Music Recommendation and Classification Utilizing Machine Learning and Clustering Methods

Robert Griesmeyer
MUSIC RECOMMENDATION AND CLASSIFICATION UTILIZING MACHINE LEARNING AND CLUSTERING METHODS

By

ROBERT GRIESMEYER

A Thesis submitted to the Department of Computer Science in partial fulfillment of the requirements for the degree of Master of Science

Degree Awarded:
Fall Semester, 2011
Robert Griesmeyer defended this thesis on November 3rd, 2011.

The members of the supervisory committee were:

Piyush Kumar
Professor Directing Thesis

Xin Yuan
Committee Member

Zhenhai Duan
Committee Member

The Graduate School has verified and approved the above-named committee members, and certifies that the thesis has been approved in accordance with university requirements.
# TABLE OF CONTENTS

List of Tables ........................................................................................................................................ vi

List of Figures ........................................................................................................................................ vii

Abstract ............................................................................................................................................. viii

CHAPTER ONE .................................................................................................................................. 1

1.1 Motivation .................................................................................................................................. 1

1.2 Research Questions .................................................................................................................. 1

1.3 Recommendation Systems Defined .......................................................................................... 1

1.4 Introducing SmartPlayer ......................................................................................................... 2

CHAPTER TWO ................................................................................................................................ 4

2.1 Recommendation Systems ......................................................................................................... 4

2.2 Content Filtering ....................................................................................................................... 4

2.3 Collaborative Filtering ............................................................................................................. 4

2.4 Music Recommendation .......................................................................................................... 5

2.5 Support Vector Machines ........................................................................................................ 5

2.6 Boosting .................................................................................................................................... 7

2.7 Dynamic Time Warping ............................................................................................................ 8

2.7.1 DTW Algorithm .................................................................................................................... 9

CHAPTER THREE ......................................................................................................................... 11

3.1 Overview .................................................................................................................................. 11

3.1.1 Training System .................................................................................................................. 11

3.2 Design Goals ............................................................................................................................ 13

3.3 Classifier System ....................................................................................................................... 13
6.1 Broader Implications........................................................................................................ 41
6.2 Future Work..................................................................................................................... 41
6.3 Summary of Contributions............................................................................................... 42
6.4 Conclusion ....................................................................................................................... 42
REFERENCES ............................................................................................................................. 44
BIOGRAPHICAL SKETCH ........................................................................................................ 46
LIST OF TABLES

5.1 Youtube identifiers ..................................................................................................................33

5.2 Accuracy evaluations dataset one ..........................................................................................39

5.3 Accuracy evaluations dataset two ..........................................................................................40
LIST OF FIGURES

3.1 Classification and recommendation data flow.................................................................13

3.2 Video rating interface .....................................................................................................17

3.3 SmartPlayer interface ...................................................................................................18

5.1 Zero crossings time series ............................................................................................30

5.2 ZCR DTW matrix values scaled ..................................................................................34

5.3 ZCR DTW matrix values un-scaled .............................................................................35

5.4 MFCC DTW matrix values scaled ................................................................................36
ABSTRACT

This paper will discuss the development of a music classification system as a component in a music recommendation system. The front-end portion of the system is an Android Media Player application named SmartPlayer. The player is an intelligent mobile recommendation system as well as a media player that is capable to playing high quality videos. In this paper the specifics of the underlying system and the front-end components will be discussed in detail. Other methods and future aspirations will also be discussed.

The system performs automatic genre classification with one feature-vector per audio file. The backbone of the system is utilizing the existing SmartPlayer SQL database containing some 100,000 YouTube music videos in mp4 format. The system utilizes multiple components of the Marsyas (Music Analysis, Retrieval and Synthesis for Audio Signals) open source project, as well as STANN (Simple Thread-safe Approximate Nearest Neighbors). A different method that will be introduced is the K-nearest neighbor method to cluster and compare existing videos in the database. The dynamic time warping method will be used to compare different time series data derived from MFCCs. This method is generally used in comparing two time series data plots but does so with respect to relative aspects of the data.
CHAPTER ONE

INTRODUCTION

1.1 Motivation

In the light of recent revelations in recommendation systems all over world the reasons to
dive into developing more sophisticated recommenders become apparent. The most sought after
recommendation systems lie in the back end of many well-known commercial products. Examples of those products are: Netflix [1], Pandora [2] and last.fm [3]. Although these systems
are polished and provide accurate predictions, this paper will illustrate alternative methods to
solving problems in recommending media.

The obstacles overcome in finding solutions towards better recommendations are
onerous. But, developing efficient ways in which to categorize and recommended media can be
very rewarding. Some of the most successful services on the web today utilize these heuristics.

1.2 Research Questions

While this thesis explores multiple subjects and disciplines, its aim is to answer the
following research questions:

Can users’ ratings be predicted effectively while only utilizing audio content from
samples?

Is it possible to merge multiple weak and strong classifiers together to obtain more
accurate results?

Will the classifications of the audio and video samples contrast from each other in a
manner that is similar to that of human taste?

Is a k-nearest approach the most efficient and accurate way to discern differences
between samples?

1.3 Recommendation Systems Defined

This section will discuss, on a high level, the different types of recommendation schemes
used today. They are partitioned into different categories that are stated below. The most
common type of recommendation systems utilize Content Filtering and Collaborative Filtering.
There are advantages and disadvantages to each of these two approaches. The advantage in
using a Content Filtering approach is that it is possible to start recommendations without a gratuitous amount of preprocessed data from users or from computation created using external factors. The recommendation samples have associated tags or metadata that describes the nature of the samples and recommendations are made by the use of this metadata. The content filtering approach doesn’t utilize the relationships between multiple users or groups of users.

The collaborative method takes advantage of information gathered by other users. Specifically, what is gathered are user ratings. Generally, these ratings are put into graphs and matrices to compare similarity between different users. Different computations are performed and a resulting matrix is formed with recommendations filled in.

Essentially, the purpose of this paper is to introduce new methods of recommending and classifying different samples in the database. These different methods include using support vector machines, k-nearest neighbor approaches, and dynamic time warping to reach our goal in recommending relevant samples to users of the system.

1.4 Introducing SmartPlayer

The Smart Media Player is an intelligent mobile media player that utilizes an intelligent recommendation system. We derive information from samples by analyzing digital audio signals and then we classify them using an SVM classification method as well as a k-nearest approach to come up with accurate results. With accepting ratings from users we are able to customize media sample libraries while only relying on the synthesis of audio signals.

The system utilizes a recommendation system backend which provides the logic for recommending videos. These recommendations are pushed to the client front-end applications per each recommendation cycle. After users have rated their current cycle of videos, the newest ratings are sent to the server. These ratings are then used to model a certain users taste which means that the classifiers used are updated to reflect that users ratings. In this paper we will discuss the different components are part of the SmartPlayer system.

The SmartPlayer system supports many different operating systems including Windows 7, Windows Vista, Windows XP, and Linux. There is also a player for Android phones. A major part of the consumer market is covered by phone applications. By releasing SmartPlayer through the Android market place the number of users could grow exponentially. This could potentially
grow our user base which would open up the doors for different and more sophisticated algorithms for recommendations and classification.
CHAPTER TWO

BACKGROUND AND EXISTING APPROACHES

2.1 Recommendation Systems

As growth in web technologies and social media continues, more and more data is being created each day. A significant portion of this data is highly reflective of individual users themselves including their preferences and behaviors. With such an immense volume of subjective data, entirely new computer paradigms have evolved to translate data into useful information. Given the behaviors, habits, and preferences of a user, the recommendation system will ideally provide additional targeted data similar to that of the observed. Current systems can achieve RMSE accuracy up to 0.8616 with movie samples [4].

2.2 Content Filtering

Several methods have been developed to facilitate selection in recommendation systems. One of the most widely used methods is Content Filtering [5], [6]. In this model, the system attempts to provide selections similar to those provided by a given user. More specifically, the system will identify the common characteristics of the user preferences of audio samples. With this data, the system will then identify other samples possessing similar qualities to the samples preferred by the user, and provide these as recommendations. Several variations of this method are used by other recommendation systems.

2.3 Collaborative Filtering

Another recommendation system model is known as Collaborative Filtering explained in [7], [8] This method utilizes preferences and behaviors of other users to come up with recommendations. The general operation of these systems is to pair users with similar tastes together into groups. Recommendations are then made based on the collective preferences of the users within each group. Thus if user x and user y are both members of the same preference group, and user x likes a particular sample, then there should be a high probability that user y also like this sample. Obviously, this method requires a significantly large initial dataset to provide relevant predictions.
2.4 Music Recommendation

Music recommendation is a simple extension of the recommendation system paradigm using similar observation and prediction algorithms. The goal of such systems is to effectively predict the approval of songs unknown to the user. There are several other currently utilized systems that perform similar audio classification and recommendation including Pandora, the iTunes Genius, EchoNest, and TheFilter. A limitation of many existing systems is the inability to predict preferences on samples for which analysis has not been performed or community preference has not yet been determined. Mainly, recommendations on a large scale system are handled by a collaborative filtering system which recommends samples according to other users’ ratings.

2.5 Support Vector Machines

A support vector machine (SVM) is a concept that uses unsupervised learning to analyze data in order to create a classification rule. This is used to create a clustering of data and to place data into their respective classes. Support vector machines are ubiquitous in classification and regression analysis. Given a training set, the support vector machine creates a training model which represents to input data given in the training process. The testing data is compared with the model when coming up with predictions. The model can determine which class a particular sample belongs in according to independent variables.

More formally, a SVM constructs a plane where it separates points on the plane according to the independent variables in the vector. This can be used for classification. Given training data in the form \( \{x_1, \ldots, x_n\} \) such that the vectors are in a subset of real numbers, namely X. We are also given their respective labels \( y_i \in \{-1, 1\} \). The purpose is to choose a hyperplane that separates points according to the label as efficiently as possible. There are many hyperplanes that are able to classify the data according to the feature vectors. A good choice of hyperplane is one that separates points one class from the other by the largest margin of distance. If this hyperplane exists then it is called the maximum-margin hyperplane. All of the vectors containing the labels -1 will be on one side of the hyperplane and the other +1 labeled instances
will be split to reside on the other side. The training instances that lie closest to the hyperplane are called support vectors. These points that are the support vectors lie on the margin. [9]

The SVM projects the training data in space $X$ into higher dimensions in order to separate the training vectors. With trying to project the data, the data is projected into the higher dimensional space $F$ by using a kernel operator $K$. In general you can consider the set of classifiers as:

$$f(x) = \left( \sum_{i=1}^{n} \alpha_i k(x_i, x) \right).$$ [9]

By using $K$ the SVM projects the training data into a different feature space in order to separate the training data with contrasting labels. The SVM computes the $\alpha_i$s that correspond to the different hyperplanes in $F$, the newer, sometimes higher dimension feature space. The SVM can create different complex boundaries in the training data in order to separate the training vectors. [10]

In the case that the data points are linearly separable we need to use an equation of the form:

$$f(x) = w \cdot x + b = 0$$ [10]

which separates the data. What you would need to do is find $w$ which is an orthogonal vector to the hyperplane and $b$ that is the offset. This would make $f(x)$ the decision rule. This rule is made better by picking a hyperplace that has the largest margin. Given a hyperplane that separates the data, we can scale $w$ by a constant and adjust $b$ accordingly.

$$w \cdot x_i + b \geq +1 \text{ when } y_i = +1$$

$$w \cdot x_i + b \leq -1 \text{ when } y_i = -1$$ [10]
These equations are the parallel bounds of the *fat plane* which is the hyperplane separating the different classes of data. The size of the fat plane is twice the margin. [10] These two functions are used to maximize the margin of the two classes of data to provide a consistent separation in respect to training label data.

The audio extraction process within Marsyas utilizes a non-probabilistic binary linear Support Vector Machine (SVM) classifier. SVMs construct sets of hyperplanes in high dimensional space in order to classify audio samples [11]. The support vector machine partitions plots a representation of relationships between feature vectors and separates these vectors by placing them into a different feature space. This is useful for our purposes because we are trying to discern different genres on music. The feature vectors used for training in our classifier system are audio features extracted by means of audio synthesis. The labels used are notations that denote different types of genres. After training the model, the SVM is able to produce output which can accurately classify the music samples into their respective genres. Consider the possibility that the SVM makes an error in classifying a particular feature vector into its correct label. If this behavior is consistent then there is still a reliable clustering of the data. This is useful for our purposes because we are not recommending videos only according to the genre labels but genre feature vectors returned by the SVM algorithm.

### 2.6 Boosting

Integration of this recommendation system with existing recommendation systems in SmartPlayer will utilize Boosting. Boosting is a technique used to promote more effective recommendation methods over less effective methods [12]. Given multiple recommendation systems for given preference predictions, some systems will predict accurately and others inaccurately. A super vector of predictions for a particular song in the system will be created where each $i$ in the vector represents a prediction or classification method. Incorporating many different methods can help in making a final prediction. To take advantage of the multiple prediction systems, each prediction method is given equal weight towards making the ultimately final prediction (0 or 1, dislike and like, respectively). If a prediction method is incorrect in finding the correct rating for a certain sample then that prediction method is given a lower weight which degrades its influence on future predictions. Eventually, it could be the case that a particular method could be phased out of the recommendation system for making too many
incorrect predictions. Those systems with accurate predictions are given increased influence over the predictions of those systems with less accurate predictions.

The type of boosting that we will be used in the SmartPlayer recommendation system is ADABOOST.M1. The boosting algorithm takes as input a training set of m samples \( S = ((x_1, y_1), \ldots, (x_m, y_m)) \) where \( x_i \) is an instance drawn from some space \( X \) and represented in some manner (typically, a vector of attributes), and \( y_i \in Y \) is the class label associated with \( x_i \). The boosting algorithm has access to another weak learning algorithm, \( \text{weaklearn} \). The boosting algorithm calls weaklearn repeatedly in a series of rounds. On round \( t \), the booster provides weaklearn with a distribution \( D_t \) over a training set \( S \). Weaklearn then computes a hypothesis \( h_t : X \rightarrow Y \) which should correctly classify a fraction of the training set that has large probability with respect to \( D_t \). The goal is to find a hypothesis that minimizes error while training. The error is measured with respect to the distribution \( D_t \) that was provided to the weak learner. This process continues for \( T \) rounds and a final hypothesis is given by combining a vector of weak hypotheses \( h_1, \ldots, h_T \) into a single hypothesis, \( h_f \).

### 2.7 Dynamic Time Warping

Knowledge discovery in the presence of large databases is at the forefront of modern classification tasks. Databases of data are becoming larger as storage becomes cheaper and the data on these repositories is spanning over a longer period of time. Algorithms for comparing sequences of data are needed in many applications to provide a clustering of the database content. The Dynamic Time Warping method is an algorithm which can make the comparing of sequences of data a more trivial process.

The Dynamic Time warping method (DTW) compares two time series sequences by comparing each click of the time series sequence with the other sequence being compared against. It does so in a manner that yields a cost with respect to the relative differences of both sequences. Essentially, the relativeness of the sequences in factored into the algorithm such that if a sequence is similar to another in such a manner that they are similar in relative time and frequency then they are computed as having a low cost. This differs from other methods such as computing the Euclidean distance between two points with arbitrary dimension because it actually can account for variance in time and space. With computing regular distances between
clicks of the sequence, if the sequences are relatively the same then they will still be computed as having a high cost because a regular distance measurement does not account for relative time and frequency.

The DTW works for the purposes of time series analysis with respect to audio because many times certain frequency components in time do not match up to an exact measurement despite the fact that they are similar. For instance, this algorithm is used frequently in speech recognition applications because the query sample used for testing against the actual training sample may be slightly off due to human inconsistencies. There is a need to account for the slight differences in pitch and tempi when computing the cost in mapping one time series into the other. If the relative differences were not accounted for then it would be very difficult to seek a match in analyzing speech.

We propose using the DTW in computing the difference in audio feature vectors. We have found that through utilizing the DTW method you can come up with a meaningful clustering of data. Those clustering can be seen in section 5.3.1.

### 2.7.1 DTW Algorithm

Given the two time series \( X \) and \( Y \), of lengths \(|X|\), and \(|Y|\),

\[
X = x_1, \ldots, x_{|X|} \\
Y = y_1, \ldots, y_{|Y|}
\]

construct a warping path \( W \)

\[
W = w_1, w_2, \ldots, w_K \\
\text{max}(|X|, |Y|) \leq K < |X| + |Y|
\]

where \( K \) is the length of the warping path and the \( k_{th} \) element of the warp path is \( w_k = (i, j) \) where \( i \) is an index from the time series \( X \) and \( j \) is the index of the time series \( Y \). The warping path must start at \( w_1 = (1, 1) \) and the end of the warping path must end be \( w_K = (|X|, |Y|) \). The optimal warping path is found by finding the path with minimum cost when comparing \( X \) and \( Y \). The distance of the warping path can be defined as:
The $Dist(W)$ is the distance between of the entire warping path, where $Dist(w_{ki},w_{lj})$ is the distance between the two data points $i$ and $j$ from the time series $X$ and $Y$, respectively. [14]

A dynamic programming implementation is needed to compute the DTW for two input sequences. The problem is broken down into smaller increments in order to solve the larger problems. A two-dimensional $|X|$ by $|Y|$ cost matrix $D$, is constructed where the value of $D(i,j)$ will contain the minimum-distance warp path between time series $X' = x_1, \ldots, x_i$ and $Y' = y_1, \ldots, y_j$. The total cost of the warping path is contained in $D(X,Y)$ which is the total cost for mapping the two time series sequences together. Since the minimum-distance warping paths are computed for the previous iterations of the algorithm you can use their cost to compute the distance to the current $D(i,j)$. This is because in order to get to $D(i,j)$ you must either pass through $D(i-1,j)$, $D(i,j-1)$, or $D(i-1,j-1)$. Therefore, the value for a cell in the cost matrix is:

$$D(i,j) = D(i,j) + \min[D(i-1,j), D(i,j-1), D(i-1,j-1)]$$  [14]

Since the minimum possible warping path is known for the previous iterations then it is possible to conclude that the current distance must be added to this previous path in order to proceed with computing the latter paths. The distances are computed and the entire matrix is filled which makes the complexity $O(N^2)$ is $N = |X| = |Y|$. Then the warping path is calculated starting from the $D(|X|,|Y|)$ element to the $D(1,1)$ element. The $D(i-1,j)$, $D(i,j-1)$, and $D(i-1,j-1)$ are all evaluated to where the one with the least cost is the first element in the warping path. The algorithm is recursive from there.
CHAPTER THREE

DESIGN

3.1 Overview

The underlying system for SmartPlayer utilizes a streamlined system which has many components. The training system uses the Marsyas synthesis tool box to extract audio data feature vectors from audio samples only after cutting samples into chunks by using FFmpeg. Another component is the SVM machine program called “Kea” which is part of the Marsyas library. This program is run to classify music samples so that each sample can be placed into a bucket respective of its musical genre. Finally, after creating feature vectors the vectors are put into a point cloud of 10 dimensions with each dimension corresponding to a genre of music. Nearest neighbors are then computed to obtain relationships between samples. All of the components can be substituted with different methods which makes this system robust.

3.1.1 Training System

The initial training dataset was acquired from the Marsyas distributors. The audio collection contains 1000 audio files organized into ten categories each of 100 songs. Each grouping is representative of its respective musical genres: classical, country, disco, hip-hop, jazz, rock, blues, reggae, pop, and metal. Using the Marsyas utility \textit{mkcollection}, we first created a labeled collection for the ‘trainer’ genres dataset.

The marsyas utility \textit{mkcollection} is a program that creates a collection of audio files within a directory. The collections of audio files are then read in my synthesis methods and other programs in the marsyas cookbook. It is necessary to construct such audio collections to facilitate integrity of input data being fed into the audio synthesis algorithms. The format of the collections is the audio locations of the files in the collections (generally by themselves but occasionally with a label to the right for each file path).

Below is a data flow representation of the recommendation system. The file named “plot.mf” contains the YouTube video ids and their respective ratings. The files are then fed into the ffmpeg program to where they will be cut into 30 second chunks which make for faster synthesis which will happen in the next stage. The 30 second samples coming in after the ffmpeg program are put into the audio synthesis stage where MFCCs are extracted.
After the synthesis, the songs are fed into the “kea” program. Kea is the machine learning tool that is provided in Marsyas. [15] The main mode (train) basically performs 10-fold non-stratified cross-validation to evaluate the classification performance of the specified classifier on the provided .arff file. In addition to classification accuracy it outputs several other summary measures of the classifier's performance as well as the confusion matrix. [16] The format of the output is similar to Weka. This program trains the system according to a training file provided that is manually created. The training file that kea uses contains MFCC coefficients which are extracted from bextract. On the end of each feature vector there is a descriptive label which denotes the genre that the song should be contained in. The creation of this training file is done manually because a human can distinguish musical genres more accurately than an automated system.
3.2 Design Goals

The goal is to create a cohesive system that can be changed in the future to conform to changed variables. The classifier system must be created in such a way that enables future modification. The current goal is to streamline the updating, editing and pushing of new music features in the database. While running the current system it becomes necessary to include a more diverse database of musical samples. The most valued part of the design is the recommender system. It must provide very accurate results which are inherently subjective in nature.

3.3 Classifier System

There are multiple data flows through the classifier and recommendation system. The input parameters to the system are the plotting (known) MF, the plotting (known) ARFF, and the query (unknown) ARFF. The MF is the collection of the songs created with the mkcollections and the ARFF (Attribute-Relation File Format) files describes the same list of instances each
with an accompanying set of attributes. Known and unknown in this case refer to the user’s preference; the plotting set contents have been actually rated by the user previously and the query set will be identified based on this information. The ARFF files both contain 10 dimensional coordinates for each song path and the sets user preference denoted be a 0 or 1. An example is of the form:

\[0.045, 0.034, 0.067, \ldots, [0 – 1]\]

The initial size used for system testing contained 100 known entries for the plotting map and 10 unknown entries for the query set. A graphical overview of the complete system data flow can be seen in Figure 3.1.

3.3.1 Feature Database

The feature database that we base all of our experiments off of is stored in a table in the music database. The feature database contains 3 types of vectors of coefficients, which are MFCCs, Chroma coefficients, and Genre coefficients. The types of vectors that are used for the classification of music samples via k-nearest neighbor calculation are the genre vectors. These are 10 dimensional vectors such that each coefficient in the vector corresponds to a genre type (i.e. classical, reggae, rock, pop, etc…). Each vector is of the form of 10 floating decimals denoting the weight of each genre influence in the sample followed by either a 0 or a 1 which denotes a negative or a positive rating, respectively. All of the vectors that are stored in the feature database are formulated as ARFF files (Weka format file). When the feature vectors are used, depending on the method, the training and testing ARFF input files are dynamically generated.

When using these different types of methods it may be useful to concatenate multiple types of vectors into one. Another strategy is to consolidate features into only relevant coefficients. This is done by Principal Component Analysis. Principal Component Analysis is used to cut down the size of the vector by evicting certain unimportant coefficients in lieu of more significant ones. This can prove to be a good strategy in making current implementations more efficient due to lower dimensionality of the data that is being acted upon.
For the current k-nearest method utilized in the SmartPlayer system, the ARFF files are dynamically generated and then inputted into the k-nearest neighbor program. The k-nearest neighbor program take in a testing and a training file as well as K which is the used as the K in the K-nearest neighbor algorithm.

### 3.3.2 Pushing Music Features Programmatically

In order to add video data to the database, a program was created to facilitate the pushing of new video data to the database. The actual .mp4 files also needed to get moved to the storage location where all of the videos in the database resided. A push program written in Python was the way in which the movement of the features existing in videos would be extracted. Videos are placed to a location on the file system if they needed to get added to the existing database. The program runs the MFCC extraction routines and then stores the information in the database in a table called YT_AUDIO_FEATS.

First the videos are cut using ffmpeg into 30 second samples which is explained in the ffmpeg section of this paper. After cutting the videos, the synthesis algorithms are run on the samples to extract MFCC data. There are 3 different types of coefficient vectors that we use for classifying videos; Those are MFCCs, Chroma coefficients, and Genre Coefficients. These coefficients are packaged in ARFF files (Weka standard format files) and they are inserted into the database. These files are then later extracted from the database for use in some of the algorithms that are used in classification and recommendation systems or training of a current system.

After the synthesis stage has completed the videos are pushed to the massive video storage directory which contains all of the videos are that currently in the system. The reason why we chose storing the feature vectors in database tables is because it makes for ease of use and quick queries when recommending videos. When an algorithm is ran, the vectors of coefficients are pulled from the database and they are concatenated together to create a single ARFF file. These files are dynamically generated to serve the purpose of the algorithm. This file can be used for training or testing.

### 3.4 SmartPlayer Client Application

The SmartPlayer front end for interfacing with the recommender back end is an Android media player application. This client stores user ratings and relays the new ratings to the server
for processing. The player utilizes the Android APIs to provide a media player experience when capturing ratings while the user is viewing recommended videos. The main idea of the Android player is to provide relevant content to the user without the user having to choose what content to view. The recommendation system server provides the locations for the videos to be downloaded and the Android player downloads those videos from Youtube. The videos are then played on the media player interface on the phone.

When the SmartPlayer application loads, the user is prompted to enter a username and password. After entering this information the mobile application sends a request to the server, which will grant the user access to the system or reject the request. After logging in the user for the first time the server will send recommendations to the mobile application in the form of xml. This xml file contains in it the videos, descriptions, any other meta information and a uri of the location where to download the videos from. The program that the Android player, as well as the other SmartPlayer Desktop application uses, is the YouTubeDL program. This program grabs the download url from the server and downloads the video to the storage of the phone.

The videos are played from the storage on the phone and are not currently streamed. After the videos on the Android player have been rated, the server will then send new video xml to the client. The previously viewed videos are then evicted and are replaced with the new videos that will be downloaded according to the xml sent from the server.

3.4.1 SmartPlayer Viewer Interface

The goal of the Android SmartPlayer interface was to make it minimalistic and at the same time support many useful features. Initially the user interface to the phone application was intended to support the user of logging user interaction that would be relayed to the server. Through usage patterns there are certain traits that a user might have during one time watching a video that they may not have at other times. It was concluded that this information was not supporting our results so it was decided to put this on hold at the moment.

The Android Media player class is what was used in developing the front end portion of the Android Player. This class made it easier to support added features in the future and allowed for logging within the application to happen. It may be noted that the Media Player class supports streaming but during the early development of this application that option was not supported. When videos are downloaded, they are stored on the phone locally. When there is a new recommendation cycle in the queue, the old videos that were previously rated are deleted.
When a user is playing a video they have the option to go to the next video or to the previously viewed video. But, if a user fails to rate the current video as either positive or negative, the user is forced to rate the current video before they can move on to the next video in the current recommendation cycle. This safe-guard was put in place for the purpose of having consistency and integrity in relation to the rating data and the video data. Each video that has been previously recommendation must be rated as either like or dislike before the new recommendation cycle has started.

Figure 3.2: Video Rating Interface
The interface supports scanning, fast forwarding, rewinding, play, and pause. All of the controls are transparent and the user interface times out when no one has pressed any buttons for a short time and reappear when there is new user interaction happening. The only buttons that are not transparent are the like and dislike buttons. During the time of this screenshot the fast-forward button is being pressed. The interface also has a scanning control that allows the user to pick an arbitrary location in the sample to play. To effectively use the scanning control the user must drag their finger along the bar at the bottom of the screen to a desired location to resume playback.

If the user wishes to rate the current sample they have two choices. They can either rate the video during the playback of the sample or they can wait until the sample playback is complete. If the video has never been rated before then when playback is complete the user is prompted to rate the video before the next video in the list is played. This procedure for rating samples was chosen to provide the system with consistency between currently and previously recommended videos. For every video being recommended a rating must be recorded. Also before a new recommendation cycle is started, all of the previously recommended video must be
rated by the user. This method should provide the system with accurate recommendations and quick cold-starts times for new users.

3.4.2 SmartPlayer Downloader Service

The Android service for the SmartPlayer application facilitates updating and evicting of video from the local SmartPlayer video database on the phone. When the application is first started and there are videos in the android sqlite database that have not yet been downloaded, the VideoDownloadService starts. If it detects that there is a video on the database that has a download progress of 0 then the download service spawns a download thread to download that particular video. The download thread calls the YouTubeDL jar file, which is the jar that contains all of the routines for downloading a video from YouTube.
CHAPTER FOUR

IMPLEMENTATION

4.1 Overview

Implementation of the primary music classification and identification system utilizes tools from three existing projects: ffmpeg [17], the Marsyas Project [15], and the Simple Thread-safe Approximate Nearest Neighbors (STANN) library [18]. The complete system also contains several automation and data generation scripts in python, and a C program coordinating between Marsyas and STANN.

4.2 Implementation Environment

The choice of operating system environments is important when considering possible mediums for the proposed system. Having support for many different mediums is essential when attempting to obtain a large user base. The environments we have chosen are Windows, Linux and Android as well as a web based player. All of the desktop versions of the media player are written in Java FX which is a platform that can be coded in to write for multiple platforms. It was initially created to provide support for developing rich internet applications. Although Java FX has support for writing phone applications we have chosen to write the media player on the phone utilizing the Android Media Player interface. This makes for better integration into the already existing content provider implementation.

4.3 Operating System and Version

The SmartPlayer was developed using the Android Framework API level 6, targeting device platform versions 2.0 and newer. The current system is tested on Android running Android version 2.1, update 1. Current versions of the desktop player are Windows 7, Windows XP, Windows Vista and Linux.

4.4 Marsyas System

The primary method for extracting relevant metadata from samples in the database is audio synthesis. The current audio synthesis library that is being used for this system is Marsyas. There is a large array of tools provided by the Marsyas system but we are only using and
explaining a subset of these APIs in this paper. Marsyas follows a data-flow model of audio computation [16]. The development API creates a MarSystem, with three categories of components: audio block processing (such as gain, or filter), analysis (such as spectrum and rms), and block synthesis. This MarSystem forms the backbone of our classifier. Communication between components of the Marsyas system is achieved using the Marsyas \texttt{realvec}, essentially, just one or two dimensional arrays of real numbers. Support for SWIG bindings allows creation of a processing network with python connecting disjoint system components.

### 4.5 FFmpeg

The stripping of audio from a video file is achieved using ffmpeg [17]. An example command to achieve audio extraction using this method: ffmpeg –ss 00:00:30:00 –t 00:00:30:00 –i video.mp4 –ab 128k video.wav, where –ss denotes the start of the stripping and –t denotes the duration of the stripping. The –ab option specifies a bit-rate to extract at, 128k in this command. The audio is stripped from ‘video.mp4’ and put into ‘video.wav’. Since many songs have introductions that are not consistent with the remaining track, song clips are not taken from the first 30 seconds of the sample. Instead, the initial 30 seconds of the track are discarded and the 30 second clip is taken from the beginning of the remaining portion. Since very few tracks have a length of less than a minute, this was not an issue in trashing of any songs.

### 4.6 Bextract

The next step was to utilize the \texttt{bextract} binary of the Marsyas system to synthesize samples. This is a critical part of the audio analysis process as this converts the audio into its respective numerical representation through performing audio synthesis functions on the raw audio data. Specifically, the audio clips are translated into Mel-frequency cepstral coefficients (MFCCs). MFCCs collectively represent the power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear Mel scale of frequency [19]. This synthesis tools is used as the means to obtain a single feature vector of MFCC coefficients from the raw audio data fed as input. The vectors extracted are our primary means of calculating genre coefficients. These vectors are then used for our genre classification methods and it is necessary that we use the vectors for positioning songs in a multidimensional space.
The information obtained by the *bextract* command must be meaningful to humans and it must produce different results in respect to contrasting audio information. This can be done by choosing different ways in which to extract audio data so that only meaningful features are returned. Other transform operations could be employed like the DWT (Discrete Wavelet Transform). The DWT is a transform method that has an advantage over the Fourier transform is that it can obtain temporal information more accurately without changing the granularity of frequency information. This is important when you need to capture both frequency and location in time. When running the Fourier transform on a raw audio time series data (amplitude value for each sample of the sequence) a certain window size can be specified. When the window size (number of samples to be included in the transform) is too large the granularity of the time components is larger which makes the resolution for time analysis less acute. The opposing action would be to use a smaller window size which would make the cuts in time more frequent. This would result in a more general reading of frequency components in that the differences between different frequencies (which frequencies would appear in their respective buckets) would be harder to tell.

### 4.6.1 Attribute Relation File Format

An ARFF (Attribute Relation File Format) file is an ASCII text file that describes a list of instances sharing a set of attributes. [20] ARFF files were developed by the Machine Learning Project at the Department of Computer Science of The University of Waikato for use with the Weka machine learning software. The header of the arff file has the name of the relation and a list of attributes which are the columns of the data and their respective labels. An example of the head is shown below:

```
@RELATION iris

@ATTRIBUTE sepalwidth  NUMERIC
@ATTRIBUTE petalwidth   NUMERIC
@ATTRIBUTE class        {Iris-setosa,Iris-versicolor,Iris-virginica}
```

The data which contains these attributes as are in the form of a vector:
In this ARFF instance the first four elements of the vector are the attributes and the last element is the label for the data which classifies the data row. Generally, these ARFF files are used in regression and classification algorithms to come up with affinities or predictions after push these data vector through a classification method such as a support vector machine. The @RELATION, @ATTRIBUTE, and @DATA denote the sections of the ARFF file which is read by the classification tool. Generally, the elements in the attribute vectors in the data section are real numbers. For our purposes, the feature vectors denote tags which are real number normalized to lie between [0, 1].

4.7 Kea

The vectors extracted using bextract are inputted into the Kea machine learning program. Kea, also distributed with Marsyas, is similar in operation to the well-known Weka machine learning tool, only with a limited subset of capabilities. Kea was selected instead of Weka due to the easy integration with other Marsyas components. Kea uses a SVM algorithm for its training and predicting. The application performs a 10-fold cross-validation that evaluates the classification accuracy of each respective ARFF file given. The current implementation uses 890 songs for the training set. The testing set includes all of the samples in the database which is currently 60,000+ samples.

Kea supports a feature called Automatic Tag Annotation which is a practice of labeling samples based upon the sample’s content. Manual tag annotation if used in systems such as Pandora and Last.fm. The problem with the manual tag annotation method is there is well-known issue known as the cold-start problem that these systems suffer from. The cold-start
problem is described as a hindering of recommending songs/tracks until those songs/tracks have already been manually annotated.

For each track in the audio collection a feature vector is calculated based on the audio content. As each track is fed into the multi-class Audio SVM several time with different tags. Once all tags have been processed, the linear SVM is trained and a tag affinity output vector (TAV) is calculated. The TAV vector can be used directly for retrieval and storage. The resulting TAV is in the form of an ARFF which contains music genres as the feature vectors and a class denoting the classified genre of the audio sample. [21]

It is necessary to describe that specification of the outputted TAV file in detail. Formally, we begin by considering a vocabulary \( V \) that consists of \( |W| \) unique words and that each “word” refers to a semantic concept, for example “techno”, “rock”, “hardcore” or “ambient”. The goal of annotation is to find a set \( W = w_1, \ldots, w_d \) of \( A \) words that are semantically meaningful and describe a query audio track \( s_q \). We describe each song as an annotation vector \( y = (y_1, \ldots, y_{|W|}) \) where \( y_j > 0 \) if \( w_j \) has a semantic association with the audio track, and \( y_j = 0 \) if it does not. These \( y_j \) are proportional to the strength of the semantic association and are thus called semantic weights. [21] The weight that is the highest upon the semantic associations is most likely the actual class of that sample.

Considering the ARFF format described 4.6.1, you can imagine the attributes of the file being annotated tags and the output (class) attribute being either one of the annotated tag attributes in the vector. For the purposes of the SmartPlayer recommendation system, the ratings given by the users for the rows of the TAV are appended to the end of the vector denoting a preference for that given sample. The rating is either a 0 or a 1. Basically, the TAV vector has weighted semantic associations as feature vectors with a tag as the last element of the vector denoting the predicted genre for that sample. That last element is replaced by a 1 or a 0 denoting a rating of that musical sample. These feature vectors and associated ratings are used to aid in providing recommendations to users. The training file contains these ratings with the annotated semantic associations but the testing file doesn’t need to contain actual ratings because the purpose of this approach is to give a hypothesis of those ratings according to the model trained by the training data.
4.8 STANN

Once the vectors have been translated into a graph in 10 dimensional space, the next step is k-clustering to predict unknown values. The cluster of data points is translated into preferences using the Simple Thread-safe Approximate Neatest Neighbors (STANN) library. STANN uses a parallel algorithm for k-nearest neighbor graph construction using Morton ordering [18].

STANN queries the k nearest points, selected to be five for initial testing. Majority voting is then used among the returned five nodes, each having value of 0 or 1 indicating the user’s preference for the song and processing continues until all unknown songs have been given preference values. The vector returned from STANN contains the calculated indexes of the k values in the input vector that were used to initialize the graph. The process is repeated until values have been calculated for all previously unknown songs.

4.8.1 K-Nearest neighbor Implementation

The purpose of using comparison algorithms is the ultimately provide relevant predictions to users who have already rated an item as being significant. When providing predictions the relationship between items becomes relevant. Many techniques for classification produce predictions through utilizing collaborative-filtering. The collaborative methods generally use user-user filtering where groups of users are paired with each other to provide relevant samples. Other groups have employed item-item collaborative filtering methods in order to analyze the differences between items rather than users. Then, in the latter stages, recommend utilizing user-user recommendations. This technique can prove for strong classifiers.

The operational cost of the actual KNN algorithm is very low being that the implementation for KNN querying is fast given low dimensional plotting points. The KNN program takes two ARFF files as input and one integer, which is the K used in the k-nearest neighbor algorithm. The ARFFs are in standard ARFF format in that they have an attributes section and a data section but the data section is changed dynamically before it is fed into the C++ KNN program. The first parameter into the C++ KNN program is an ARFF training file that is the plotting file (file that is first plotted into the 10-dimensional point cloud). The vectors
in the data section of the ARFF files have vectors from 1 to n where n is the string denoting the
genre of the sample. An example of this vector is:

0.059246, 0.006816, 0.610851, 0.03305, …, … , 0

This vector shows the different weights distributed respective of different genres and its
corresponding rating associated with that sample. This vector is factored into the algorithm and
plotted into the other training point in the point cloud in 10 dimensions. The points are plotted
according to the feature vector not incorporating the nth node in the vector, which is the rating as
stated above. The rating is only used when computing a rating for an unknown video. After the
training file is plotted in the point cloud then the training file is read in as a command line
argument. This ARFF is of exactly the same format as the training file but the rating appended
to the vector (10 dimensional plus a rating) is not used and is a dummy value, which will be
filled in later when the program completes.

Choosing a correct K value for the algorithm is important. Consider that the point cloud
only has 50 points in the plotting points and there is the situation that an unknown rating is
desired for a particular query sample. It may be the case that all of the points are distributed
evenly throughout the point cloud. If K is too large then it may be the case that you are
considering the ratings of samples further away than anticipated which could provide incorrect
recommendations. If K is too small, say K = 1, then you could also consider the situation where
a there is only one positive rating among a large cluster of negative ratings. In this case, if the
query point is the first nearest k then you would be giving a resulting guess rating that does not
represent the user’s tastes accurately. In our experiments we would like to avoid these situations.

Through using this content based approach the relationships between items is computed
rather than the relationships between users. Content based systems can determine the
relationships between items while a collaborative system can provide a mapping between users
and items. Hybrid systems are ubiquitous in a recommendation system paradigm. The
bottleneck of using a single collaborative recommendation system is the search for neighbors
among a large user population or potential neighbors.

In [7] they use item based algorithms to explore the relationships between items first
before running collaborative based algorithms. The collaborative algorithms map items to users.
Recommendations for users are computed by finding items that are similar to other items the user has liked. The item based algorithms used can provide the system with recommendations with less computation. The similarities are computed by using item-item collaborative filter methods. The concept is to compute the similarities between two samples, say i and j, and to use this similarity to find co-rated items. The items that are rated similarly come from different user ratings. [7]

Through using our content based k-nearest algorithm the similarities between samples are computed in relation to audio content. These similarities are seen as being item-to-item comparisons where most other methods employ a collaborative approach as seen in [7]. Collaborative approaches can yield a high accuracy when having large datasets at your disposal. Large datasets are needed to use those approaches because if there is a lack in rating data then it can be impossible to recommend new content. Also, those approaches need a large amount of users to be beneficial because the similarities between items are computed by looking at other users rating information.

When employing a content-based approach it is possible to make recommendations even when there is only one user in the system. Basically, by gathering meta-data from the input samples, similarities can be noted between them. We have chosen to gather audio meta-data but there are other types of feature vectors that can be compiled to recommend samples with. For instance, the duration of a video or the language of the video could be elements in a fundamental feature vector. Basically, any information that can be used to categorize a sample can be used and run through a classifier utilizing machine learning heuristics.

4.9 Comparing Time Series Data

A time series is a sequence of data points taken as samples upon uniform time increments. This sequence can be either a single value or a vector of values. Time series data is used in many applications across the spectrum of applications. They are ubiquitous in finance applications, signal processing, and economics. They are also powerful applications towards using time series data for image processing. An example of a time series is the DOW Jones index. The value of the DOW can grow and shrink which would denote the y plane with the x plane being the increments in time during the change in y.
The comparison algorithm that is used in this paper doesn’t compare raw time series data that is measured in amplitude. Rather, it takes advantage of audio synthesis algorithms to obtain time series data. Through audio synthesis, the ZCR (Zero Crossing Rate) is computed for each sliver of the input sample. The Zero Crossing Rate is a measure of oscillation from positive to negative that occurs in the raw time series data.

$$ZCR = \frac{1}{T-1} \sum_{t=1}^{T-1} F\{s_t, s_{t-1} < 0\} \quad [22]$$

where $T$ is the length of the signal and $s$ is the signal being analyzed. $F$ is 1 if the argument inside the parenthesis is true and 0 if it is false. $F$ is a function that determines whether there has been a sign changed elapsed in the last two increments of the signal. Essentially, this equation computes the number of sign changes in the frame of analysis [22]. By determining the number of sign changes or zero crossing rate a sample has there are other assumptions and observations that can be derived. For instance, zero crossing rate analysis can be used in building a primitive pitch detection algorithm. There are also uses with this method in finding silence occurring through the duration of a sample. Occasionally, researchers do analysis on audio and before conducting experiments upon the content of the sample they cut out all of the silence that occurs. This way they can get a clear read on the actual content of the sample without incorporating silence.

Essentially, each 30-second sample is cut into much smaller increments in order to get an accurate reading of changes over time as the song progresses. The way in which the coefficient vectors are computed is similar to that of the K-Nearest Algorithm started in the chapters previously. For this particular method the song is cut into chunks of 50ms samples starting with the 30-second sample and splitting from there. The 50ms samples are then put through synthesis methods to obtain the ARFF files that are later used in comparison algorithms. These vectors obtained by gathering data from the ARFF files serve as time series coefficients that are put into the dynamic time warping scheme. Two time series plots with equivalent dimensions and size are then put through the Dynamic Time Warping algorithm. Specifically, for time series $A$ and $B$, the DTW will measure the cost of mapping $A$ into $B$ or visa-versa. This cost is the measure of
similarity between both of the time series plots. Basically, the larger the cost, the larger difference there is between the two samples, A and B.

The result of synthesis is an accurate time series plot showing the occurrences in ZCR over time for one time series. There is an example of the plotted time series shown in the figure 4.1 below. This time series is drawn through using 250 ZCR samples. For each sample of the time series the ZCR was computed and documented. The different zero crossing rate measures illustrate the differences in intensity during the duration of the entire sample. With computing two different time series, there are differences and similarities that can be noted. Similar sequences can be seen as having recurring trends throughout the sample. These similar sequences would be closer in clustering a large database of samples. It is possible that recommendations could be derived through clustering of samples using ZCR or different type of features gathered.
4.10 Dynamic Time Warping

The Dynamic Time Warping method is a tool that is used in measuring the differences between two sequences in time. It does so in a manner that incorporates the variance in time and speed. For instance, if a person sings part of a tune then it could be the case where the person sings to song too slow or it could also be the case that the person doesn’t sing the tune in the
correct octave. With using regular direct comparison methods it could be that there is not a match where the person singing the sample is off by a small margin. If the data is offset in such a way in that incorporating relativity creates a positive match to another sequence then the dynamic time warping method would be the best fitting algorithm to use in these cases.

The Dynamic Time Warping algorithm is ubiquitous in the area of Music Similarity. The need for classifying audio samples in large database is becoming more useful as music collections become larger and storage becomes cheaper. Other related works compare using DTW against high-dimensional feature vectors to come up with accurate results when finding similar samples within large music databases [23].

We will briefly discuss and define the Dynamic Time Warp and the DTW distance when comparing two finite time series. Let Q and C be two multi-dimensional time series of length n and m:

\[
Q = (Q_1, \ldots, Q_n) \quad Q_i = (q_{i1}, \ldots, q_{iu})
\]

\[
C = (C_1, \ldots, C_m) \quad C_j = (c_{j1}, \ldots, c_{jd}) \quad [23]
\]

To align two sequences using DTW, an n-by-m matrix is constructed where element (i, j) contains the distance \(d(Q_i, C_j)\) (where \(d\) is, e.g., the Euclidean distance between elements \(Q_i\) and \(C_j\)). A warping path \(W\) from element \((1, 1)\) and \((n, m)\) is created that represents the alignment between the two time series compared Q and C. [23]

\[
\text{DTW}(Q, C) = \min_{W} \sum_{k \in W} \sqrt{\sum_{i=1}^{k} w_k} \quad [23]
\]

where \(W\) is the possible warping path.

Many times in computing the DTW between two time series sequences it becomes necessary to reduce the length of the time series data being ran through the algorithm. When computing the warping path between two samples with high dimension the cost can be very high.
Therefore there are methods available to reduce the size of the time series to where there is still the desired granularity and the relevant information is not lost. This is achieved using a length reduction function that approximates the information confined within the time series such that when inputting $C$, $\bar{C}$ is derived, which is the approximated time series. [23] One method used is the Piecewise Aggregate Approximation (PAA). This approximation scheme can be used for 1-dimensional time series as well as for d-dimensional time series.

For the purposes of the comparison methods used in this paper, we use the Dynamic Time Warping method to come up with clustering based upon using MFCCs to show similarities between samples. To show what the relationships are between different samples, a similarity matrix is computed by using the DTW function on a square matrix where each element of the matrix is a positive cost. This similarity matrix represents the cost of the DTW computation between the two samples.

The two methods that will be discussed in computing warping paths both have the same comparing methods essentially but deal with different datasets. Through experimenting with different MFCC datasets, other clustering sets were discovered. Using the ZCR coefficients, time series data was extracted and plotted. With comparing time series data against each other sample, a DTW matrix is derived which shows the clustering.

Another different but similar method was experimented with which used the same type of comparison methods. With extracting different MFCC feature vectors, a different similarity matrix was created which illustrated the differences in samples in reference to pitch and harmonic amplitude rather than zero crossings. The zero crossing feature can show how intense the activity is in one sliver of an input sample for a given duration. The MFCC features can show much more through characterizing the sample with harmonic information. This is important information when discerning one sample from another using a harmonic metric.

4.10.1 Dynamic Time Warping Using ZCR

The time series data used for computing the DTW matrices was derived by means of feature abstraction via the Marsyas toolbox. The files that were created are of the ARFF format and contained Zero Crossing Rate coefficients. The command used in obtaining the ARFF files is contained in the Marsyas library and is called “Bextract”. This is one of the flagship utilities
of the Marsyas libraries and is currently used in classification algorithms across many applications around the world.

We will define the similarity matrix as being a DTW matrix. Therefore, let A be the DTW matrix of size M x N to where \( A = [a_{i,j}]_{i=1,...,M; j=1,...,N} \) and \(|M| = |N|\). The \( A_{i,j} \) entry of the matrix contains the cost in computing the DTW of the two ZCR time series sequences for the \( i^{th} \) row and the \( j^{th} \) column, respectively. After computing the DTW matrix, we are left with this matrix of size M x N that gives a clustering of the distances between songs respective of each other. The clustering in the setting of recommendation systems is useful because upon determining that a song is rated as positive, finding a similar song, which has a low cost of DTW with the positively rated song, is trivial.

The values in the DTW matrix are distributed evenly which makes for a good clustering of videos that are shown to be alike. The Figure 3 shows the values of the DTW matrix plotted out on a graph. The samples that were used in computing the DTW matrix were taken from YouTube. Their YouTube identification strings are shown in the table 5.1 below.

<table>
<thead>
<tr>
<th>Table 5.1: Youtube identifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>yn8dYSkud74</td>
</tr>
<tr>
<td>ynpPHIe4OXw</td>
</tr>
<tr>
<td>ys0xYwWE-Bs</td>
</tr>
<tr>
<td>ysNIEaQ8tbM</td>
</tr>
<tr>
<td>ynBjoYiVHz4</td>
</tr>
<tr>
<td>nPTe6YrG6I</td>
</tr>
<tr>
<td>ys33giTgr1s</td>
</tr>
<tr>
<td>ysnw-0w9UkQ</td>
</tr>
<tr>
<td>ynelgvgT9qE</td>
</tr>
<tr>
<td>ynS9ema5wYs</td>
</tr>
<tr>
<td>ys5MpSbliZg</td>
</tr>
<tr>
<td>ysPYXY4AbIA</td>
</tr>
<tr>
<td>ynG7p_cB-vk</td>
</tr>
<tr>
<td>ynSkDfbi7xg</td>
</tr>
<tr>
<td>ys87HTtQNs</td>
</tr>
<tr>
<td>yss6XKrdr6U</td>
</tr>
<tr>
<td>yngqrzPbtqQ</td>
</tr>
<tr>
<td>ynsuWboDZgA</td>
</tr>
<tr>
<td>ysBBSEFAxh8</td>
</tr>
<tr>
<td>yssSroFytdZs</td>
</tr>
<tr>
<td>yn-KTsIwSmU</td>
</tr>
<tr>
<td>ynTV3OPHePQ</td>
</tr>
<tr>
<td>yshUFah31iQ</td>
</tr>
<tr>
<td>yssZJ5OZr1Y</td>
</tr>
<tr>
<td>ynLp-iXfz9E</td>
</tr>
<tr>
<td>ynu8g6o4JBU</td>
</tr>
<tr>
<td>_iOQp0k94</td>
</tr>
<tr>
<td>ysx-vo5eq3M</td>
</tr>
<tr>
<td>ynnxW181eR8</td>
</tr>
<tr>
<td>ynzPIW0VQTA</td>
</tr>
<tr>
<td>ysKti7w5x58</td>
</tr>
<tr>
<td>ysztA41Y1aU</td>
</tr>
</tbody>
</table>
Figure 5.2: ZCR DTW matrix values scaled
4.10.2 Dynamic Time Warping Using MFCCs

Computing the time warping cost for Zero Crossings seemed to show an even distribution when plotting the cluster but there are other coefficients that can be used in computing music similarity. Mel-Frequency Cepstral Coefficients are also computed by using the Marsyas library bextract command. This will return feature vectors of size 13 that can be used to compute distance between arbitrary points in the time series. Basically, the same DTW algorithm is altered so that it can support arbitrary dimensions. At the point in which the Euclidean distance is computed in the DTW routine, instead of merely comparing two points, the distance of up to \( d \) points are computed. Each click of the time series is a 13 dimensional array. Formally, for the two time series, \( Q \) and \( C \), the distance between them is computed. The two sequences can be defined as such:
where $Q$ and $C$ are the Query time series sequence and the compared sequence, respectively. The same DTW matrix that was computed for the ZCR matrix is computed but with using MFCCs in lieu of zero crossing coefficients. This new MFCC matrix shows the relative difference in harmonics between the different input samples.

Figure 5.4: MFCC DTW matrix values scaled
CHAPTER FIVE

EVALUATIONS

5.1 Evaluation Goals

The goal of a predictive model is to provide the system with accurate predictions provided the database of music video samples in the large database. The predictive model is likely to produce results much better than a random prediction model. We plan to achieve this goal by choosing distinct features given the audio data provided. The differences between each two sets of feature vectors should be large enough to yield a clustering that is meaningful to predictive modeling. Essentially, with choosing feature vectors, the aim is to evaluate similar to how humans evaluate musical samples.

Due to the inherently subjective nature of music recommendation, evaluation of the recommendation system necessitates a human element. The question is whether the classifier and recommendation system was able to correctly predict song preference for the individual. System testing and evaluation was thus performed using user music preferences. For the testing of accuracy, videos that have already been rated by a particular user were distributed into the testing and training files for performance evaluations. The samples were pushed through the synthesis and SVM training methods available in the Marsyas tool box. These tools were used to derive 10 dimensional vectors with each element in the vectors representing a genre of music. The different genres are reggae, classical, rock, pop, disco, metal, jazz, hip-hop, country and blues. Each feature vector is of 10 dimensions and has a rating which corresponds to a user rating at the end of the vector.

For the performance evaluations there were 580 testing samples and 574 training samples. The variable K for the KNN searches was ranged from [1-60]. Generally, the K variable that is used in KNN computation of the genre vectors is somewhere around 5-10. After computing the nearest K vectors a majority vote is taken which will determine the predicted rating of the query sample. This is done for all query samples in the input vectors set. The table below shows the KNN performance evaluations for 580 testing samples and 574 training samples. The times were tested in Microseconds and the variable K is used from values [1-60].
5.2 Evaluation Techniques

To obtain accuracy results a large pool of training files and testing files are chosen from the database. The training files are evaluated using 10-fold cross validation which is a common technique for accessing how the results of a statistical analysis will categorize a testing dataset. This is done by partitioning a dataset into 10 different subsets for which 1 subset will be used for testing and the other 9 will be used for training. This is done 10 times (or for 10 folds), with each iteration delegating a different subset for being the testing set. Accuracies are taken of the prediction accuracy of each training model (the 9 training subsets) and after 10-folds are completed the average accuracy is noted. This evaluation technique tests how well the input data will fit the model.

5.3 Results

A series of accuracy tests were conducted on the K-nearest approach using the database of samples that are already existing in the database. The different measures that were documented were Root Mean Squared Deviation or Root Mean Squared Error (RMSE) and the number of incorrect predictions. RMSE represents the amount of error for each of the predictions against actual ratings averaged over the entire list of recommendations.

Each list of rated songs used for training was put through 10-fold cross validation. This metric is used to test how accurate a predictive model is using a specific dataset. The predictive model is run 10 times with each subset of 10% being used as testing and the other 90% as training.

For each fold, the KNN classifier was run 20 times for each K, 1 through 20. The best RMSE and number of correct verses incorrect was recorded and outputted. For the table below the accuracy observed is 71.566% which is the accuracy averaged for all folds of the training data.
<table>
<thead>
<tr>
<th></th>
<th>Accuracy</th>
<th>RMSE</th>
<th>Incorrect/Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>1&lt;sup&gt;st&lt;/sup&gt;</td>
<td>65.957%</td>
<td>0.583459</td>
<td>16/47</td>
</tr>
<tr>
<td>2&lt;sup&gt;nd&lt;/sup&gt;</td>
<td>68.085%</td>
<td>0.564832</td>
<td>15/47</td>
</tr>
<tr>
<td>3&lt;sup&gt;rd&lt;/sup&gt;</td>
<td>70.212%</td>
<td>0.545776</td>
<td>14/47</td>
</tr>
<tr>
<td>4&lt;sup&gt;th&lt;/sup&gt;</td>
<td>76.595%</td>
<td>0.483779</td>
<td>11/47</td>
</tr>
<tr>
<td>5&lt;sup&gt;th&lt;/sup&gt;</td>
<td>70.212%</td>
<td>0.545776</td>
<td>14/47</td>
</tr>
<tr>
<td>6&lt;sup&gt;th&lt;/sup&gt;</td>
<td>74.468%</td>
<td>0.505291</td>
<td>12/47</td>
</tr>
<tr>
<td>7&lt;sup&gt;th&lt;/sup&gt;</td>
<td>65.957%</td>
<td>0.583459</td>
<td>16/47</td>
</tr>
<tr>
<td>8&lt;sup&gt;th&lt;/sup&gt;</td>
<td>70.212%</td>
<td>0.545776</td>
<td>14/47</td>
</tr>
<tr>
<td>9&lt;sup&gt;th&lt;/sup&gt;</td>
<td>76.595%</td>
<td>0.483779</td>
<td>11/47</td>
</tr>
<tr>
<td>10&lt;sup&gt;th&lt;/sup&gt;</td>
<td>74.468%</td>
<td>0.505291</td>
<td>12/47</td>
</tr>
</tbody>
</table>

For evaluations on Dataset Two the predictive model does not perform as well mainly because most of the ratings submitted by this user were negative. With respects to the constraints the highest accuracy observed was by the 5<sup>th</sup> fold. The overall accuracy of this training data is 56.138%. This is still better than the baseline which is random predictions. For Dataset One the accuracy is rather high with the highest accuracy being 76.595%. This is a very accurate rating from a single classifying method. While using a hybrid system, it is more possible to obtain a higher accuracy rather than with a single recommender. In the scope of the entire SmartPlayer recommendation system, this predictive model serves as one component.
In evaluating the accuracies in Dataset Two the prediction accuracies are a bit lower than for dataset one. Through using different datasets there is a possibility that the data is harder to model if the ratings are unpredictable. When this occurs the prediction algorithm can have trouble coming up with recommendations. It still can be noted that the prediction accuracy of these methods are still better than a random predictive model. This means that in almost any case, this predictive model can come up with relevant samples to recommend to the user. It is possible for a recommender system to provide users with less than 50% accuracy.

<table>
<thead>
<tr>
<th></th>
<th>Accuracy</th>
<th>RMSE</th>
<th>Incorrect/# of samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>1&lt;sup&gt;st&lt;/sup&gt;</td>
<td>55.454%</td>
<td>0.667423</td>
<td>49/110</td>
</tr>
<tr>
<td>2&lt;sup&gt;nd&lt;/sup&gt;</td>
<td>60.000%</td>
<td>0.632455</td>
<td>44/110</td>
</tr>
<tr>
<td>3&lt;sup&gt;rd&lt;/sup&gt;</td>
<td>60.000%</td>
<td>0.632455</td>
<td>44/110</td>
</tr>
<tr>
<td>4&lt;sup&gt;th&lt;/sup&gt;</td>
<td>46.363%</td>
<td>0.732368</td>
<td>59/110</td>
</tr>
<tr>
<td>5&lt;sup&gt;th&lt;/sup&gt;</td>
<td>62.727%</td>
<td>0.610513</td>
<td>41/110</td>
</tr>
<tr>
<td>6&lt;sup&gt;th&lt;/sup&gt;</td>
<td>58.181%</td>
<td>0.646669</td>
<td>46/110</td>
</tr>
<tr>
<td>7&lt;sup&gt;th&lt;/sup&gt;</td>
<td>56.363%</td>
<td>0.660578</td>
<td>48/110</td>
</tr>
<tr>
<td>8&lt;sup&gt;th&lt;/sup&gt;</td>
<td>54.545%</td>
<td>0.674199</td>
<td>50/110</td>
</tr>
<tr>
<td>9&lt;sup&gt;th&lt;/sup&gt;</td>
<td>55.454%</td>
<td>0.667423</td>
<td>49/110</td>
</tr>
<tr>
<td>10&lt;sup&gt;th&lt;/sup&gt;</td>
<td>52.293%</td>
<td>0.690698</td>
<td>52/109</td>
</tr>
</tbody>
</table>
CHAPTER SIX

CONCLUSION

6.1 Broader Implications

SmartPlayer is an extensible recommendation system that can handle many more classifiers. This robust system is designed to allow for multiple types of recommender systems into a super vector containing weighted predictions. In the boosting section in the earlier chapters we discussed a classification technique that could optimize the accuracy of the recommendation system. This can be achieved by employing a super vector technique where each element of the vector is an output from a recommendation scheme. With favoring the stronger classifiers and assigning weights to the different predicted ratings, it is possible to assemble a very strong and accurate recommendation system.

6.2 Future Work

In the future the plan for SmartPlayer is to make a synchronous system where a user can store his or her videos on their own personal cloud. Currently, you can view videos on your own storage device which makes for quick playback with no buffering delay. This would be the advantage of playing the video locally. Many personal media solutions are moving to the cloud arena as space is getting cheaper.

Concerning new classification algorithms, the approach used can be enhanced. Through using different transform algorithms. More relevant features can be used in feature vectors which provides users with better recommendations. With the flooding on a large pool of users in the system collaborative filtering methods can be used as well which are highly regarded in recommender systems all over the world.

Upon employing multiple recommendation schemes into the mix the accuracy of predictions could be higher. The current recommendation system is also not yet optimized for a larger pool of users. Currently there are enough users to give results for researching purposes but in the future it will be possible for the system to support thousands of users. Through streamlining the current techniques being used the system is easily scalable.
6.3 Summary of Contributions

Through the work contained in this thesis, a classification and recommendation system framework has been designed, implemented, and evaluated. It has the properties:

- API for easy integration for producing predictions for third parties
- An interactive real time user experience
- Predictive modeling that gains momentum during user of the product

In addition, multiple recommendation strategies were used to return accurate predictions. These strategies include:

- Support vector machine methods
- Clustering and data mining
- Algorithmic and storage optimization

6.4 Conclusion

Recommendation and classification systems have made it easier to facilitate new discovery of music samples. We have shown that the recommendation system proposed in this paper can do much better than random recommending. We have also shown the similarities between musical samples according the synthesized audio features. The purpose of this analysis was to model a predictor which can show the similarities and differences between samples while only relying on the synthesis of audio. This can prove to be a beneficial tool when there is lack of user information.

Although collaborative filtering methods are very effective in building a general idea of how users interact with sample data, analyzing audio data through synthesis methods model samples according to the actual content of the data which is not susceptible to subjective scrutiny. The actual meta-data extracted from audio samples can be clustered in an effective way to illustrate the differences in respect to other samples. We have shown that this clustering is meaningful when classifying samples.

Similarities and differences in musical samples can be indicated by analyzing time series data. It is possible to analyze these differences when comparing two raw time series sequences that show changes in amplitude over time. In this paper we used feature vectors of MFCC
coefficients and zero crossings to plot time series data. These time series sequences can be compared and they can produce a meaningful clustering which shows the differences in respect to harmonic and zero crossing information.

We have also shown that through using SVM methods, a feature vector of a genre representation can be created signifying genre information for individual samples. These vectors can be drawn in 10-dimensional space and the closeness of samples directly can signify similarities. This is shown in this paper to be a powerful tool in recommending samples with high accuracy. It can be noted that it is possible to recommend a sample that no human has even seen before because the algorithm proposed is independent of any outside observers. It is also possible to only recommend videos with collecting recommendations from a single user without consulting other users’ recommendations. This is a virtue of content filtering. Recommendations systems can be created through only using audio synthesis methodologies.
REFERENCES


BIOGRAPHICAL SKETCH

Robert Griesmeyer was born in Hollywood, Florida and graduated from Hollywood Hills Highschool in 2003. After graduation he started a career in Music Education and during that time Robert did one US tour with the music group Magic of Orlando in 2004 and two more tours with the Boston Crusaders in 2005 and 2006. After attaining his Associates degree in Music Education he moved to Tallahassee to attend Florida State University to study Information Technology. After gaining profound knowledge in computer systems he graduated with a B.S. in Information Science.

Robert Griesmeyer is currently a graduate student in Computer Science. His research areas include Recommendation Systems, Support Vector Machines, Audio Processing and data mining. In the last few years he has worked with Piyush Kumar and the rest of the SmartPlayer team in building a robust recommendation system for the SmartPlayer Media Player. Robert’s contributions include his implementation of the K-nearest neighbor recommendation system as well as his research in Audio synthesis methods relating to content filtering methods. Robert is the primary developer of the Android Smartplayer project and is currently maintaining this implementation.