistance FROM similarity
                    WHERE queryTrackID = " . $queryTrackID . " AND tagDistance > 0
                    ORDER BY tagDistance LIMIT 9,1
                )
                -- AND artistName NOT LIKE '%" . mysql_escape_string($queryArtist) . "%'
            ORDER BY combinedDistance LIMIT 2
        )
        UNION
        (
            -- ****************************************
            -- three BAD recommendations !        *
            -- ****************************************
            SELECT trackID, trackName, artistName, fileName, 1000 AS whichSystem, combinedDistance
                AS distance FROM music_track
            INNER JOIN similarity ON trackID = comparedTrackID
            WHERE queryTrackID = " . $queryTrackID . "
                -- AND artistName NOT LIKE '%" . mysql_escape_string($queryArtist) . "%'
            ORDER BY combinedDistance LIMIT 12000, 3
        )
    )
```

```
        AS bigUnion
        GROUP BY trackID
            -- , trackName, artistName, fileName, distance
        ORDER BY artistName;";

    $recommendations = mysql_query
    (
        $recommendations_SQL_statement
    )
    or die('ERROR SELECTING TABLE<br><br>' . $recommendations_SQL_statement . '<br><br>' .
mysql_error())

?>
```

Insert Ratings – PHP Script from Web Application:

```php
<?php

   $ratingMessage = "";
   $errorMessage = "";

   // validate data
   if( $_SERVER["REQUEST_METHOD"] == "POST" )
   {

      //print_r( $_POST );

      unset( $tagRatings );
      unset( $mfccRatings );
      unset( $hybridRatings );
      unset( $badSongRatings );

      // if user does not select a radio button value for EVERY row/recommendation
      //      (default value for radio button is zero... which is hidden and not selectable)
      if( in_array( "0", $_POST ) )
      {
         $errorMessage = '<font color="red">* Please rate every song. *</font>';
         $ratingMessage = '';
      }
      // otherwise, carry on with data formatting and insert into database !
      else
      {

         // assign values from form input
         foreach( $_POST as $index=>$value )
         {
            // remove last char
            // (last char/letter was used to create set of 5 radio buttons per recommended song)
            $key = substr( $index, 0, -1 );
            // add feedback value to appropriate array based on key/index
            switch( $key )
            {
               // 1, 10, 100, or combination of them (SUM)
               case 1:
                  $tagRatings[] = $value;
                  break;
               case 10:
                  $mfccRatings[] = $value;
                  break;
               case 100:
                  $hybridRatings[] = $value;
                  break;
               case 11:
                  $tagRatings[] = $value;
                  $mfccRatings[] = $value;
                  break;
               case 101:
                  $tagRatings[] = $value;
                  $hybridRatings[] = $value;
```

```
            break;
        case 110:
            $mfccRatings[] = $value;
            $hybridRatings[] = $value;
            break;
        case 111:
            $tagRatings[] = $value;
            $mfccRatings[] = $value;
            $hybridRatings[] = $value;
            break;
        // default is rating submitted for bad recommendation (1000)
        case 1000:
            $badSongRatings[] = $value;
            break;
    } // end switch
} // end foreach loop */

// add NULL ratings if there are no recommended songs based on specific system
// (this will not happen the way this system is set up!)
if( empty( $tagRatings ) )
{
    for( $i=0; $i<3; $i++)
        $tagRatings[] = "NULL";
} // end if

if( empty( $mfccRatings ) )
{
    for( $i=0; $i<3; $i++)
        $mfccRatings[] = "NULL";
} // end if

if( empty( $hybridRatings ) )
{
    for( $i=0; $i<3; $i++)
        $hybridRatings[] = "NULL";
} // end if

if( empty( $badSongRatings ) )
{
    for( $i=0; $i<3; $i++)
        $badSongRatings[] = "NULL";
} // end if

// prepare anonymous data for insertion into database
$insertQuery =
    "INSERT INTO session
    (
        queryTrackID,
            tagRating1, tagRating2, tagRating3,
            mfccRating1, mfccRating2, mfccRating3,
            hybridRating1, hybridRating2, hybridRating3,
            badSongRating1, badSongRating2, badSongRating3
    )
    VALUES
    (" .
    $queryTrackID . ", " .
```

```
                $tagRatings[0] . ", " .  $tagRatings[1] . ", " . $tagRatings[2] . ", " .
                $mfccRatings[0] . ", " .  $mfccRatings[1] . ", " . $mfccRatings[2] . ", " .
                $hybridRatings[0] . ", " .  $hybridRatings[1] . ", " . $hybridRatings[2] . ", " .
                $badSongRatings[0] . ", " . $badSongRatings[1] . ", " . $badSongRatings[2] .
            ");";

        $feedbackInserted = mysql_query( $insertQuery )
        or die('ERROR:<br>' . $insertQuery . '<br>' . mysql_error());

        // update message on page to reflect form data successfully being submitted to database
        $ratingMessage =  "<script type='text/javascript' charset='utf-8'> alert('Thank you for your
feedback.  Anonymously, it will help to get a feel for the accuracy of the three parts of this system.');
window.location.href='main.php'; </script>";
        $errorMessage = "";

    } // end big if else

  } // end huge if

?>
```

Top Tag Preload – PHP Script:

```php
<?php

   // keep track of time php script takes to complete
   $timeStart = microtime(true);

   // extend max execution time
   $MAX_EXEC_TIME = 0; // unlimited number of hours?
   ini_set('max_execution_time', $MAX_EXEC_TIME * 60);

   // Total number of top tags we want to store and use
   $MAX_TOP_TAGS = 10;

   // let's keep track of how many songs return zero top tags in Last.fm
   $NumTracksWithNoTags = 0;

   // API key is constant
   $API_KEY='**********************************';

   // connect to music recommendation system database

   // retrieve data from music_track table
   $data = mysql_query('SELECT * FROM music_track')
   or die('ERROR SELECTING TABLE' . mysql_error());

   while($music_row = mysql_fetch_array( $data ))
   {
      // preserve original from database
      $trackID = $music_row['trackID'];

      // replace spaces with '+' for Last.fm URL to call API
      $trackName = str_replace(" ", "+", $music_row['trackName']);
      // remove special characters like # and %
      $trackName = preg_replace('/[^A-Za-z0-9\.\-\+\&\!\?\']/', '', $trackName);

      $artistName = str_replace(" ", "+", $music_row['artistName']);
      $artistName = preg_replace('/[^A-Za-z0-9\.\-\+\&\!\?\']/', '', $artistName);

      //print '-----------------------------------------------<br>';
      //print'<br>Track: '.$trackName. ', Artist: '.$artistName.'<br>';

      // HTTP request using Last.fm API
      // remove special characters like # and $
      $lastfmURL = 'http://ws.audioscrobbler.com/2.0/?method=track.getTopTags&track=' .
            mysql_real_escape_string($trackName) . '&artist=' .
            mysql_real_escape_string($artistName).'&api_key='.$API_KEY.'&format=json';

      //print $lastfmURL."<br>";

      $content = file_get_contents($lastfmURL);
      $JSONtoptags = json_decode($content, TRUE );

      // if no tags returned: check two reasons why that is
      if ( empty( $JSONtoptags['toptags']['tag'] ) )
```

```
    {
      // if error because of something not spelled correctly or song doesn't exist in system
      if ( !empty( $JSONtoptags['error'] ) )
      {
        //print "ERROR with " . $trackName . ".<br>" . $JSONtoptags['message'] . "<br>";
        $NumTracksWithNoTags ++;
      }
      // if error because there are no tags for this song
      else
      {
        //print 'NO TAGS FOR<br>'.$trackName. '<br>'.$artistName .
          '<br>-----------------------------------------------<br>';
        $NumTracksWithNoTags ++;
      } // end  inner if else
    }

    // check if only one tag set returned; process data accordingly
    elseif ( ! isset( $JSONtoptags['toptags']['tag'][1] ) )
    {
      $tagData = $JSONtoptags['toptags']['tag'];
      //print "tag name:<br>".$tagData['name']."<br>tag count:<br>" .
          $tagData['count'].'<br>``````````````````````````<br>';

      // insert each pair into database
      $insertQueryString = "INSERT INTO top_tag (tagName, tagCount, trackID)
        VALUES (\"".mysql_real_escape_string($tagData['name'])."\", ".$tagData['count'].",
".$trackID.");";

    $insertRow = mysql_query($insertQueryString, $conn)
      or die('<br>ERROR: <br>' . $insertQueryString . '<br>' . mysql_error()."<br>");
    }

    // otherwise more than one tag returned; process data accordingly
    else
    {

      // only grab five top tag pairs
      $tagCount = 1;

      // retrieve each top tag and tag count
      foreach($JSONtoptags['toptags']['tag'] as $tagData)
      {
        // only specified number of top  toptags
        if( $tagCount > $MAX_TOP_TAGS)
          break;

        //print "tag name:<br>".$tagData['name']."<br>tag count:<br>" .
            $tagData['count'].'<br>``````````````````````````<br>';

        // insert each pair into database
        $insertQueryString = "INSERT INTO top_tag (tagName, tagCount, trackID)
          VALUES (\"".mysql_real_escape_string($tagData['name'])."\", ".$tagData['count'] .
              ", ".$trackID.");";

        $insertRow = mysql_query($insertQueryString, $conn)
          or die('<br>ERROR: <br>' . $insertQueryString . '<br>' . mysql_error()."<br>");
```

```
            $tagCount ++;
        } // end foreach

        //print 'Top Tags entered into database.<br>';

    } // end outer if else if else

} // end while

mysql_close($conn);

// keeping track of time script takes to complete
$timeEnd = microtime(true);

// calculate execution time
$executionTime = gmdate( "i:s", ($timeEnd - $timeStart) );

print '-----------------------------------------------<br><b>Program Execution Time = </b>' .
      $executionTime.' minutes<br>';

?>
```

Gather MFCC Averages – Sonic Annotator Batch Script:

```
start c:\sonic-annotator-1.0-win32\sonic-annotator.exe -d vamp:qm-vamp-plugins:qm-mfcc:coefficients -S
mean --summary-only -w csv --csv-basedir "I:\datrumpet\Desktop\Drop box\Dropbox\__THESIS\_Thesis
Project\DB\Mp3 MFCC DATA to import\averages A" "I:/mp3 dataset/Spinnaker Radio A"  -r
```

Gather MFCC Variances – Sonic Annotator Batch Script:

```
start c:\sonic-annotator-1.0-win32\sonic-annotator.exe -d vamp:qm-vamp-plugins:qm-mfcc:coefficients -S
variance --summary-only -w csv --csv-basedir "I:\datrumpet\Desktop\Drop
box\Dropbox\__THESIS\_Thesis Project\DB\Mp3 MFCC DATA to import\variances A" "I:/mp3
dataset/Spinnaker Radio A" -r
```

MFCC Preload – PHP Script:

```
<?php

   // extend max execution time
   $MAX_EXEC_TIME = 0; // unlimited number of hours?
   ini_set('max_execution_time', $MAX_EXEC_TIME * 60);

   // connect to music recommendation system database

   // clear all data from mfcc_data table in order to repopulate
   $clearData = mysql_query('DELETE FROM mfcc_data;');
   //print 'MFCC table data is now cleared...<br>';

   // read in a folder of CSV files exported from Sonic Annotator !
   // array of Strings (file path names)
   $globAverage = glob("I:/datrumpet/Desktop/Drop box/Dropbox/__THESIS/_Thesis Project/DB/Mp3
MFCC DATA to import/averages/*.csv");

   $globVariance = glob("I:/datrumpet/Desktop/Drop box/Dropbox/__THESIS/_Thesis Project/DB/Mp3
MFCC DATA to import/variances/*.csv");

   $index = 0;

   // loop for each csv file in averages directory !
   foreach($globAverage as $filePathAverage)
   {
         // retrieve first file name in variances folder, too !
         $filePathVariance = $globVariance[$index];

         $index ++;

         $fileAverage = fopen($filePathAverage, 'r');
         $fileVariance = fopen($filePathVariance, 'r');

         // loop while end-of-file not yet reached
         while ( ( $csvLineAverage = fgetcsv($fileAverage) ) !== FALSE )
         {
            $csvLineVariance = fgetcsv($fileVariance);

            //var_dump ($csvLineVariance); exit();

            // retrieve file name which matches up with music track file name
            $fileName = basename($filePathAverage);

            // get rid of extra auto-generated Sonic Annotator name junk (46 characters including .csv)
            $SQLFileName = str_replace( "_vamp_qm-vamp-plugins_qm-mfcc_coefficients.csv",
                                        "", $fileName );
            //$SQLFileName = substr( $fileName, 0, -46 );

            //print $SQLFileName."<br>";

            // insert row into MySQL DB
            $trackIDQueryString = "SELECT trackID FROM music_track WHERE fileName = \"" .
                                  $SQLFileName."\";";
```

```
        $trackIDQuery = mysql_query( $trackIDQueryString );

        $trackID = mysql_fetch_array ( $trackIDQuery )[0];

        if( $trackID == NULL)
        {
           print $SQLFileName .
                 " does not match up with a music_track.fileName value in the database !<br>";
           print "<br>Please fix this csv file name to match the song file name and try again.<br>";
           exit();
        }

        $insertQueryString =
              "INSERT INTO mfcc_data (frameNumber, varianceValue, averageValue, trackID)
                    VALUES (1, ".$csvLineVariance[3].", ".$csvLineAverage[3].", ".$trackID."),
                           (2, ".$csvLineVariance[4].", ".$csvLineAverage[4].", ".$trackID."),
                           (3, ".$csvLineVariance[5].", ".$csvLineAverage[5].", ".$trackID."),
                           (4, ".$csvLineVariance[6].", ".$csvLineAverage[6].", ".$trackID."),
                           (5, ".$csvLineVariance[7].", ".$csvLineAverage[7].", ".$trackID."),
                           (6, ".$csvLineVariance[8].", ".$csvLineAverage[8].", ".$trackID."),
                           (7, ".$csvLineVariance[9].", ".$csvLineAverage[9].", ".$trackID."),
                           (8, ".$csvLineVariance[10].", ".$csvLineAverage[10].", ".$trackID."),
                           (9, ".$csvLineVariance[11].", ".$csvLineAverage[11].", ".$trackID."),
                           (10, ".$csvLineVariance[12].", ".$csvLineAverage[12].", ".$trackID."),
                           (11, ".$csvLineVariance[13].", ".$csvLineAverage[13].", ".$trackID."),
                           (12, ".$csvLineVariance[14].", ".$csvLineAverage[14].", ".$trackID."),
                           (13, ".$csvLineVariance[15].", ".$csvLineAverage[15].", ".$trackID."),
                           (14, ".$csvLineVariance[16].", ".$csvLineAverage[16].", ".$trackID."),
                           (15, ".$csvLineVariance[17].", ".$csvLineAverage[17].", ".$trackID."),
                           (16, ".$csvLineVariance[18].", ".$csvLineAverage[18].", ".$trackID."),
                           (17, ".$csvLineVariance[19].", ".$csvLineAverage[19].", ".$trackID."),
                           (18, ".$csvLineVariance[20].", ".$csvLineAverage[20].", ".$trackID."),
                           (19, ".$csvLineVariance[21].", ".$csvLineAverage[21].", ".$trackID."),
                           (20, ".$csvLineVariance[22].", ".$csvLineAverage[22].", ".$trackID.");";

        // print "insert query:  ".$insertQueryString."<br><br>";

        $insertRow = mysql_query($insertQueryString, $conn)
             or die(var_dump(mysql_fetch_array ( $trackIDQuery )) . '<br><br>trackID = ' .
             $trackID . '<br>' . $trackIDQueryString . '<br>' . mysql_error()."<br>");

      } // end while

  } // end foreach

  mysql_close($conn);
?>
```

Tag Similarity Preload 1 of 4 – PHP Script:

```php
<?php

   $csvFile = "I:\datrumpet\Desktop\Drop box\Dropbox\__THESIS\_Thesis Project\DB\Similarity
Tags\Similarity Tags 1b of 4b.csv";

   // extend max execution time
   $MAX_EXEC_TIME = 0; // unlimited number of hours?
   ini_set('max_execution_time', $MAX_EXEC_TIME * 60);

   // connect to music recommendation system database

   // * select only those music tracks that have one or more entries in top_tag table ! *
   $musicResult = mysql_query('SELECT DISTINCT trackID FROM top_tag ORDER BY trackID;')
      or die('ERROR SELECTING TABLE' . mysql_error());

   // store primary keys for all music tracks into an array
   while( $thisTrackID = mysql_fetch_array( $musicResult ))
   {
      $trackIDs[] = $thisTrackID[0];
   } // end while loop

   print "Total tracks with one or more top tags: " . sizeof( $trackIDs ) . "<br><br>";

   // instantiate arrays for use in outer and inner loops below
   $queryTrackTags = array();
   $comparedTrackTags = array();
   $allTrackTags = array();
   $theseRows = array();

   // open csv file to put data into
   $handle = fopen( $csvFile, 'w' );

   $queryTrackCounter = 1;

   // iterate through array of trackID's (songs with at least one top tag)
   foreach( $trackIDs as $queryTrackID )
   {

      if( $queryTrackCounter >770 && $queryTrackCounter <= 1938 )
      {

         // reset / clear out array for reuse in for loop !
         unset( $queryTrackTags );
         unset( $theseRows );

         $queryTagsResult = mysql_query("SELECT tagName, tagCount FROM top_tag
                                          WHERE trackID = " . $queryTrackID .
                                          " ORDER BY tagCount DESC;")
                                       or die('ERROR SELECTING TABLE' . mysql_error());

         // store all tag names and tag counts for query song into an array
         while( $thisQueryTag = mysql_fetch_array( $queryTagsResult ))
         {
```

```
            $queryTrackTags['name'][] = $thisQueryTag[0];
            $queryTrackTags['count'][] = $thisQueryTag[1];
        } // end while

        // store sum of query song tag counts for normalization
        $queryTagCountSum = array_sum( $queryTrackTags['count'] );

        // iterate through array of query song tag counts and normalize each one
        //      (by dividing by total tag counts)
        foreach( $queryTrackTags['count'] as $queryTagCount)
        {
            $queryTrackTags['normalized count'][] = $queryTagCount / $queryTagCountSum;
        } // end foreach

        //print "-----------------------------------------------------------------------------------------------------<br>";
        //print_r( $queryTags );
        //print "<br><br>";

        $updateQueryString = "";
        $firstTimeThroughLoop = TRUE;

        foreach( $trackIDs as $comparedTrackID )
        {

            // proceed with similarity calculations if the two songs are not the same song !
            if ( $queryTrackID != $comparedTrackID )
            {

                // reset / clear out arrays for reuse in for loop !
                unset( $comparedTrackTags );
                unset( $allTrackTags );

                $comparedTagsResult = mysql_query("SELECT tagName, tagCount FROM top_tag
                                                    WHERE trackID = " . $comparedTrackID .
                                                    " ORDER BY tagCount DESC;")
                                                or die('ERROR SELECTING TABLE' . mysql_error());

                // store all tag names and tag counts for compared song into an array
                while( $thisComparedTag = mysql_fetch_array( $comparedTagsResult ))
                {
                    $comparedTrackTags['name'][] = $thisComparedTag[0];
                    $comparedTrackTags['count'][] = $thisComparedTag[1];
                } // end while

                // store sum of compared song tag counts for normalization
                $comparedTagCountSum = array_sum( $comparedTrackTags['count'] );

                // iterate through array of compared song tag counts and normalize each one
                //      (by dividing by total tag counts)
                foreach( $comparedTrackTags['count'] as $comparedTagCount)
                {
                    $comparedTrackTags['normalized count'][] = $comparedTagCount /
$comparedTagCountSum;
                } // end foreach
```

```
            //print "-----------------------------------------------------------------------------------------------------
<br>";
            //print_r( $comparedTrackTags );
            //print "<br><br>";

            //print "-------------------------------------<br>";

            // consolidate tag names (and corresponding tag counts) from query track and compared track
            //      calculate and store value of each term:  (count1 + count2) * (count1 - count2) ^ 2
            foreach( $queryTrackTags['name'] as $queryTagName )
            {

               $queryIndex = array_search( $queryTagName, $queryTrackTags['name'] );
               $queryCount = $queryTrackTags['normalized count'][ $queryIndex ];

               $allTrackTags['name'][] = $queryTagName;

               // if tag name is present for both tracks
               if ( in_array( $queryTagName, $comparedTrackTags['name'] ) )
               {

                  $comparedIndex = array_search( $queryTagName, $comparedTrackTags['name'] );
                  $comparedCount = $comparedTrackTags['normalized count'][ $comparedIndex ];

                  $allTrackTags['term'][]
                     =   pow( $queryCount - $comparedCount, 2 ) * ( $queryCount + $comparedCount );

                  //print "*****\t\t\t" .
                   ( ( $queryCount - $comparedCount) ^ 2 ) * ( $queryCount + $comparedCount ) . "\t\t\t";

               }
               // otherwise tag is only present in query track
               else
               {

                  $allTrackTags['term'][]
                     = pow( $queryCount - 0, 2 ) * ( $queryCount + 0 );

                  //print "\t\t\t" . ( ( $queryCount - 0 ) ^ 2 ) * ( $queryCount + 0 ) . "\t\t\t";

               } // end if else

               //print $queryTagName . "<br>";

            } // end inner inner foreach

            //print "--------------<br>";

            foreach( $comparedTrackTags['name'] as $comparedTagName )
            {

               // if tag name is only present for compared track
               //    (already covered ones that are and ones unique to query track)
               if (  ! in_array( $comparedTagName, $queryTrackTags['name'] ) )
               {
```

```
                $comparedIndex = array_search( $comparedTagName, $comparedTrackTags['name'] );
                $comparedCount = $comparedTrackTags['normalized count'][ $comparedIndex ];

                $allTrackTags['name'][] = $comparedTagName;

                $allTrackTags['term'][]
                   = pow( $comparedCount - 0, 2 ) * ( $comparedCount + 0 );

                //print "\t\t\t" . ( ( $comparedCount - 0 ) ^ 2 ) * ( $comparedCount + 0 ) . "\t\t\t";

                } // end if

                //print $comparedTagName . "<br>";

             } // end inner inner foreach

             // Calculate the square root of the sum of all "terms" in allTrackTags array
             //    (Eucdistance) between the two tracks ! =)
             $euclideanDistance = sqrt( array_sum( $allTrackTags['term'] ) );

             //print "Sum of Terms = " . array_sum( $allTrackTags['term'] ) . "<br>";
             //print "Similarity between " . $queryTrackID . " and " . $comparedTrackID . " = " .
$tagDistance . "<br><br>";

          }
          else
          {

             // two tracks are identical, so distance will be ZERO
             $euclideanDistance = 0;

          } // end if else

          // retrieve simID from similarity table with the current queryTrackID and comparedTrackID
          $simIDQueryString = "SELECT simID FROM similarity WHERE queryTrackID =" .
                              $queryTrackID . " AND comparedTrackID = ".$comparedTrackID.";";
          $simIDQuery = mysql_query( $simIDQueryString );
          $simID = mysql_fetch_array ( $simIDQuery )[0];

          if( $simID == NULL)
          {
             print $queryTrackID . " and " . $comparedTrackID .
                   " do not match up with a set of queryTrackID and comparedTrackID values in the
similarity table !<br>EXITING NOW.<br>";
             exit();
          }

          // add value pair to CSV file to ultimately insert into temp table and update via MySQL table join
! =)
          $theseRows[$simID][] = $simID;
          $theseRows[$simID][] = $euclideanDistance;

       } // end inner foreach

       // insert 15k rows into CSV file
       foreach( $theseRows as $thisRow )
```

```
        {
            fputcsv( $handle, $thisRow );
        }

    print "UPDATED 15312 rows with querytrackID " . $queryTrackID . "<br>";

    } // end big if

    $queryTrackCounter++;

  } // end outer foreach

  fclose( $handle );
  mysql_close($conn);

?>
```

MFCC Similarity Preload 1 of 4 – PHP Script:

```php
<?php

   $csvFile = "I:\datrumpet\Desktop\Drop box\Dropbox\__THESIS\_Thesis Project\DB\Similarity MFCC\Similarity MFCC 1 of 4.csv";

   $covariancePath = "I:\datrumpet\Desktop\Drop box\Dropbox\__THESIS\_Thesis Project\DB\Mp3 Covariance Matrix\MFCC INVERSE Covariance Matrix.csv";

   // extend max execution time
   $MAX_EXEC_TIME = 0; // unlimited number of hours?
   ini_set('max_execution_time', $MAX_EXEC_TIME * 60);

   // connect to music recommendation system database

   // * select all music tracks that have data in mfcc_data table ! *
   $musicResult = mysql_query('SELECT DISTINCT trackID FROM mfcc_data ORDER BY trackID;')
      or die('ERROR SELECTING TABLE' . mysql_error());

   // store primary keys for all music tracks into an array
   while( $thisTrackID = mysql_fetch_array( $musicResult ))
   {
      $trackIDs[] = $thisTrackID[0];
   } // end while loop

   print "Total tracks with MFCC data available: " . sizeof( $trackIDs ) . "<br><br>";

   // instantiate int for auto-increment
   $autoNum = 1;

   // instantiate arrays for use in outer and inner loops below
   $queryTrackMFCCs = array();
   $queryMatrix = array();
   $comparedTrackMFCCs = array();
   $comparedMatrix = array();
   $covarianceMatrix = array();
   $differenceMatrix = array();
   $differenceMatrixTransposed = array();
   $productMatrix = array();
   $theseRows = array();

   // read covariance matrix (constant for entire dataset) into two dimensional array
   $covarianceFile = fopen( $covariancePath, "r" );
   // loop while end-of-file not yet reached
   while( ! feof( $covarianceFile ) )
   {
      $covarianceMatrix[] = fgetcsv( $covarianceFile );
   }
   fclose( $covarianceFile );

   // open csv file to put data into
   $handle = fopen( $csvFile, 'w' );

   //print_r( $covarianceMatrix );
```

```
$queryTrackCounter = 1;

// iterate through array of trackID's)
foreach( $trackIDs as $queryTrackID )
{

   // only process first quarter of dataset
   if( $queryTrackCounter <= 3828 )
   {

      // reset / clear out array for reuse in for loop !
      unset( $queryTrackMFCCs );
      unset( $queryMatrix );
      unset( $theseRows );

      $queryTagsResult = mysql_query("SELECT averageValue, varianceValue, trackID FROM
mfcc_data
                                       WHERE trackID = " . $queryTrackID .
                                       " ORDER BY frameNumber;")
                                    or die('ERROR SELECTING TABLE' . mysql_error());

      // store averages and variances for query song into an array
      while( $thisQueryMFCC = mysql_fetch_array( $queryTagsResult ))
      {
         $queryTrackMFCCs['average'][] = $thisQueryMFCC[0];
         $queryTrackMFCCs['variance'][] = $thisQueryMFCC[1];
      } // end while

      //$queryTrackMFCCs['trackID'][] = $queryTrackID;
      $queryMatrix =
         array_merge( $queryTrackMFCCs['average'], $queryTrackMFCCs['variance'] );

      //print "-------------------------------------------------------------------------------------------<br>";
      //print_r( $queryMatrix );
      //print "<br>";

      $firstTimeThroughLoop = TRUE;

      // iterate through array of trackID's to compare to each trackID
      foreach( $trackIDs as $comparedTrackID )
      {

         // reset / clear out array for reuse in for loop !
         unset( $comparedTrackMFCCs );
         unset( $comparedMatrix );
         unset( $differenceMatrix );
         unset( $differenceMatrixTransposed );
         unset( $productMatrix );

         $comparedTagsResult = mysql_query("SELECT averageValue, varianceValue FROM
mfcc_data
                                       WHERE trackID = " . $comparedTrackID .
                                       " ORDER BY frameNumber;")
                                    or die('ERROR SELECTING TABLE' . mysql_error());
```

```
// store all tag names and tag counts for compared song into an array
while( $thisComparedMFCC = mysql_fetch_array( $comparedTagsResult ))
{
   $comparedTrackMFCCs['average'][] = $thisComparedMFCC[0];
   $comparedTrackMFCCs['variance'][] = $thisComparedMFCC[1];
} // end while

//$comparedTrackMFCCs['trackID'][] = $comparedTrackID;
$comparedMatrix =
   array_merge( $comparedTrackMFCCs['average'], $comparedTrackMFCCs['variance'] );

//print "-----------------------------------<br>";
//print_r( $comparedMatrix );
//print "<br><br>";

//*****************************************//
// proceed with similarity calculations //
//*****************************************//

// calculate the difference and transposed difference between two matrices (arrays)
for( $i = 0; $i <= 39; $i++ )
{
   $differenceMatrix[0][$i] = $queryMatrix[$i] - $comparedMatrix[$i];
   $differenceMatrixTransposed[$i] = $queryMatrix[$i] - $comparedMatrix[$i];
} // end for loop

//print "Difference of the two: <br>";
//print_r( $differenceMatrixTransposed );
//print "<br>";

// calculate the product between the transposed difference matrix and the covariance matrix
//       iterate through each of 40 columns of covariance matrix
for( $i = 0; $i <= 39; $i++)
{
   $productMatrix[$i] = 0;

   // within current column of covariance matrix, iterate through 40 values
   //      and iterate through 40 values of difference matrix
   for( $j = 0; $j <= 39; $j++ )
   {
      // sum of products
      $productMatrix[$i] += $differenceMatrixTransposed[$j] * $covarianceMatrix[$i][$j];
   } // end inner for loop

   //print "= " . $productMatrix[$i] . "<br>-------------------------<br>";
} // end outer for loop

//print "<br>Product matrix:<br>";
//print_r( $productMatrix );
//print "<br><br>";

// start the Mahalanobis Distance at zero
$mahalanobisDistance = 0;

// calculate the product between the previous resulting matrix product and the difference matrix
for( $i = 0; $i <= 39; $i++)
```

```
            {
               // sum of products
               $mahalanobisDistance += $productMatrix[$i] * $differenceMatrix[0][$i];
            } // end for loop

            //print "= " . $mahalanobisDistance . "<br>";

         // add data of all 15k comparisons to queryTrack into 2D array
         //      to ultimately insert into temp table and update via MySQL table join ! =)
         $theseRows[$autoNum][] = $autoNum++;
         $theseRows[$autoNum][] = $queryTrackID;
         $theseRows[$autoNum][] = $comparedTrackID;
         $theseRows[$autoNum][] = "NULL";
         $theseRows[$autoNum][] = $mahalanobisDistance;
         $theseRows[$autoNum][] = "NULL";

         } // end inner foreach

         // insert 15k rows into CSV file
         foreach( $theseRows as $thisRow )
         {
            fputcsv( $handle, $thisRow );
         }

      } // end big if

      $queryTrackCounter++;

   } // end outer foreach

   fclose( $handle );
   mysql_close($conn);

?>
```

Normalized Hybrid Similarity Preload 1 of 4 – PHP Script:

```php
<?php

   // connect to music recommendation system database

   $csvFile = 'I:\normalized hybrid distance CSVs\combinedDistanceNEW1a.csv';
   $logFilePHP = 'I:\normalized hybrid distance CSVs\logFilePHP1a.txt';

   // extend max execution time
   $MAX_EXEC_TIME = 0; // unlimited number of hours
   ini_set('max_execution_time', $MAX_EXEC_TIME * 60);

   // * select only those music tracks that have one or more entries in top_tag table ! *
   $musicResult = mysql_query('SELECT trackID FROM music_track
                                    WHERE trackID IN (SELECT trackID FROM top_tag)
                                    ORDER BY trackID;')
      or die('ERROR SELECTING TABLE<br>' . mysql_error());

   // store primary keys for all music tracks with top tags into an array
   while( $thisTrackID = mysql_fetch_array( $musicResult ))
   {
      $trackIDsWithTags[] = $thisTrackID[0];
   } // end while loop

   // * select all music tracks in music_track table ! *
   $musicResult2 = mysql_query('SELECT trackID FROM music_track
                                    ORDER BY trackID;')
      or die('ERROR SELECTING TABLE<br>' . mysql_error());

   // store primary keys for all music tracks with top tags into an array
   while( $thisTrackID2 = mysql_fetch_array( $musicResult2 ))
   {
      $trackIDsAll[] = $thisTrackID2[0];
   } // end while loop

   print "Total tracks with one or more top tags: " . sizeof( $trackIDsWithTags ) . "<br>";
   print "Total tracks: " . sizeof( $trackIDsAll ) . "<br>";



   // instantiate arrays for use in outer and inner loops below
   $theseRows = array();

   // open csv file to put data into
   $handle = fopen( $csvFile, 'w' );
   $handle2 = fopen( $logFilePHP, 'w' );

   // this file does roughly one quarter of 7700 query tracks
   $queryTrackCounter = 1;



   // OUTER LOOP
   // iterate through array of trackID's (songs with at least one top tag)
```

```
foreach( $trackIDsWithTags as $queryTrackID )
{

// clear out arrays
unset( $tagDistance );
unset( $tagDistanceSorted );
unset( $mfccDistance );
unset( $mfccDistanceSorted );
unset( $comparedTrackID );
unset( $simID );
unset( $thisComparedTrackDistance );
unset( $comparedTrackDistanceResults );

   if( $queryTrackCounter <= 2000 )
   //if( $queryTrackID = 65544 )
   //if( $queryTrackCounter == 1 )
   //if( TRUE )
   {

      // retrieve tagDistance and mfccDistance for each combination of query track and compared track
      $comparedTrackDistanceResults =
         mysql_query("SELECT tagDistance, mfccDistance, comparedTrackID, simID FROM similarity
                              WHERE queryTrackID = " . $queryTrackID .
                              " ORDER BY comparedTrackID;")
                                   or die('ERROR SELECTING TABLE<br>' . mysql_error());

      while( $thisComparedTrackDistance = mysql_fetch_array( $comparedTrackDistanceResults ))
      {
         $tagDistance[]          =    $thisComparedTrackDistance[0];
         $mfccDistance[]         =    $thisComparedTrackDistance[1];
         $comparedTrackID[]      =    $thisComparedTrackDistance[2];
         $simID[]                =    $thisComparedTrackDistance[3];
      } // end while

      // SORT ARRAYS ASCENDING (AFTER FILTERING OUT NULL AND ZERO)
      $tagDistanceSorted    =  $tagDistance;
      foreach( $tagDistanceSorted as $tagKey => $tagValue )
      {
         if( $tagValue == 0 || $tagValue == NULL )
            unset( $tagDistanceSorted[ $tagKey ] );
      }
      sort( $tagDistanceSorted );

      $mfccDistanceSorted =  $mfccDistance;
      foreach( $mfccDistanceSorted as $mfccKey => $mfccValue )
      {
         if( $mfccValue == 0 || $mfccValue == NULL )
            unset( $mfccDistanceSorted[ $mfccKey ] );
      }
      sort( $mfccDistanceSorted );

      // store tagDistance MIN, tagDistance MAX, mfccDistance MIN, and mfccDistance
      // MAX for normalization
      //       (MIN and MAX are only of the lowest 20 distance values !)
      $tagDistanceMAX       =  $tagDistanceSorted[19];
      $tagDistanceMIN       =  $tagDistanceSorted[0];
```

```
        $mfccDistanceMAX    =   $mfccDistanceSorted[19];
        $mfccDistanceMIN    =   $mfccDistanceSorted[0];

/*      print "tagDistance MIN = " . $tagDistanceMIN . "<br>";
        print "tagDistance MAX = " . $tagDistanceMAX . "<br>";
        print "mfccDistance MIN = " . $mfccDistanceMIN . "<br>";
        print "mfccDistance MAX = " . $mfccDistanceMAX . "<br>"; */

          if( $tagDistanceMAX == $tagDistanceMIN )
          {
            //print "divide by zero error!<br>tag distance max:  " . $tagDistanceMAX .
            //      "<br>tag distance min:  " . $tagDistanceMIN . "<br>";
            print $queryTrackID . " queryTrackID tagDistance DIVIDE BY ZERO!<br>";
            fwrite( $handle2, $queryTrackID . "     tagDistance issue\n" );
          }

          if( $mfccDistanceMAX == $mfccDistanceMIN )
          {
            //print "divide by zero error!<br>mfcc distance max:  " . $mfccDistanceMAX .
            //      "<br>mfcc distance min:  " . $mfccDistanceMIN . "<br>";
            print $queryTrackID . " queryTrackID mfccDistance DIVIDE BY ZERO!<br>";
            fwrite( $handle2, $queryTrackID . "     mfccDistance issue\n" );
          }



        // clear out arrays
        unset( $theseRows );

        // instantiate integer error code
        $combinedNormalizedDistance = -666;



        // INNER LOOP
        // inside outer loop, iterate through list of ALL tracks (about 15k of them) and compare
        foreach( $comparedTrackID as $index => $thisComparedTrackID )
        {

/*              print "current variable values:<br>tagDistance:  " . $tagDistance[ $index ] .
                "<br>mfccDistance:  " . $mfccDistance[ $index ] .
                "<br>simID:  " . $simID[ $index ] .
                "<br>----------------------------------------------------------<br>"; */



          // use the following calculation if the compared track has top tags and thus tagDistance is not
NULL
          //if ( in_array($comparedTrackID, $trackIDsWithTags )
          if( array_key_exists( $index, $tagDistance ) )
          {
            if( $tagDistance[ $index ] != NULL )
            {

              $combinedNormalizedDistance =
                (
```

```
                    (
                        ( $tagDistance[ $index ] - $tagDistanceMIN )
                        /
                        ( $tagDistanceMAX - $tagDistanceMIN )
                    )
                    +
                    (
                        ( $mfccDistance[ $index ] - $mfccDistanceMIN )
                        /
                        ( $mfccDistanceMAX - $mfccDistanceMIN )
                    )
                )
                / 2;
            }
            // otherwise do the following calculate if no top tags exist and thus tagDistance is NULL
            else
            {

                $combinedNormalizedDistance =
                    ( $mfccDistance[ $index ] - $mfccDistanceMIN )
                    /
                    ( $mfccDistanceMAX - $mfccDistanceMIN );

            } // end if else
        } // end if

        // check if combined distance not assigned a calculated value from above
        if( $combinedNormalizedDistance == -666 )
        {
            print "ERROR: queryTrackID is " . $queryTrackID . "<br>tagDistance is " .
                  $tagDistance[ $index ] . "<br><br>" .
                 "tagDistance array:<br>";
            print_r( $tagDistance );
            print "<br>";
            fwrite( $handle2, "ERROR: queryTrackID is " . $queryTrackID . "\ntagDistance is " .
                    $tagDistance[ $index ] . "\n" );
        }
        else
        {
        // add value pair to CSV file to ultimately insert into temp table and then update via MySQL table
join ! =)
        $theseRows[ $simID[ $index ] ] [] = $simID[ $index ];

        $theseRows[ $simID[ $index ] ] [] = $queryTrackID;
        $theseRows[ $simID[ $index ] ] [] = $comparedTrackID[ $index ];
        $theseRows[ $simID[ $index ] ] [] = $tagDistance[ $index ];
        $theseRows[ $simID[ $index ] ] [] = $mfccDistance[ $index ];

        $theseRows[ $simID[ $index ] ] [] = $combinedNormalizedDistance;
        } // end if else

    } // end inner foreach

    // insert rows into CSV file
    foreach( $theseRows as $thisRow )
    {
```

```
            fputcsv( $handle, $thisRow );
        }

    //print "saved " . sizeof( $trackIDsAll ) . " normalized hybrid distance values
    //        for querytrackID " . $queryTrackID . "<br>----------------------------------------------------------------
-<br>";

    } // end big if

    //if( $queryTrackCounter % 100 == 0 )
    //  print "Sets complete:  " . $queryTrackCounter . "<br>";

    $queryTrackCounter++;

  } // end outer foreach

  fclose( $handle );
  mysql_close($conn);

?>
```

VITA

Jaime C. Kaufman holds a Bachelor of Music degree from Ohio University in Music Performance and expects to receive a Master of Science degree in Computer and Information Sciences from University of North Florida in December of 2014. Dr. Ching-Hua Chuan of the University of North Florida is serving as Jaime's thesis advisor. Jaime is currently employed as a consultant/big data ingest developer for Illumination Works, LLC, in Cincinnati, Ohio. Jaime is also a freelance musician (trumpet). Prior to graduate school, Jaime served as a musician in the United States Navy for eight years.

Jaime has software development experience in Java, PHP, HTML, and CSS and database experience with MS Access, MySQL, and Hive. In his free time, Jaime enjoys racquetball, playing music, video games, movies, and spending time with his wife of thirteen years and two young daughters.