Konstantin Stekhov, Matthew Ramos
CSS 457
Final Project Report
12/10/2014

Audio Retrieval Based On Vector Quantization

Summary

This project introduces the basic audio retrieval software based on the vector quantization technique and the Linde–Buzo–Gray algorithm (LBG). In this program we get a set of training audio wav files and add them to the program in order to construct a codebook (a library of files' characteristics that matches the input files with similar files saved in the program). We do this by determining the Mel-Frequency Cepstrum Coefficients (MFCC), which are the representation of an audio file that we can use for comparison and apply the LBG algorithm for inserting MFCCs into the data codebook which we then use for input audio query comparison to the library.

Finally, the program will display the files that match the input query and would determine to which kind of a file type it belongs. In other words, the program would recognize which type of audio file is the input file (for example speech, music, etc). This program might have a multiple applications such as consumer phone applications that would determine the name of a song that you play in the application or a system that would recognize the voice of an owner of some sort of product they are trying to access. Many modifications would need to be undertaken in order for the program to be production ready.

Background

Voice recognition and content based audio retrieval has many different applications in the modern world so it is a very interesting topic to research. Besides the basic consumer applications such as those that are used on mobile phone (Siri on iPhone or applications that recognize songs based on input audio file), voice recognition software is used in car systems, telephony, and military.

All of the software implementations for speech recognition and audio retrieval follow the common basic pattern:

- Feature Extraction (MFCC)
- Training of the system
- Feature Matching
- Retrieving that audio files

We made a basic program in Matlab that demonstrates the basic approach for audio retrieval based on vector quantization and LBG algorithm. In terms of the frameworks and programming languages this project fits well using only Matlab as our main implementation, since it comes along with a number of useful tools and build-in features that will help to assist the project idea towards a good workable prototype. On the other hand, our program uses the outdated algorithm for this particular area or research so it has a poor performance compared to the modern programs that use different, faster, and more accurate approaches.

The majority of modern voice recognition software tools use the Hidden Markov Models approach in their implementation which has proved to be more effective for this particular task. We did try to implement the HMM algorithm in our application due to the time constraints and decided to instead learn the vector quantization approach. Vector quantization is widely used in data compression, image and video processing, pattern recognition, and as clustering algorithm in general so the knowledge that we learned can be applied to other areas.

Techniques

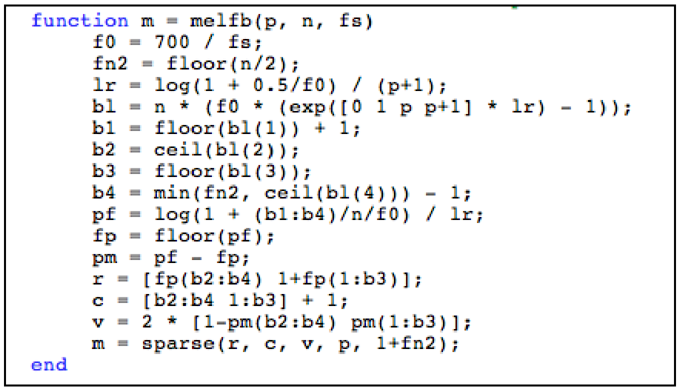
The initial step in this program which handles content audio retrieval is feature extraction. An example of this process in depth would be to identify the components of a wav audio signal that are good for identifying the linguistic content and discarding all the other unnecessary stuff which carries information such as background noise.

1. Frame the audio signal into short blocks of frames. (25ms per frame is standard)
2. Calculate the power spectrum for each frame acquired by computing its periodogram estimate.
3. Apply the mel filterbank to the power spectra, sum the energy in each filter.
4. Take the logarithm of all filterbank energies.
5. Take the Discrete Cosine Transform (DCT) of the log filterbank energies.
6. Keep DCT coefficients 2-13, discard the rest.

The Mel scale relates perceived frequency, or pitch, of a pure tone to its actual measured frequency. Example of the formula used can be found in figure 2(a)

(a) $M(f) = 1125 \ln(1 + f/700)$

(b)



```
function m = melfb(p, n, fs)
     f0 = 700 / fs;
     fn2 = floor(n/2);
     lr = log(1 + 0.5/f0) / (p+1);
     bl = n * (f0 * (exp([0 1 p p+1] * lr) - 1));
     b1 = floor(bl(1)) + 1;
     b2 = ceil(bl(2));
     b3 = floor(bl(3));
     b4 = min(fn2, ceil(bl(4))) - 1;
     pf = log(1 + (b1:b4)/n/f0) / lr;
     fp = floor(pf);
     pm = pf - fp;
     r = [fp(b2:b4) 1+fp(1:b3)];
     c = [b2:b4 1:b3] + 1;
     v = 2 * [1-pm(b2:b4) pm(1:b3)];
     m = sparse(r, c, v, p, 1+fn2);
end
```

FIGURE 1 : (a) Frequency to Mel Scale formulas (b) Matlab version (implemented by Minh N. Do, http://minhdo.ece.illinois.edu/teaching/speaker_recognition/code/melfb.m)

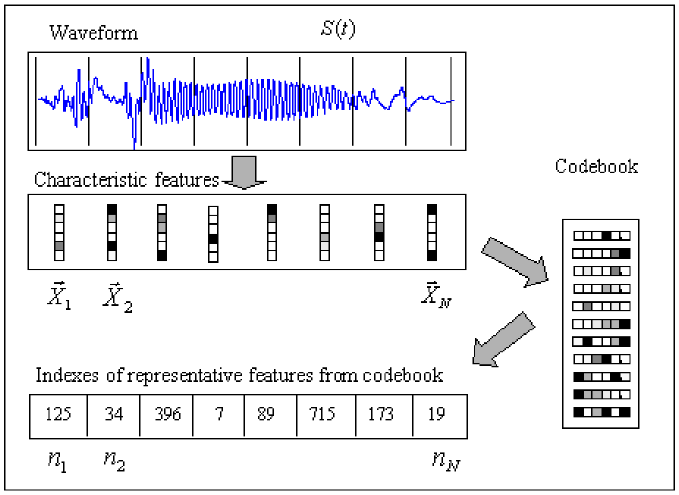


FIGURE 2: The process from an audio signal to a codebook representative

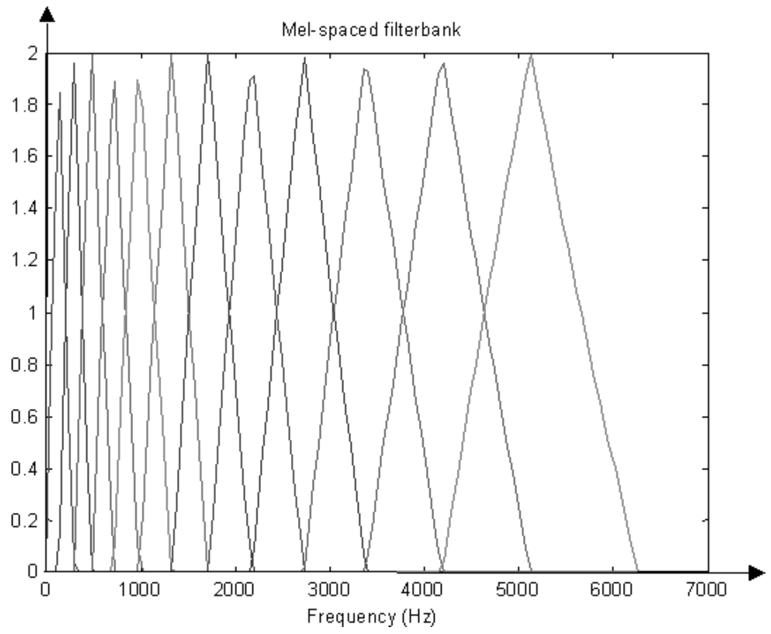


FIGURE 3: Mel-spaced filter-bank with 20 filters

We use the LBG algorithm for clustering and performing vector quantization. The algorithms is fairly simple and widely used for many different purposes.

In order to understand how the algorithm works we need to introduce few definition. **Vector quantization** is the process of mapping vectors from a large vector to a finite number of regions called clusters (4). Each center of a cluster called **codebook** or **centroid** (4). "The collection of all codewords is called a **codebook** " (4).

The algorithm is basically implemented by doing the following steps:

1) Get the 1-vector codebook by finding the mean value

2) Split each current codebook y_n according to the rule:

$$y_n^+ = y_n(1+\varepsilon) \qquad y_n^- = y_n(1-\varepsilon)$$

where 'ε' is the splitting parameter in our program it is 0.03. In this step, we simply multiply the codebook by (1+ε) and (1-ε) thus making it twice larger

3) Nearest-Neighbor Search: for each training vector, find the codeword in the current codebook that is closest (based on Euclidean distance) and associate with the nearest centroid.

4) "Centroid Update: update the codeword in each cell using the centroid of the training vectors assigned to that cell" (4).

5) "Iteration 1: repeat steps 3 and 4 until the average distance falls below a preset threshold" (4)

6) "Iteration 2: repeat steps 2, 3 and 4 until a codebook size of M is designed" (4).

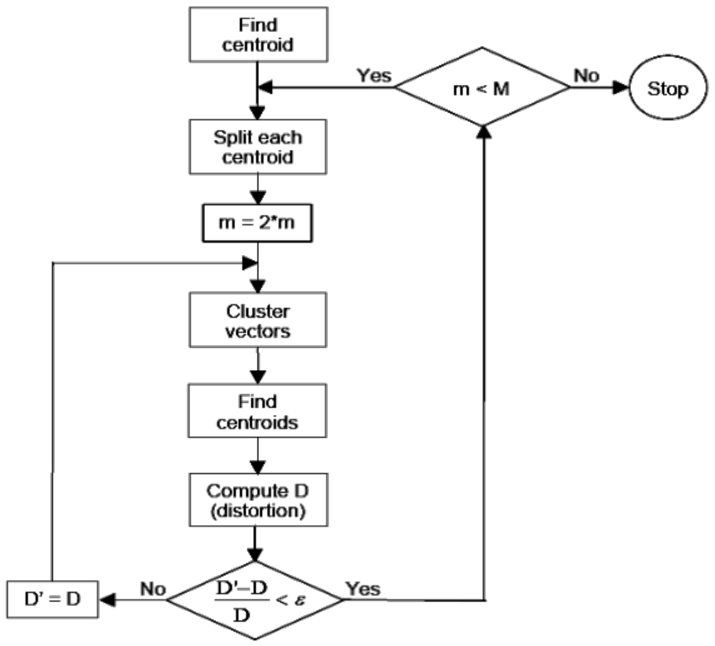


FIGURE 4 : The flowchart for the LBG algorithm

We used Euclidean distance formula in order to compute the distances within the LBG algorithm and between the codebooks in data matching.

$$\mathrm{d}(\mathbf{p},\mathbf{q}) = \mathrm{d}(\mathbf{q},\mathbf{p}) = \sqrt{(q_1-p_1)^2 + (q_2-p_2)^2 + \cdots + (q_n-p_n)^2}$$

$$\sqrt{\sum_{i=1}^{n}(q_i-p_i)^2}$$

FIGURE 5: Formula for Euclidean distance

This is the simplest method of finding a distance between two vectors but there are many other ways to do that.

Design

The program consists out of three matlab files. The main file is the runner of all of the other parts of the program. The VRClass file contains the class with all of the methods that compute the mel-frequency cepstrum coefficients, vector quantization, and distances. The melfb file contains the code that computes filter bank amplitudes (this file was made by Minh N. Do and we are simply using it). VRClass is the heart of the program that contains the number of static methods mfcc, makeCodebook, and computAndPrint which do all the work.

The findEuclideanDistance method determines the Euclidean distance between the two input vectors. The function accepts two vectors between which we are trying to find the distance. First, the function find the size of each input vector and if the number of rows in both vectors is the same then it performs the other operations. Then it creates the distance vector where we would store the resulting distances that the function returns. After that we loop through each element of both array and find a Euclidean distance between them. We store the resulting distances in the distance vector and return it.

The mfcc method computes the mel-frequency cepstrum coefficients. It accepts an input file data that you can generate by reading an audiofile via audioread. After, the method received the input data it splits it into different samples and frames based on the variables samplesPerFrame (which the number of sample per frame) and distanceBtwBeginning (which is the distance between beginning of each frame). Then it applies the window function to the resulting matrix that contains the data split into frames in order to the spectrum matrix. Next it calls the melfb function to compute the amplitudes of the filter bank based on the numberOfFilters (which is the number of filters that would be applied during mel-frequency wrapping step) variable. Finally, after getting the amplitudes it gets the MFCC coefficients by finding a discrete cosine transform and returns the result.

The makeCodebook function creates a codebook using the vector quantization algorithm. It accepts the vector of training vectors and the variable with number of centroids that would be used in the algorithm. Then it splits the initial codebook based on the LBG formulas multiply each element by 1 + split parameter (that we set for the LBG algorithm to work) and 1 - split parameter and adds all of the resulting variables to a new codebook and thus making it twice larger. After doing that the algorithm finds a distance between each codeword and calculates the centroid for each of the training vectors. Then the algorithm updates the centroids and calculates the distortion for each of the clusters that were formed. It updates the centroids until the (updated distortion - current distortion)/current distortion is less than split parameter.

The computeAndPrint function gets the name of the input file that we are trying to compare, the vector with file name for the codebook creation, the vector with the labels that correspond to the files for the codebook, the threshold that is going to be used for determining the match, and the number of centroids for clusters that would be calculated using the LBG algorithm. This function uses the all the other function in this class for calculating the Euclidean distance, figuring out mel-frequency cepstrum coefficients, and generating the codebook.

First, this function reads the input audio file and finds out the MFCC coefficients for it. Then the function creates the codebook for this single file so we can find the distance to the other files in the library later. Next, we loop through the array of filenames and read audio files, find their MFCC coefficients, and add all of them to the codebook.

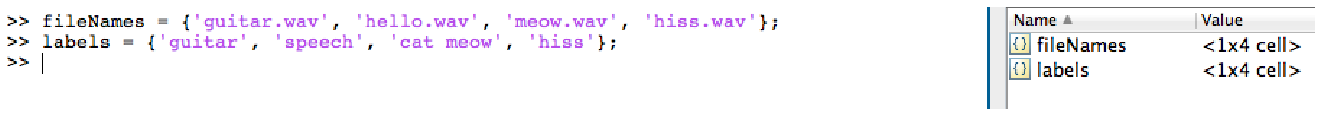
After finding all of this information about the audio files we can calculate the distance between the input file and the audio files in our library. We achieve that by finding the distance between each codeword in our codebook and the query input file codeword/codebook. We store the results in the matrix that contains all of the distance which is called distanceMatrix. Then we look for the minimum distance in the distanceMatrix so that we know which codewords have the minimum distance between each other.

Finally, we check if the minimum distance that we found out is within the threshold parameter that we passed into the function. If it is within the threshold then we found the index of the codeword from our library that is the closest match to the query input audio file. If the minimum distance is larger than the threshold then there is no matching codeword in our library and thus the query audio file is unrecognized.

User Guide

The entire program is made in MATLAB so the user would need to install it in order to run the program. There are two ways to run the program. The simplest way is to use the computeAndPrint function from the VRClass class.

1) Create a vector that would hold the names of the audio files (that would be used for training the system and creating the codebook) and vector that would hold the corresponding labels to those file

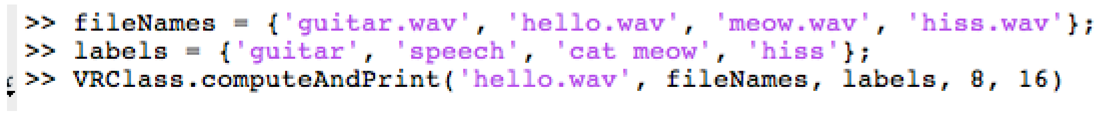


```
>> fileNames = {'guitar.wav', 'hello.wav', 'meow.wav', 'hiss.wav'};
>> labels = {'guitar', 'speech', 'cat meow', 'hiss'};
>> |
```

Name ▲	Value
fileNames	<1x4 cell>
labels	<1x4 cell>

FIGURE 6: Setting up the file names and labels vectors

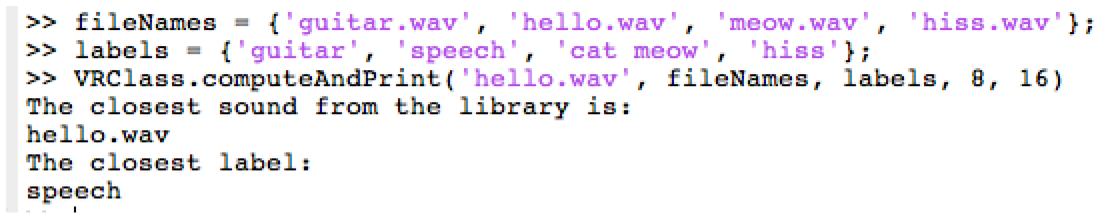
2) Call the VRClass.computeAndPrint function with arguments there are the input query file name, the vectors which were constructed in the previous step, the number for the threshold, and the number of centroids that would be used in the LGB algorithm



```
>> fileNames = {'guitar.wav', 'hello.wav', 'meow.wav', 'hiss.wav'};
>> labels = {'guitar', 'speech', 'cat meow', 'hiss'};
>> VRClass.computeAndPrint('hello.wav', fileNames, labels, 8, 16)
```

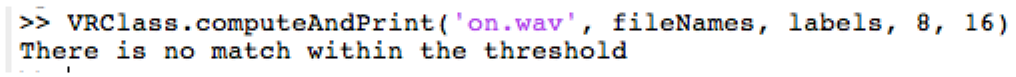
FIGURE 7: Running the computeAndPrint static function

3) Run the function and it would return the closest match to the query audio file within the threshold or would display the the message that there is no match for the query



```
>> fileNames = {'guitar.wav', 'hello.wav', 'meow.wav', 'hiss.wav'};
>> labels = {'guitar', 'speech', 'cat meow', 'hiss'};
>> VRClass.computeAndPrint('hello.wav', fileNames, labels, 8, 16)
The closest sound from the library is:
hello.wav
The closest label:
speech
```

FIGURE 8: Example of the results from running the program when the query is found



```
>> VRClass.computeAndPrint('on.wav', fileNames, labels, 8, 16)
There is no match within the threshold
```

FIGURE 9: Example of the results from running the program when the query is not found

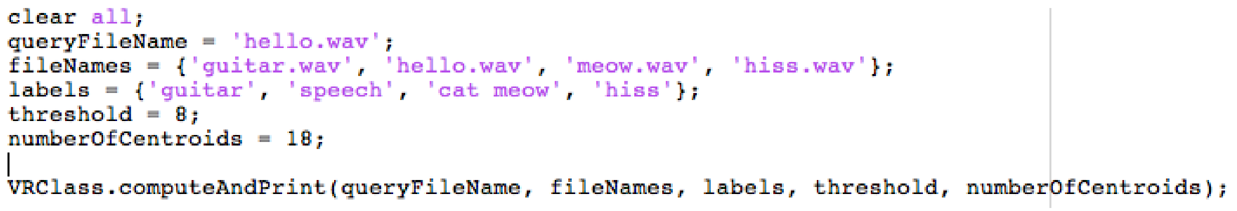
Another way to use the program is by running the main file but in order to do that you would need to modify (or not depending on the purpose) the vectors with names of the audio files and labels as well as the name of the query file (if necessary) if you want to the program with your own training files.

1) Put the audio files in the same folder where the program files and the current Matlab path is

2) Modify the fileNames and labels vectors accordingly so that it contains the names and labels of the training files that you want to use for the codebook

3) Change the value of the variable queryFileName so that it contains the name of the query audio file that you want to process

4) Change the values of the threshold and numberOfCentroids variables so that they store the desired numbers for your search

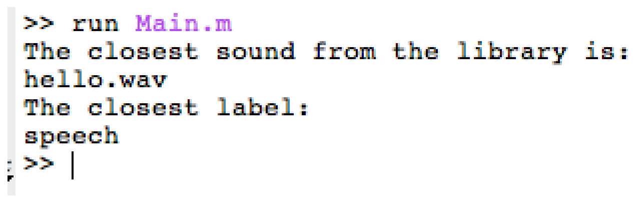


```
clear all;
queryFileName = 'hello.wav';
fileNames = {'guitar.wav', 'hello.wav', 'meow.wav', 'hiss.wav'};
labels = {'guitar', 'speech', 'cat meow', 'hiss'};
threshold = 8;
numberOfCentroids = 18;

VRClass.computeAndPrint(queryFileName, fileNames, labels, threshold, numberOfCentroids);
```

FIGURE 10: Example of the Main.m file with all of the variables set up

5) Run the Main.m file (you can run Main.m right away because it has all of the variables set for the example based on the audio files that are in the same folder as the program files) and the program would return the message with the closest match if there is any in the desired threshold



```
>> run Main.m
The closest sound from the library is:
hello.wav
The closest label:
speech
>>
```

FIGURE 11: Example of the results of running Main.m file when the query is found

Alternative Approaches

Hidden Markov Models (HMM) aka “Markov Chains” is would be a possible way to handle the continues amount of sequential data better than vector quantization during the process of feature matching. HMM is a state-of-the-art feature matching technique compared to what we are using now. VQ is in fact easier to implement contrast to its advance equivalent.

The approach utilizing HMM for our current prototype with Matlab shouldn’t be an issue for future use if this project were to expand any further. Hidden Markov models are best known for identifying temporal pattern recognition like audio signals such as vocal speech and musical score that may following. The outstanding accomplishments using this advance method will gain a higher with accuracy in the long run. Some examples using HMM can be demonstrated as such in figure 4 as an example how each frame has its representation from the generated spectrum, it is generally faster to location a series that matches and recognizes the pattern of a any given input of signal. So long as there exists a similar signal, there will be an output that can be verified.

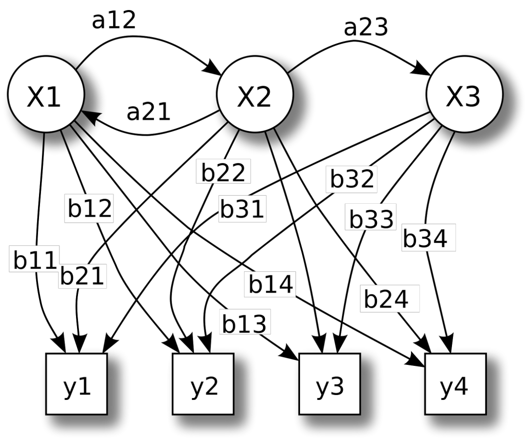


FIGURE 12: Hidden Markov Model (*x* - states, *y* - possible observations, *a* - state transition probabilities, and *b* - output probabilities)

Besides HMM different approaches can be taken when figuring out the mel-frequency cepstrum coefficients. Instead of using Hamming window formula for avoiding the power leakage we can use Kaiser or Kaiser-Bessel window function. This function

$$w_n = \begin{cases} \frac{I_0\left(\alpha\sqrt{1-\left(\frac{2n}{N}-1\right)^2}\right)}{I_0(\alpha)} & \text{if } 0 \le n \le N \\ 0 & \text{otherwise} \end{cases}$$

FIGURE 13: Formula for Kaiser window function

In the formula above *N* is the length of the sequence. I_0 is the zeroth order Modified Bessel function of the first kind, which is then defined as in the figure below.

$$x^2\frac{d^2y}{dx^2} + x\frac{dy}{dx} + (x^2 - \alpha^2)y = 0$$

FIGURE 14: Bessel functions are the solutions to this equation with arbitrary alpha

Bessel function is the harmonic function which is important in many problems with wave propagation and others further explanation of this function is beyond the scope of the project. In the Kaiser window function the higher the alpha the narrower gets the window and therefore the truncation is not as severe as in Hamming window function.

We also could have used a different formula for computing the distances between vectors such as Manhattan distance or Bray-Curtis distance.

$$d_1(p,q) = |p-q|_1 = \sum_{i=1}^{n} |p_i - q_i|$$

FIGURE 15: Formula for Manhattan distance

$$d = \frac{\sum |x_i - y_i|}{\sum (x_i + y_i)},$$

FIGURE 16: Formula for Bray-Curtis distance

Limitations

Currently our prototype MATLAB program doesn't have a user interface so it drastically limits potential users if they don't have a basic knowledge of programming in Matlab. We didn't have time to implement the user interface (partly because of poor knowledge of Mathlab) especially considering the fact that the basic purpose of the project is to learn the concepts of the signal processing.

Another drawback of the program is that it is relatively slow compared to similar professional software products because it uses vector quantization technique and LBG algorithm for creating the codebook. There are multiple improvements to the algorithm that could have been implemented in order to boost performance of the program.

One of the ways of improving LBG algorithm would be using a utility measure. The classic LBG algorithm has a problem where a lot of elements in a large cluster are badly approximated. So, the improvement of the algorithm would be finding a way to split large clusters into smaller ones based on a utility measure thus minimizing distortion in a cluster. In order to do that we would need to compute the mean quantization distortion over all cells Dm = 1/N*(Sum of all distortions), where N is the number of cells in the training vector. Then we would assign a utility measure U for each cell where Ui = Di/Dm, where Di is the distortion for a given cell. Then improved algorithm would perform the equalization of cells by joining the cells with the low utility index with adjacent cells and splitting the cells with high utility into smaller cells. This approach would lower the distortion.

The program doesn't compute the weighted Euclidean distances. The application of the weights to the distance would reduce the chance of getting outliers in the distance matrix that determines the match and when creating a codebook.

The program can only store a limited amount of data about the training audio files because it simply uses vectors for storage. In other words, the amount of memory that user can utilize strictly depends on the user's machine so that stack overflow is possible.

Problems

It was difficult to figure out how to compute the amplitudes of the mel filter bank for the spectrum matrix (which we get from splitting the audio file into frames and applying the Hamming window function) while computing the mel-frequency cepstrum coefficients for a given input file. We couldn't solve this problem by ourselves but we found the function which does that and was already implemented by Minh N. Do (an instructor at University of Illinois) so we used it in our code.

The LBG algorithm for creating the codebook for our program in createCodebook function was very difficult to figure out. We had to read a lot of materials and learn a bit of Matlab in order to understand and implement it. We used the code and explanation by Minh N. Do in his little project as a basic template for implementing this algorithm.

Another problem was that there was not enough time to learn how to create a user interface using Matlab. We didn't solve this problem so there is no user interface. A possible solution for this problem could have been implementing a bash script that would call the Matlab to execute a certain scripts based on the input from the console (or command prompt) if we want to use a way of interaction with a user. Also, we could have simply learn how to create GUI using Matlab or convert the code into C++ and implement user interface in it. In addition, because Matlab can use a lot of Java libraries we could possibly make a user interface in Java.

Future Work

For any future work and features that might be uniquing feasible as an additional extension to our prototype. One addition would be to add a microphone audio sample subroutine that takes in a sampling snippet of audio. A key feature with this postulation would be ideal to generate an audio database or larger codebook structure. An ability to add new files to the library without modifying the program code.

The algorithm for creating the codebook can be drastically improved by lowering the distortion as mentioned above in the limitation section and also by using classified pre-selection which would speed up the algorithm. Another way of extending this particular feature would be completely replacing the LBG algorithm and using a different one.

The project could have had a user interface; however, there was only a limited amount of time of four weeks to throw together a bare prototype in Matlab. As mentioned before it can be implemented in Java, Matlab, C++, or simply by using a command prompt and bash scripting.

One of the most important extensions for this program would be a database. The data about the training files (such as codewords or mel-frequency cepstrum coefficients) can be stored in the database so that the user would have a large amount of data for comparison. It would greatly increase the accuracy of the prediction or would return more matching files. Another approach for this program could be to make it so the program would be independent from data.

In other words, it can have a separate module that would allow hooking up different types of databases and perform the queries but it would be too difficult.

References

1) Foote, Jonathan T. "Content-based retrieval of music and audio." *Voice, Video, and Data Communications* 6 Oct. 1997: 138-147.

2) Niewiadomy, Dominik, and Adam Pelikant. "Implementation of MFCC vector generation in classification context." *J. Applied Computer Science* 16.2 (2008): 55-65.

3) Niewiadomy, Dominik, and Adam Pelikant. "Digital speech signal parameterization by Mel Frequency Cepstral Coefficients and word boundaries." *Journal of Applied Computer Science* 15.2 (2007): 71-81.

4) Kumar, Ch Srinivasa, and P. Mallikarjuna Rao. "Design of an automatic speaker recognition system using MFCC, Vector quantization and LBG algorithm." *Ch. Srinivasa Kumar et al./International Journal on Computer Science and Engineering (IJCSE)* 3.8 (2011).

5) "Vector Quantization." *Vector Quantization*. N.p., n.d. Web. 30 Nov. 2014. <http://www.mqasem.net/vectorquantization/vq.html>.

6) Lartillot, Olivier, and Petri Toiviainen. "A Matlab toolbox for musical feature extraction from audio." *International Conference on Digital Audio Effects* 10 Sep. 2007: 237-244.

7) Sitiber, Michael, and Billin Zhang Stiber. “Signal Computing:Digital Signals in the Software Domain.” 2014. 12 <http://faculty.washington.edu/stiber/pubs/Signal-Computing/Signal%20Computing.pdf>