MUSIC CLASSIFICATION AND IDENTIFICATION SYSTEM

Alan P. Schmidt   Trevor K. M. Stone

Department of Computer Science
University of Colorado, Boulder
{schmidap, tstone}@colorado.edu

ABSTRACT

We have implemented a system to recognize and classify music. Given an audio sample; our system is able to determine the genre, artist, album, and title of the given song. This is done by creating a database of models based on songs it's trained on. Input songs' features and models are compared with existing models to determine the identity of the song, along with its genre, artist, and album.

This system is robust enough to handle thousands of songs, and allows for classification of any number of input samples. Since the system is classifying songs on an acoustic level, it is immune to small amounts of noise, added silence, and encoding rate of the input sample.

1. INTRODUCTION

While a great deal of work has gone into improving and expanding computer speech recognition systems, recognition and classification of non-speech sound domains has received little attention. The techniques developed and used in speech audio domains are not very specific to any particular type of sound. With this in mind, it should be possible to extend and use the same techniques in the recognition and classification of music. We have shown that it is possible to implement a system using the speech audio techniques to classify a music file by genre, artist, and album and to compute the identity of the music file without resorting to linear search.

2. BACKGROUND LITERATURE

Using Hidden Markov Models to represent generators of sequences of data has long been shown to be an acceptable solution for recognizing the origin of an observation. HMM's have been used in many areas of speech recognition, as the problem is inherently this: that our system is observing some sequence of data and we need to classify the state from which it most probabilistically came. For continuous data (where there are no specific symbols being generated by a state), it is possible to use a Gaussian distribution to represent the probability of observing some output. It has been seen that if we can guarantee that there are never any state transitions, the Gaussian distribution used with a single model can be used to tell the probability of that state observing the given sequence. Gaussian Mixture Models are an extension of this idea.

Training of Gaussian Mixture Models to classify various data has been done for many research projects. In recent times, systems using GMM's have been trained to recognize and classify audio. These systems have been shown to be quite successful in the areas of speaker detection [6] and the tracking of speech through audio documents containing speech, noise, music, and combinations thereof [7].

Gaussian Mixture Models have also been used to classify instruments based on small music samples. Marques and Moreno [1] were able to construct such a system using Gaussian Mixture Models. Their conclusions indicate that use of GMM's for classification of non-speech audio can be done. There were limitations in this approach of only using small audio samples. This research implies that classification of entire songs using mixture models may be feasible.

A company called Relatable [2] developed an open-source song finger-printer called TRM that is similar to what we here propose for song identification. TRM interacts with a closed-source database that stores song metadata. TRM analyzes the first 30 seconds of a song for 34 features. Relatable's website does not give a description of their method beyond that, but it seems likely that they use Gaussian Mixture Models. Relatable reports over 99% accuracy on mp3s encoded at 96 kb/s, so we feel our performance goals below are justified. TRM and Relatable's database were used by Napster to block trading of protected songs.

The straightforward approach to speaker identification and classification with Gaussian Mixture Models is linear search. Given an audio sample, a system extracts a collection of features such as Mel-Frequency Cepstral Coefficients. It then, for each GMM in the system, computes the log likelihood that the features were generated by that GMM. The GMM which has the highest likelihood for having produced the presented features is considered the class of the audio. The number of calculations required to calculate the likelihood for a model makes such a linear search algorithm infeasable for a practical application with a large number of GMMs. For a system like ours, therefore, it is necessary to organize the GMMs in a way to reduce the number of GMMs tested for a match while still retaining reasonable classification accuracy. Further, since the time to compute a GMM increases with the amount of training audio, a database system cannont recompute a model for a broad class, such as genre or artist, with each presentation of new songs. Our system must therefore be able to make broad classifications without a GMM for that specific class.

Classification of songs into more general genres (or more specifically, regions of acoustic similarity) by training an unsupervised neural network has been experimented with. Research done by Fruhirth and Rauber [4] uses a self-organizing map to provide a clustering procedure that is shown to successfully cluster songs with similar acoustic properties.
Similar approaches using neural networks and vector-machines have been used to create systems capable of recognizing the artist of a song. The work done by Whitman, Flake, and Lawrence [5] supports the idea that clustering of songs based on classification of the audio can be done on areas other than by song or by genre. Applying these ideas into our system should help in clustering models.

Work has also been done in data clustering directly on Gaussian Mixture Models. Using vector quantization, data can be partitioned into clusters based on some given cost function. Research by Buhmann [8] suggests a method for clustering data in this fashion with optimizations to avoid running into an NP-hard combinatorial optimization problem. These techniques should provide a good base for our classifications.

3. OBJECTIVES

3.1. Song Identification

Our system should be able to correctly identify 95% of songs it has trained on using a brute force classification approach. Methods that limit the set we test against should reduce both the speed heavily and reduce the accuracy minimally. The system should be able to recognize songs that have been tampered with by adding chunks of dead audio, altering bit-rate, or those whose original quality is less than the example presented for training.

3.2. Song Classification

Our system should correctly classify a song according to genre, artist, and album at 95%. This, however, is assuming that the system has been trained sufficiently to have built a reasonable model of that class. We will test the system's classification abilities by presenting it with known and unknown songs by various artists and in various genres.

3.3. Database Maintenance

In a production system, users should be able to flag a database entry as possibly incorrect. The system will learn about new songs submitted to it and will update its genre and artist knowledge accordingly. The system's classification performance should not significantly worsen as new genre and artist examples are learned.

3.4. Additional Features

A production system should also have a song-recommendation feature. Given a song or set of songs a user likes (possibly accompanied by a set of songs the user doesn't like), the system will recommend another set of songs, based on acoustic similarity, that the user may like. Given sets of songs the user likes or dislikes, the system will be able to guess whether a user will like or dislike a given song. We did not have time to implement an interface for such a feature, but our system does calculate a set of "nearby" songs, so it should be straightforward to add this feature.

4. SYSTEM DESIGN

The main system running our system is a Pentium 2 400 MHz running with 128 MB of ram and 94 GB of storage space. All of the code written for this project is written in C, C++, or Perl. The choice of language was very dependant on the task that needed to be solved. Any calculation that is computationally expensive is written in C. Several operations that involve keeping track of results, sorting, and operating on them have been written in C++. Classification based purely on GMMs was implemented in Perl for speed of development and ease of list maintainance. Finally, functionality dealing with large numbers of files and directory expansion was implemented in Perl.

We use a few pieces of freely available software to aid our task. We use mpg123 [11] to convert audio data from MP3 format to raw audio data. Songs will be analyzed based on data from a feature extractor provided by Bryan Pellom. This feature extractor extracts 19 Mel-Frequency Cepstral Coefficients and 1 normalized power vector for each 10 ms of audio.

For our system, we decided to build a Gaussian Mixture Models for each song in our training set. Training into a GMM involves converting the MP3 file into raw audio, converting this raw audio into a series of feature vectors, and then training a model on the input vectors. Testing for identity involves taking in an audio sample's features and testing them against a subset of trained models. We also experimented with classification based on GMM similarity, without computing likelihoods.

4.1. How to Represent Audio

Audio can be represented in any format from which you can convert to WAV format. While this can be any format, for the duration of this paper we focus on using MP3 files. Once a file is in WAV format, we have constructed a set of utilities to convert to raw audio, down sample to 16 kHz, and convert to mono. The reason for these conversions is to comply with the feature extraction code available to us. We have converted to mono for two reasons. First, computing features and models for mono audio files takes half the time of stereo files. Second, we have experimentally determined that mono files provide better overall models.

4.2. MP3 to Feature Conversion

To convert from MP3 to features, a number of steps had to be taken. Most of these steps are done with the third party software and can be run within a script. The first step is to use the mpg123 to generate a WAV file. Fortunately, mpg123 provides an interface to down-sample a file to both mono and 16 kHz as it's converting to WAV. Once a WAV file has been generated it has to be converted to raw audio. Calling a simple program we have written that simply strips off the header information does this. With a valid raw audio file, it is simply a matter of calling the available feature generation code on the file.

4.3. Feature to Model Calculation

The calculation of models based on incoming feature vectors is done using a GMM class that wraps around the C code we wrote before designing the project. The GMMs we trained had 16 mixtures of 20 means and 20 variances each. The training program simply initializes a new GMM, trains it on the incoming feature vector, and outputs that GMM to a specified model file. This GMM file can then be read and manipulated as our approach dictates.

4.4. How to Classify Input Audio

Classification of the input audio involves two basic steps. First, the input audio needs to be converted into the appropriate test input. Depending on the classification method, this can be either the MFCC features or a GMM constructed from them. Once this input file is calculated, it is passed to the appropriate classification method.

5. CLASSIFICATION METHODS

5.1. Brute Force

This is the most basic of all classification methods. In essence, a test song's features are tested against every possible model. The model that most probabilistically matches the test song is returned as the match. This method lacks any sub-classifications such as genre, artist, or album. The way that these would be classified is by simply reading information about the matched song as stored in the filename or in the metainformation stored within the MP3 file itself. This approach is the linear search, which we earlier indicated was infeasible. We thus provided this for baseline performance comparisons.

5.2. Decision Trees

Decision trees are a general structure of grouping items into a tree-like structure, and then following the branches of the trees based on a rule determined for each node as the tree was built. The type of items in the leaves and the decisions to get there can be anything. For this system, we show two approaches.

5.3. Decision Tree – Top Down

Using a weighted sum vector, we built a decision trees for artist and genre using the ID3 algorithm developed by Quinlan [9] as described in Mitchell [10] for continuous-valued attributes. The weighted vectors were computed by sorting GMMs' mixtures (mean and variance vectors) by the weight of that mixture to normalize vector element order between GMMs. We then multiplied the weight by each vector element and summed each mean vector and each variance vector, concatenating the two sum vectors to produce a final 40-dimensional weighted vector. The class of each GMM was computed by hand at the artist level. (I.e. we grouped all songs by a given artist into the same genre.) The tree learning did not prune used attributes, since they are not binary and multiple splits can be obtained using different ranges of values for the same attribute.

5.4. Decision Tree – Bottom Up

In this method, the "natural" tree structure that can be associated with how MP3s are arranged on a system is used as the tree. Typically, MP3s are stored in a user's music directory. Within the music directory, directories for each genre exist. Each genre directory contains directories for all artists of that genre. This structure continues for both albums of an artist and songs of an album. Our implementation of this method calculates a combined model for each album based on the songs in the given album's directory. A model is calculated for an artist by combining the models for the artist's albums. This is continued for artist and genre. The GMMs are combined by computing their average mean and variance vectors. Higher-level combined models are based on the GMMs one level deeper, rather than the individual song GMMs.

To traverse the tree, the test song's likelihood is computed for each of the genre models. The most probable model directory is entered and the process repeats for artists. This is continued until we reach a song. This is the strict version of this method.

As a loose variant, not only is the most likely genre/artist/album/song checked against the test song, but the next 4 likely models are checked as well. Doing this creates a larger search space, but accounts for possible faulty models.

5.5. Clustering

Using weighted sum vectors computed as described in section 5.2, we organized GMMs into clusters, considering each cluster a class. We applied two variations on this approach, one based on human-labeled data and one that used an unsupervised learning technique.

To classify a GMM in our clustering schemes, the program computed an ordered list of the nearest (in Euclidean distance) cluster means. The program then determines, for each of the j clusters with closest means to the GMM, the k nearest vectors as a "possible match" list. In an application setting, the models in the possible match list would be used to determine the likelihoods of the features for the presented audio file to attempt identification. We did not take this final step, however, since we tested the system's ability to classify models of audio that hadn't been included in the training set. This multi-level "nearest point" scheme could be improved by dividing large clusters into smaller clusters to reduce the amount of points that must be examined to construct the possible match list.

We ran both supervised (human-labeled) and unsupervised (machine-labeled) tests of clustering. In the supervised case, we calculated the mean vector for each class (artist or genre), allowing an example to reside in multiple cases in the genre case. In the unsupervised case, we used the ("hard") k-means algorithm as described in [10]. The program first randomly assigned each GMM to one of j clusters, not allowing multiple-membership in clusters. The program then iteratively recalculated the mean of each cluster and the classification of each example, based on the new closest mean, stopping once the system was stable.

6. RESULTS

Initial results for all method indicate that the test machine used is not nearly optimal for an initial coding of this system. Since speed and time are such an important issue, it can be seen that narrowing down the test set in some fashion is needed for the system to be at all usable.

For the following results, we used the classification methods with approximately 1500 songs. Due to time constraints, some of the results come from what we view as a legitimate subset of test data.

6.1. Brute Force

Since this method checks a song against each model, it is incredibly accurate. However, since this method checks a song against each model, it is incredibly slow. Running the classifier on all data resulted in a classification rate of 99%. This is above our initial projected requirement. However, the time to compute this result was 3 days on a very fast machine.

In order to limit the time it takes to run a test, a limited subset of a test song's features were used. Testing the first N feature vectors against all model resulted in the following classification rates:

n	% correct
100	31.8
300	64.8
500	76.6
1000	89.7
3000	96.2

It is interesting to see that using only the first 30 seconds of audio results in a classification rate not far from that of using an entire song's features. This happens because most songs in general have an internal structure that includes repetition of common themes. It only takes one pass of these themes to get an accurate classification. The songs that this method does not hold true for are those with extended intro segments or more advanced musical ideas. Although this is a problem, it is only a problem for a small subset of songs. For this reason, it is advisable to only test on the small subset of features to speed up classification. Only testing the beginning of the file would be problematic if tested songs had silence or noise inserted at the beginning. This could be a problem, for instance, in a system designed to detect unwanted songs on a file-sharing network.

6.2. Decision Tree – Top Down

Using the ID3 algorithm we built a decision tree for artist and one for genre, both based on human-supplied class labels. The tree was built using our 1505-model corpus, holding out 151 as a test set. The test set was evenly distributed across artists. Each leaf of the decision tree was labeled with the majority class of examples used to create it. To test a song, we followed the path indicated by the decision tree rules and compared the song's class to the class of the leaf it reached. Our results are listed in the table below.

Class	Number of Labels	Number of Matches
Artist	78	47, 31.1%
Genre	15	75, 69.7%

6.3. Decision Tree – Bottom Up

Strict:
When using this model, a song is only ever tested against models that are a subclass of previous best matches. Because of this, the test is much faster than other methods. However, because of the strict model following nature, the accuracy is not very good at all. Testing done on a limited subset of the data results in only a 5% classification rate.

Loose:
Being able to test against the "runners up" adds a great deal of backup to the system. Although the test set is larger than the strict method, it is still far smaller than that used in brute force. Using the same limited subset for testing, a classification rate of 60% has been observed.

For both of the above methods, the main problem comes in the decision-making part of the process. The system compares against conglomerate models when deciding which paths to follow. It is our belief that construction of these models based on the genre/artist/album/song hierarchy is not a feasible approach. Many artists do not stick to any one genre. This follows to their albums and songs respectively. Also, some artists can easily fit into more than one genre (e.g. rap metal, Celtic punk, and alternative rock). It is believed that if the base models were more solidly constructed, this would be a valid approach. We discuss automated solutions similar to this below.

6.4. Clustering

To test our clustering classification system, we used our 1505-model corpus, holding out 151 (evenly distributed alphabetically by artist name) as a test set. We compared each test model with the 5 closest models in the 2 clusters with mean closest to the model. We calculated the number of test examples that matched the human-determined class (artist or genre) of at least one model from the possible match list. In the unsupervised case, we also calculated the percentage of possible matches that shared a class with the test examples. These results are shown in the table below. These results show that unsupervised clustering outperformed supervised clustering for both artist and genre classification.

Class	Type	Clus ters	Matches	Avg. Match Rate
Artist	Unsupervised	100	112, 74.2%	22.3%
Artist	Supervised	78	48, 31.8%	
Artist	Supervised – Full Vectors	78	49, 32.5%	
Genre	Unsupervised	30	137, 90.7%	45.8%
Genre	Supervised	15	95, 62.9%	
Genre	Supervised – Full Vectors	15	90, 59.6%	

For these experiments, a song that shared a class with any of the 10 models on its “possible match” list was considered a match. The Average Match Rate column indicates for the unsupervised tests what the average percentage of matches among the “possible match” list was. The number of clusters for the unsupervised cases was chosen somewhat arbitrarily, with the intention of allowing the system a little freedom to split broad classes into more similar parts. This ability is likely why the unsupervised clustering proved more successful – many artists produce songs which sound fairly different, so a song may not be close to its artist’s mean vector even though it lies within the convex hull of examples of that artist’s work. This characterization is especially true of genres, where human classifications such as “rock” can encompass a wide variety of soundscapes.

The fact that using all 640 elements of the GMM vector offered no performance gain is encouraging, since using full vectors requires a significant performance reduction. Unsupervised clustering tests (which require several iterations to establish the clusters) did not complete after several hours, and were terminated so that computer time could be used on more feasible tests.

Unsupervised clusters also significantly outperform decision tree classification (discussed in section 6.2), while supervised clusters and decision trees show roughly equal performance.

7. FUTURE WORK

7.1. Objectives Not Accomplished

In the initial discussion of this project, we proposed to give the user database modification and maintenance options. Due to time constraints, our database does not have as many examples as we had hoped and we did not have time to develop easy-to-use tools to maintain it.

We also only have a very limited song recommendation feature built in to the classifiers. Most of the nearness calculations are already in place, so this could be easily built in.

7.2. To Make the System Better

Some of the first improvements that could be made to this system are to use approximation methods to speed up the training and testing of models. Tricks like this can be applied throughout the system to speed up code by a large factor. One such method is to reduce the number of samples whose features are examined. We found that for files of a few minutes of audio, model building took

For bottom-up decision trees, we need to identify the best way to construct models that are representative of a collection of other models. This problem may be easily solvable if training speed can be enhanced. One approach would be to build the hierarchy based on the unsupervised clustering method rather than hand-labeled collections.

It is very important to continue experimentation with clustering algorithms. Limiting the test set is the prime way to enhance and scale system performance. Clustering also gives a very easy way to recommend songs to the listener and to cluster genres, artists, and possibly albums. Further experiments should include other clustering methods and varied numbers of clusters, perhaps even allowing for an adaptive number of clusters.

This system currently has no good user interface other than terse usage messages for a large collection of scripts, subscripts, and executable programs. Creating a user interface for this application would make the system usable by people other than the designers. With a programmer’s interface, the system could be incorporated into existing systems; for instance it could allow an MP3 player to check a database to determine if it has accurate metadata about songs.

7.3. Interesting Ideas

One of the most interesting aspects of this system that needs further investigation is to identify which of the

MFCC features actually apply towards music. In addition, how can we use the features to gain even more knowledge about a piece of music? Determining this would help greatly in all areas of clustering and combinational model building.

This system can be expanded and used in any number of applications. One use is in a file sharing system to block unauthorized song uploading. A radio could be fitted with access to this system so that the user could determine what song was playing without waiting for a DJ announcement. The system could be used to recommend new songs to a user based on expressed preferences or it could make music selections based on the songs a user has recently played, hopefully matching the mood the user is in. The possible uses are varied and socially very interesting.

8. REFERENCES

[1] Marques, Janet and Pedro J. Moreno, A Study of Musical Instrument Classification Using Gaussian Mixture Models and Support Vector Machines, Cambridge Research Laboratory, 1999

[2] Relatable, LLC, company website (news and tech sections), http://www.relatable.com/

[3] Lee, K.F., Automatic Speech Recognition: The Development of the SPHINX SYSTEM, Kluwer Academic Publishers, Boston, 1989.

[4] Fruhwirth, M. and Rauver, A., Self-Organizing Maps for Content-Based Music Clustering, Department of Software Technology, Vienna University of Technology, Wien, 2001

[5] B. Whitman, G. Flake, and S. Lawrence, Artist Detection in Music with Minnowatch, Proceedings of the 2001 IEEE Workshop on Neural Networks for Signal Processing, Falmouth, MA, pp. 559-568, 2001

[6] D. Reynolds and R. Rose, Robust Text-Independent Speaker Identification Using Gaussian Mixture Models, IEEE ASSP, vol 3, no 1, pp. 72-83, January 1995

[7] M. Seck, I. Magrin-Chagnolleau, and F. Bimbot, Experiments on Speech Tracking in Audio Documents using Gaussian Mixture Modeling, IRISA, France

[8] J. M. Buhman, Learning and Data Clustering, Rheinische Friedrich-Wilhelms-Universität, Bonn, Germany

[9] Quinlan, J. R., Induction of Decision Trees, Machine Learning, 1(1), pp 81-106, 1986.

[10] Mitchell, Tom M, Machine Learning, WCB/McGraw-Hill, 1997.

[11] Hipp, Michael, http://www.mpg123.de/