Implementation and optimization of a music recognizer based on Gaussian Mixture Models

Pedro Díaz (53452F)

Helsinki University of Technology
pdiaz@cc.hut.fi

Abstract

This paper describes the implementation of a music recognizer based on Gaussian Mixture Models. This recognizer implements most of the ideas outlined on my midterm paper, although due to lack of time some of them are not implemented completely. The performance of the implementation is quite good but there is still a lot of space for further improvements. The most promising improvements are outlined in section 4 of this document.

1. Review of project goals and objectives

The project's main objective was to build a music recognizer within a client-server architecture and with special emphasis on query optimization. This goal was met, although due to the lack of time to complete the project, some planned features are still incomplete or not even implemented. The following sections describe how the system was implemented.

2. Project implementation

2.1. Software description

The project software comprises several specialized command line utilities, written in Python and C. Python was used on the utilities that required heavy string processing or database access and little computation. C was used to implement the computationally heavy part of the project (GMM training and testing). The following list describe the role of each utility on the system :

- `songfeats.py`: This is a python script that extracts the features of a given song file. It uses the program `sox` for converting the song file to a raw 16Khz, 16bit PCM format. After this the `fea` program (included with the SONIC toolkit) is used to do the actual work and extract the song features. The extracted features are stored in a file with extension `.fea`
- `trainGMM`: This is a C program that reads a features file (`.fea`) and trains a Gaussian Mixture Model with the observation vectors of the features file. The user can specify how many components to use as well as how many iterations to perform. The output of this program consists of a binary dump of the GMM parameters (weights vector, mean and covariance matrices). The filename extension for the output of this parameters is `.params`. Each features vector is composed of 13 values: 12 from the MFCC and 1 for the energy. No Delta or Delta-Delta values are used.
- `newauthor.py`: This python script takes a `.params` file and an author name and inserts this information in the Database. It returns an author ID (positive integer, unique for all the authors).
- `newsong.py`: This python script takes a `.params` file, an author ID and a title, and inserts this information in the database.
- `queryGMM`: This program takes a `.fea` file and queries the song database for the song information. Several search methods (explained later on this document) and optimizations are available to the user

2.2. Optimization techniques used

Since implementing an efficient recognizer was one of the main objectives of the project, the topic of implemented optimization techniques deserves a subsection on its own. All the optimizations implemented fall under one of these two categories:

- **Database organization:** Techniques to improve the search performance by somehow organizing the contents of the database.
- **Log-probability computation optimizations:** Optimizations done to calculate the log-probability of a set of feature vectors against a GMM

2.2.1. Hierarchical organization of the database

The most important optimization regarding database organization is the **hierarchical organization** of the songs in the database. This technique consists on arranging the

songs hierarchically so that each parent is a GMM that models all the songs that are its childs in the hierarchy.

On the implementation a simple two-level hierarchy is used: songs are grouped by its author. Further improvements on this aspect would be to extend this hierarchy to more levels.

2.2.2. Log-probability computation optimizations

As stated in my midterm paper about the project, the calculation of the log-probability can be optimized a lot precomputing some values. The software implementation provides three versions of the log-probability procedure:

- A "verbatim" version, where the formula is implemented without any kind of optimization or precomputation.
- An optimized version, which implements the same function as the "verbatim" version (i.e.: both versions are mathematically equivalent) but uses all kind of tricks and optimizations in order to improve performance.
- A nearest neighbor version, which implements a Nearest Neighbor search across all the components. Since some simplification and assumptions are made, this version is not mathematically equivalent to the above two. It can be seen as a quick estimate of the log-prob value.

3. Experiments performed and software review

This section describes the experiments performed once the project implementation was done. These experiments are not meant to be an exhaustive performance benchmark of the system, since I had not enough time or resources to build a suitable song database.

3.1. System setup

A set of about 800 songs was processed. Each song was modeled with an 30-component GMM, which was trained for 10 iterations. Each artist was modeled with a 80-component GMM, which was also trained with 10 iterations.

Since training a GMM is the most computationally expensive procedure of the system, the features of each song were filtered with a 1:15 ratio. This means that only 1 of each 15 feature vectors was actually used for training the GMM. This filtering method was empirically proved sufficient to let the GMM "capture" the sound of the song.

3.2. First tests

The first tests done showed that the system performs quite decently regarding accuracy and speed. Most of the songs

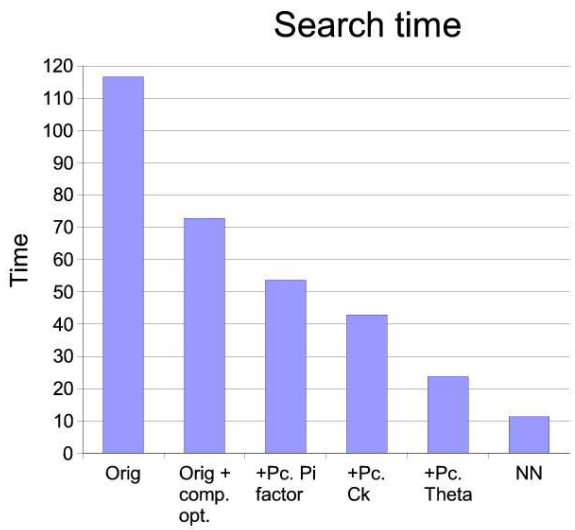

Figure 1: Performance of several log-probability function versions

[1] got recognized correctly, and, due to the optimizations and use of a compiled language, the speed was very high compared with the program used in homework #3 of the course (speaker identification). Although the first impression was positive, two problems with the current implementation began to become clear:

- A two-level hierarchical database is not enough, even with such small song database. While much faster than just a linear search, hierarchical searches for long songs could take as long as 10 seconds.
- One of the foundations about nearest neighbor search is the tradeoff of accuracy for more speed. This approach seemed to work well while searching within a set of song GMMs, but failed more than the other methods while trying to determine the author of the song. This is probably due to the fact that author GMMs model a much wider "sound concept", and therefore probably more than one component of the GMM is important for computing the final log-probability.

3.3. A small query performance benchmark

Figure 1 shows search time for a song query. The song used to do the query is specially large, 60 minutes and 2 seconds.

Several versions of the log-probability function are tested. From left to right:

[1] About 99% with a normal log-likehood computation and around 95% if using a nearest neighbor search, but this figures must be taken with a grain of salt since the song database is too small

- The original ("verbatim") version, without any kind of optimizations.
- The original ("verbatim") version, with compiler optimizations.
- Three optimized versions, each one adding a new precomputed term to the previous version
- A nearest neighbor search.

The benefits of precomputing values or using a nearest neighbor search are obvious from the figure.

3.4. Guessing the author of an unknown song

An interesting experiment performed was to query the system about a song not stored in the database. Obviously the system can not guess the title of the requested song, but, interestingly enough, it can correctly guess the author of the song in most of the cases (provided that the author is present in the database). This author search can be implemented by only searching in the top level of the database, which consists of author GMMs.

Another interesting result in this type of searches is that, when the search fails to correctly determine the author, the incorrect author is somewhat related to the actual answer. For example, when querying about an unknown song performed by Frank Sinatra & Bono [2] the system answered with the author "Frank Sinatra"

3.5. Incomplete and modified song files

Another interesting experiment performed was to query the system using feature vectors extracted from modified or incomplete song files. On most of the cases the system was able to guess correctly the author and title of the song. So far, and based on the experiments performed in this area, the following conclusions can be made:

- When using an incomplete song file for the query, the sampling and feature extraction process is crucial for the success of the query. The first prototypes of the system used a very simple sampling process where the song was not uniformly sampled (each X seconds 100 features were extracted). While this sampling process performed good when the query used the features of a complete song file, it had very bad results when the song file used to query was just an portion of the song. The current sampling process (extract all features of the song file and then take only 1 of each 15) looks like is quite robust and performs well under this situation.
- The accuracy of the results do not seem to be affected by the encoding of the song file. The system was able to correctly match an unknown MP3 file with an OGG song in the database.
- Several other modifications (such as adding line noise, using a low-pass filter, playing the song at double speed) where done to the songs used to query the system, and the system still answered correctly to most of them.

[2] Duets are treated as a different artist in the database

4. Further improvements

As it has been said, the current state of the system is far from finished. There are a lot of improvements that could be done in other to improve both accuracy and speed:

- Regarding speed, maybe the most promising improvement would be to create more hierarchy levels in the database. Rather than creating by hand these categories (like we did with the author hierarchy level), a clustering algorithm could be used in order to improve the reliability and accuracy of the searches.

  Another option would be to parallelize the search between several processors. This parallelization could be done query-wise (distribute the client queries to each processor of the system) or search-wise (for each query conduct a parallel search).

  Heuristics to filter the search space are also another interesting path. The general strategy would be to, after completely determining the search space, quickly filter out the unlikely candidates before starting the (computationally) costly log-prob test procedure. Examples of these heuristics could be:

  – Take into account the song duration, if we know it. Calculate a safe margin around this value (e.g.: $\pm 40\%$) and filter out the songs whose length is not inside this interval. This is a fast filtering technique that only requires a slight modification of the song database (include another column with the song duration)
  – Some music formats include metainformation about the song (ID3 fields in MP3 files for example). While this information can not be assumed to be correct or complete (why are we performing the search otherwise?) it can be used to narrow the search space by only testing elements with "similar" metadata. The notion of "similarity" between strings could be implemented with string matching algorithms.

  Taking into account the final objective of the system can also be useful. If the system is to be deployed as a commercial song recognition service then it is likely that most of the queries would be about a small set of songs (popular and new songs).

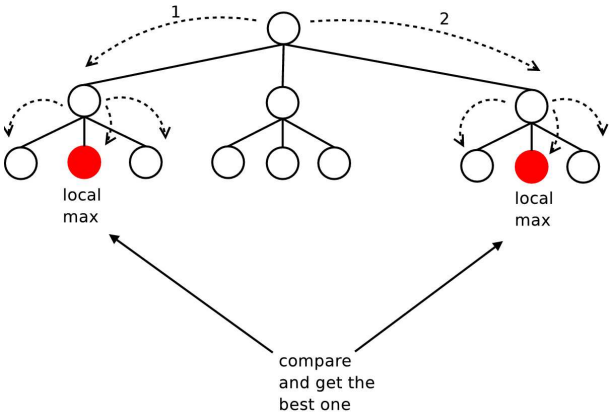

Figure 2: Beam search over a 2-level hierarchical database

Therefore, a small *cache* containing the most requested songs could be implemented and searched before anything else.

- Accuracy could be improved , specially when using the nearest neighbor search, extending the current hierarchical search to resemble a beam search. Figure 4 shows a beam search with branching factor 2 over a 2-level hierarchical database. The exact branching parameter should be determined empirically, since a too high branching factor would reduce the search speed and a too low factor would undermine the accuracy.

  As I have said previously, a nearest neighbor search in the higher levels of the song hierarchy (the author level in the implementation) can do more bad than good, due to the lost of precision when calculating the log-probabilities. Using a more precise log-prob function in these higher levels should improve accuracy, at the cost of an increased search time.

5. Conclusions

Although the system implemented lacks of some features and improvements, it serves as a proof that Gaussian Mixture Models can be used to recognize music with a high degree of accuracy and reasonable speed.

This project is also a good example of how performance can be improved by working at several levels at once:

- By improving the algorithms behind it, such as the nearest neighbor search
- By improving the data structures, such as using an hierarchical database
- By reorganizing the code, precomputing some values
- By using optimizing compilers

I also consider this project a good start point for someone who wants to research more in this area. The most essential features are already there and therefore no extra time has to be wasted reimplementing the "skeleton" of the music recognizer.

6. References

[1] Reynolds, Douglas A. and Rose, Richard C., "Robust text-independent speaker identification using Gaussian mixture speaker models", IEEE Trans. Speech and Audio Proc., 3(1):72–83, 1995.

[2] More, Andrew W., "Clustering with Gaussian Mixtures", `http://www.cs.cmu.edu/~awm/tutorials`

[3] S. B. Davis and P. Mermelstein, "A Comparison of Parametric Representations for Monosyllabic Word Recognition in Continuously Spoken Sentences", IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-28, No. 4, pp. 357-366, August 1980.