Spoken Language Identifier
18-551 Final Written Report

Spring 2011, Group 4
Bridget Liao (bliao@andrew.cmu.edu)
Jinyoung Park (jinyoung@andrew.cmu.edu)
Eric Turner (elturner@andrew.cmu.edu)

Introduction

The Problem

There are many applications in industry that have a demand for automatic language recognition. One such application is the use of Language Line Services. As discussed in the Marc Zissman's Comparison of Four Approaches to Automatic Language Identification of Telephone Speech [9], these services will distinguish the language being spoken by a telephone caller, and redirect the call to a representative who is fluent in that language. Most of these services are done manually, where human receptionists attempt to distinguish the language being spoken through trial and error. The introduction of an automatic system could reduce overhead time, as well as error rates for these systems.

Another likely application would be to combine a language recognition system with automatic real-time speech translation. Many services like Babelfish or Google Translate have been released that allow for text-to-text translation, but recently these services are being adapted for real-time speech-to-speech translation. While the accuracy of these systems continues to be improved, a permanent shortcoming is the need to know the input language. For many situations the input language is known, but there are occasions when one wishes to translate speech of an unknown language. A system that is able to identify the language spoken and then feed the signal to a specific translator would be able to act as a universal translation device, something that has only been present only in science fiction.

Evolution of Project

During the developmental stages of our project, we first devoted most of our time reading materials about previous research and projects on speech processing and recognition that had been done in previous years. We found background on different techniques on speech recognition in discussions with Professor Richard Stern because of his knowledge on signal processing and speech manipulation and he pointed us to different research papers that he thought would be useful to us. In addition, he allowed us access to databases of speech that we could later use while training and testing our system.

In addition to these discussions, we developed much of our system breakdown from reading Lee's chapter on the principles of spoken language recognition in the *Springer Handbook of Speech Processing* [1]. From this, we made decisions on what algorithms to use and how each one could be implemented. We decided against using word recognition because of the huge amounts of data that would have to be stored and our limitations on the DSK, and instead decided on implementing our system based on phoneme recognition and classification.

With all the algorithm choices in mind, we initially decided on splitting our system into two different sections, one for phoneme training and identification, one for the actual language training and identification; both of which will be discussed further later on. We created a

schedule of what we would implement and tried to follow these deadlines throughout the duration of the project. A few changes to our algorithms and implementations were made because of time constraints, but the overall system breakdown was still the same.

Most of the code in our system was implemented by our group members, however, we did use some third party code for certain sections of our projects such as the Hidden Markov Model based on advice about the difficulty of the implementation by Professor Stern and because of limits on time and capabilities. We initially planned on using speech databases from CMU's Language Technology Institute because we felt that it would have contained a diverse set of speech samples of each language; however, because of some issues with noise in the database samples, we decided instead to use audio books of different languages found online that had very clean speech and minimal noise in the background [3].

We originally wanted to use the DSK to compute much of our front-end phoneme classification implementation. However, we ran into network problems sending data packets from the PC to the DSK and even after extended attempts to fix the problem and asking the TAs about what could be wrong, we could not find the nature of the problem. If we had more time to work on the project, we would have spent more time debugging the problem with the hardware. We finally made a decision not to use the DSK before we had our stand-alone code on the PC working as a language classifier.

Each group member worked on a separate language classifier that would take in either a raw speech file or a text file of phoneme values of a sample of speech. We trained our language classification system on several gigabytes of raw speech that included multiple speakers of different gender and different chapters of audio books, thus ensuring a diverse training set that would be unbiased to a certain speaker or accent.

Finally, we consolidated our code together into what we had initially envisioned as our spoken language identifier. There were many issues that were addressed during the entire project process, and these obstacles are discussed later.

Background

Ten years ago, speech processing and techniques were still being developed and not many research projects like ours could be implemented. However, speech processing is now an established field that has a wide range of applications. Although the output signal or final decision of different speech systems can vary widely, all systems start by characterizing important features within the speech signal. It is important to enumerate which of these characteristics are useful and which are not. Oftentimes, being able to ignore unwanted characteristics is just as challenging as isolating the features that we are aiming to utilize.

As described by Harper and Maxwell in their chapter in the *Springer Handbook of Speech Processing* [8], some of the characteristics to ignore include:

- Individual speaking style
- Variations in speaking rate
- The speaker's emotional state
- Mispronunciations

Another significant factor is the genre of speech. In most languages, formal speech sounds very different than casual speech or slang. Dialect and difference of certain vocabularies from different regions can also affect the language selection.

Next, it is important to establish the features in speech that are deemed useful in language identification. Since speech samples in the time domain are difficult to interpret, it is useful in many speech applications to convert a speech sample from the time domain into a phoneme sequence.

Phonemes are the smallest distinguishable units in speech. A single phoneme lasts on the order of milliseconds in speech, and is the symptom of a particular configuration of the speaker's throat, mouth, tongue, teeth, and so forth. In C.-H. Lee's chapter in the *Springer Handbook* [3], phonemes are described as the manageable set of spoken letters. By deconstructing speech into a series of these fundamental units, an analysis of the important speech content is related to the field of Information Retrieval.

Another important characteristic is that each phoneme can be modeled as a stationary random process, which means that there are several ways to model the characteristics of a phoneme sample. In our project, we took the Mel-Frequency Cepstral Coefficients, which are less affected by quantization effects, and provide an accurate model of noiseless speech. However, there is an inherent trade off when choosing the order of the model. Naturally, a smaller order model will give reduced distinction between different phonemes, while a large model requires more samples of a particular phoneme snippet, which means either allowing higher variance in auto-correlation coefficients, or increasing the window size, which may invalidate the assumption of stationarity. These factors have lead to an industry standard of taking 20 millisecond snippets to represent a single phoneme, where they can overlap by some amount.

By deconstructing speech into sequences of phonemes, it is possible to distinguish between languages. Each language has a phoneme set that represents all possible sounds within that language. In some cases, elements of these phoneme sets can be distinct from other languages. In such situations, the detection of one of these unique phonemes allows for accurate identification of the language. In most cases, the phoneme sets of languages overlap, and a judgment can be made based on the relative frequencies of phonemes in each language. A system that deals with multiple languages must be able to distinguish each element in the union of all the languages' phoneme sets.

Using these phoneme sequences, a language identifier could test for the frequency of strings of phonemes in a speech sample, and match the result to the relative frequencies of those strings in its known languages. In addition to simple concatenation of phonemes, some systems also

parse semantic words from the speech samples. This process depends heavily on knowledge of the language being identified, so our group decided to classify using only models developed from phonemes.

Current State of Research

In terms of research, language identification has become a very energized field in the last eight years. Many competitions are held each year, testing the state of the technology, including the Language Recognition Evaluation (LRE) held by NIST. The 2009 results [10] show one of the latest records of the capabilities of modern algorithms, tested with libraries of 23 target languages as well as additional languages undisclosed to participants. The competitors were tasked with identifying the input language from 30-second, 10-second, and 3-second samples. The best of the competition were able to accomplish this task with under 2% error rate for 30-second clips and under 10% error for 3-second clips.

The typical modern algorithms can be broken into five subsystems. The first of these is phoneme tokenization; common algorithms for this step are Linear Prediction Coding (LPC) or MFCC. The next subsystem is the phoneme code-book creation algorithm. The coefficients generated from a phoneme will vary from one sample to the next, so the system needs a canonical representation of each phoneme, and a way of mapping each possible phoneme to the canonical version. Algorithms used to develop these mappings are called Acoustic Segment Models [2]. Traditional algorithms for segmenting the phoneme space include the Generalized Lloyd Algorithm [7], or other clustering algorithms like Gaussian Mixture Modeling [11]. Most modern approaches use variations on Hidden Markov Modeling (HMM). Two such algorithms are the Universal Voice Tokenization (UVT) and the Parallel Voice Tokenization (PVT) described in the chapter of the *Springer Handbook* by H. Li, B. Ma, and C.-H. Lee [2].

The next subsystem performs vectorization of tokenized phonemes. When a speech signal is read in, it is first parsed into phoneme coefficients, and then those coefficients are matched to the phoneme code-book. Vectorization is the algorithm used to match to the code-book. The simplest such algorithm is Vector Quantization (VQ), which just takes the Euclidean distance of the input vector, compared to each vector in the code-book, and selects the closest one. Other routines include Dynamic Time Warping as well as another application of HMM [7]. Using a more effective code-book generation will lead to better classification, so this is a likely point to determine a trade-off between error minimization and efficiency.

Last, and most importantly, is the Vector-Based Classifier. These are the subsystems for training and classifying languages based on their phoneme vector representations. There are many possible algorithms for this type of classification. One of the simplest choices is the bag-of-sounds concept, where the most frequent phonemes are measured from both the sample speech, and each of the known languages, and these lists of most frequent phonemes are compared. This process can also be done with pairs of sequential phonemes, or even trigrams. Alternatively, the Support Vector Machine is common among many contemporary systems [2].

In many of these systems, we can consider the Bayesian probability of a given choice. In this case, the overall probability of observing a language needs to be considered. In actual applications, this could be very important. For instance, the probability of observing Swahili in Pittsburgh is very low, so before reporting it, a system should be absolutely certain. However for the laboratory or academic setting, these probabilities would reduce accuracy, since the system is likely to encounter a more diverse set of languages during testing.

Previous Projects

One group during the Spring 2002 semester (Group 5) of 18-551 also implemented a spoken language identifier meant to distinguish between English, Spanish, and French. However, their project implemented only a single probability matrix per language to determine a result It aimed to store each language's characteristics within the transition probabilities between two consecutive phonemes.

For the front-end of their system, they used Mel Frequency Cepstrum Coefficients (MFCC) to tokenize their phonemes, and Vector Quantization (VQ) to vectorize these phonemes, using the LBG algorithm to separate their sample speech into 128 phonemes. Their training data comprised of each group member reading about 30 minutes of news into the system in three different languages: English, French, and Spanish. They built a 128-entry phoneme codebook from their training set and created probability matrices for each language by determining the probability of each successor phoneme in each language matrix. They then tested their system with 30-second samples, comparing the sum of probabilities of each language matrix to determine the best language match.

Their results indicated that their system tended to misclassify group members' speech as their own native language, regardless of the actual language spoken. In addition, during their demonstration, other native speakers of English would consistently classify to English, and non-native speakers would mostly classify to their respective native language. In the end, their program seemed to perform accent identification more than language identification, which, even though the group reported as a success, did not solve the initial problem that they looked at.

What's New

We made several changes from the previous things that have been done in the group project from 2002. The most important change was that we increased the amount of training data that was used. One hypothesis for why the previous project's system failed as a language identifier was because of the strictness of their training set. We broadened the range of our speech data by using audio books instead of recordings of only group members as the previous group did. By using audio books, we ensured a relatively large range of vocabulary for each language, varied speakers, as well as large amounts of clean speech that is ideal for training, which in turn, lowers the strictness in judgment for the language classifiers.

Another change that we made was in the decision system component. Given that phoneme sampling from speech typically underestimates the length of a single phoneme, the result is repetitions of the same phoneme abutting each other. This would mean that the most probabilistic successor phoneme to an utterance would be the same phoneme, for any language. This obfuscates the differences between languages, thus reducing the effectiveness of their system. In our implementation, we used an Ensemble Decision Strategy [2]. We implemented three different methods of feature mapping to determine the language, all which are explained in detail later on in this paper. This allowed us to utilize different approaches to identify a language and minimize possible errors that could result from relying only on a single algorithm. The final result is a combination of the three identifiers weighted equally.

Algorithm Choices and Design

Speech classification relies on many principles from pattern recognition, in which elements of the sample space of observations (in this case raw speech) are transformed into a feature vector, which can then be classified. Traditionally in speech processing applications, a useful transformation is to convert raw speech into a phoneme sequence, which can then be used to make up the features extracted from that speech [2]. This classification is referred to as the front-end of our program, which is detailed in later sections.

Once the speech is transformed into a sequence of phonemes, it is necessary to perform a statistical analysis on this sequence, and compare its characteristics to those of known languages. To accomplish this, an Ensemble Decision Strategy was chosen, which combines several different classification algorithms into a signal language ranking [2]. This allows simple classification algorithms to be used, whose individual performance may be sub-par, but where the average of the classification of all of them would give a more accurate result. Discussed below are the specific algorithms used to form this ensemble.

Speech Segmentation

As discussed above, raw speech must be converted into phonemes in order to be classified. Phonemes are small segments of speech, which are assumed to be a stationary process. Each phoneme represents a single sound utterance, which can be combined to form syllables, words, sentences, etc. [1]

For this project, raw speech is assumed to be sampled at 16 kHz. This sampling rate allows full information retention of the speech signal, which typically require lower sampling rates than more complex signals, such as music. A phoneme is considered to be a speech segment of 20 milliseconds (or 320 samples). In order to reduce the likelihood of missing a phoneme, due to capturing a transition, each speech segment overlaps by 160 samples (50%).

Thus, after segmentation, speech is represented by a sequence of vectors, each of which is 320 samples in length. Since each segment will be analyzed for frequency content, they are

multiplied by a 320-point hamming window, in order to reduce frequency blurring.

Mel-Frequency Cepstral Coefficients

To perform a phoneme classification, the input speech must be reduced to contain only information unique to the sound associated with the distinguished phonemes. The classification process should ignore many aspects of speech that do not relate to the syntax of what is being said. For instance, this can include speaking style, speaking rate, gender of speaker, and channel characteristics [8].

To remove most of these characteristics, but to keep features required for phoneme identification, the Mel-Frequency Cepstral Coefficients of each segment were computed. Other operations could have been performed, such as looking at the LPC (linear predictive coding) coefficients, but MFCCs have been formulated to best model human hearing, and thus keep only the important information.

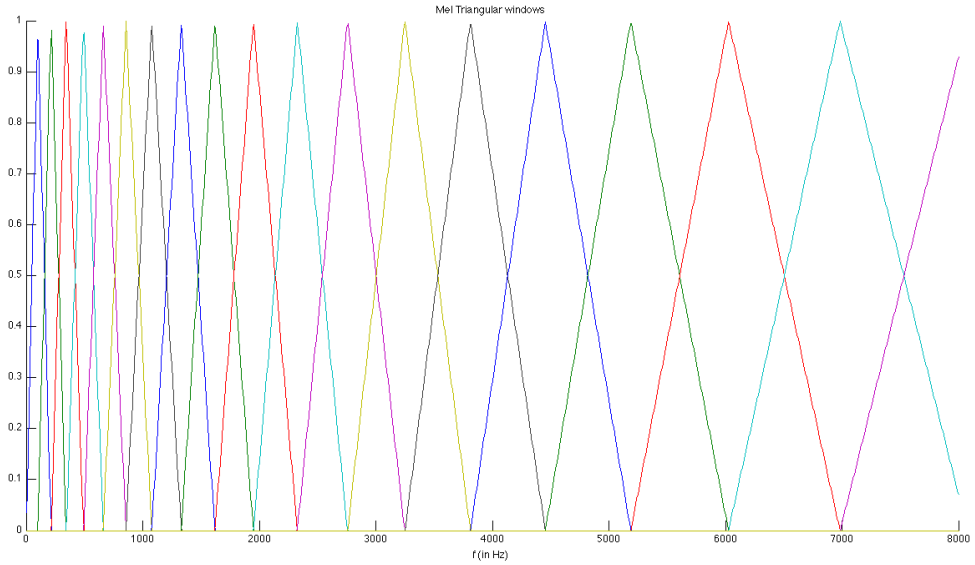
MFCCs have become an industry standard for phoneme classification for noise-less speech. Since this project assumes that the incoming speech does not contain any noise, this algorithm is the obvious choice.

To calculate the Mel-Frequency Cepstral Coefficients, first the Discrete Fourier Transform taken. Since each segment has a length of 320 samples, the 512-point radix-2 FFT is taken, for efficiency of implementation [5].

Next, the frequency spectrum is windowed using triangular windows that are evenly spaced in Mel-frequency. The Mel-frequency spectrum is a non-linear, monotonic transformation from Hz, which accentuates the spacing between lower frequencies. The transformation from Hz to Mels is as follows:

$$m = 1127 + ln(\frac{f}{700} + 1)$$

Thus, evenly spaced triangular windows in the mel domain appear with the following form in the Hz domain. Note that the windows are placed with 50% overlap:



Equi-spaced Triangular Windows in Mel Domain

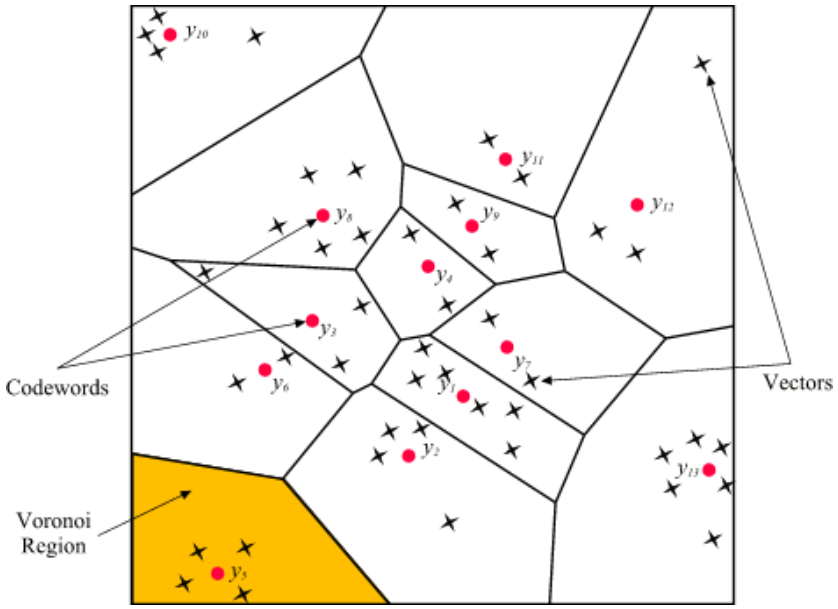
Note that due to the Nyquist frequency of 8000 Hz, the windows are placed so that the left edge of the first window occurs at 0 Hz, and the right edge of the last window occurs at 8000 Hz. In order to provide sufficient resolution, 32 windows were used.

Each window is applied to the frequency domain of the speech segment, and then the log of the power of the result is computed. This process results in 32 values that represent the content at each frequency band defined by the windows above.

This sequence of 32 values was then passed through a Discrete Cosine Transform. The result of this transform is referred to as the Mel-Frequency Cepstral Coefficients vector.

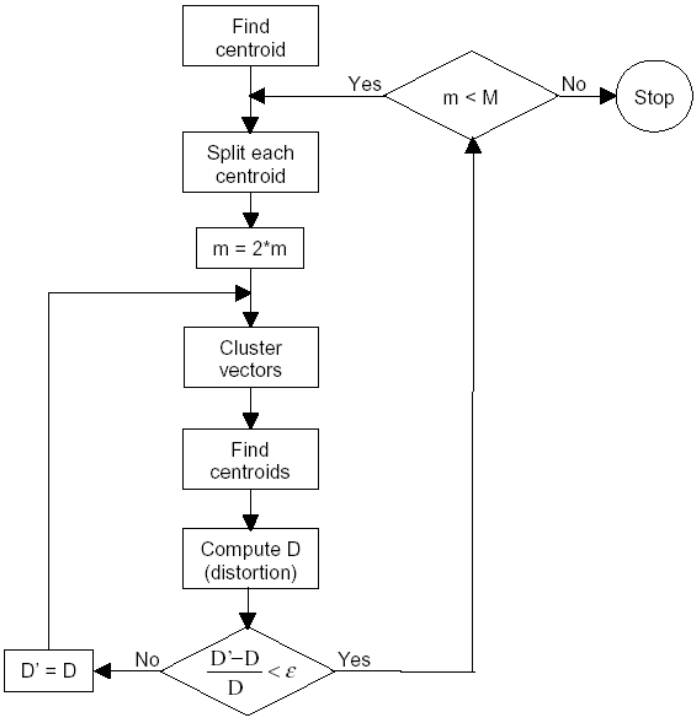
Vector Quantization

When determining what phoneme was observed in a sequence, we need a code-book of codewords to use as a reference, in which a codeword is a standard phoneme. In order to create this code-book we map a large number of observed vectors from a large vector space into a finite number of regions. Each region will represent a phoneme that the vector will be identified with. [7]
The following figure shows an example of such mapping:



Each Voronoi region displays the boundaries within which all vectors are classified to the corresponding codeword (representing a phoneme). Image taken from [7]

To implement this mapping, the Linde-Buzo-Gray algorithm, a derivative of the k-means clustering algorithm, was used. The following flowchart describes the basic idea behind the algorithm:



Flowchart depicting and adapted process of the Linde-Buzo-Gray algorithm for phoneme classification. Image taken from [7]

To expand on how the algorithm works, the algorithm first begins by collecting all the vectors. The first centroid is the mean of all the vectors. The centroid is then split into two centroids, and

the vectors are clustered to either of the new centroids that they are closest to (as determined by Euclidean distances). Then, the new centroids are re-evaluated, becoming the mean of their respective vectors. Afterwards a distortion variable is calculated; this distortion lets us know how much the orientation of the Voronoi diagrams has changed because of this split. If this split did not cause a noticeable change, then we re-cluster the vectors, and evaluate the new centroids as the means of the newly clustered vectors. Otherwise, the centroids are split again and the process repeats, until we have the desired number of centroids.

Despite its simplicity and usefulness, there are some setbacks using this algorithm. Because the LBG algorithm is dependent on a splitting method, the number of centroids, and therefore the number of codewords/phonemes, are limited to a power of two. If we wanted a less restricted number of phonemes, we would have had to use other mapping methods. However, this is a minor limitation, and overall this algorithm seemed to correctly and evenly distribute the vectors into our desired phonemes. We have however encountered one unusual problem, where we believe outliers are affecting our results. Further description is described in our Issues section.

Bag of Sounds Classifier

One of the algorithms used to classify an input phoneme sequence as a language ranking is the bag-of-sounds classification method. In each sequence of phonemes, the most common sub-sequences are stored and compared to known languages [2]. Sub-sequences of length 1 to length 5 were observed, and the most popular 10 sub-sequences of each length were recorded for an input speech.

For example, if the input speech were represented by the following phoneme sequence:

... A B A A C A A A E A D A B A A C F A G A H A ...

The most popular phoneme would be ‘A’. The second-most popular phoneme would be ‘B’. The most popular phoneme pair would be ‘A A’, and then ‘A B’. This analysis can continue for up to sub-strings of five phonemes (which would yield ‘A B A A C’).

When training on a language, sample speech is given, and the 10 most popular sub-sequences of lengths 1 through 5 are stored, along with their frequency numbers (thus 50 pairs of sub-sequences and integers). This process is done for each language.

When testing on unknown speech, the same 50 values are computed as in the training phase, then a ranking is generated for each language by comparing these values computed on the input speech with the values precomputed with that language’s training speech. This comparison is done with three metrics:

- For each length, the intersection of the sets of most popular sub-sequences is determined, and the cardinality of this intersection is added to the ranking for this language. For example, if the test speech had a most-popular phoneme list of

A,B,C,D,E,F,G,H,I,J and a known language had the most-popular phoneme list of B,A,C,Z,Y,F,J,H,E,X then the intersection of these sets would be 7 phonemes. This metric ignores the order of these sets.

- For each sub-sequence within the intersection described above, the relative order compared to other phonemes is considered. If two phonemes appear in the same relative order, then a point is awarded. For example, in the sets described above, the phoneme B is more popular than C in both cases, so a point is awarded. Additionally, C is more popular than E, so another point is awarded, and so forth.
- Finally, the absolute placement of each phoneme sub-sequence in the popularity lists is factored into play. Note that for the test speech, the most popular phoneme is A, whereas for the trained language, A is the second-most popular. Because this is a deviation of only a single place, a point is awarded. For cases where the absolute place is preserved (for instance, C is third-place both times), two points are awarded.

The above scoring is computed for each of the five lengths, and then normalized so that the maximum score possible is 100. The output of the classifier is the ordered list of languages, based on the above scoring scheme.

Probability Matrix Classifier

Another classifier we used was an expansion of the algorithm used by the previous group. The idea of probability matrices was the same, in that we looked at the relative probabilities of seeing one certain phoneme after seeing another certain phoneme. We would count how many times we observe a certain phoneme, and determine what was the likelihood it would show up after phoneme 1, after phoneme 2, etc. [7]

For our classifier, we did not stop with just one probability matrix. We created 3 different probability matrices, each observing a different relationship between phonemes. The first matrix was the same as what the previous group had; it looked at the probability of one phoneme appearing immediately after another. The second and third matrices however, looked at the probability of one phoneme appearing two and three spaces after another, respectively. This was done to improve the accuracy of this classifier, gathering more information from observing the occurrence of phonemes than just one matrix had.

For example, if the input speech were represented by the following phoneme sequence (the same as the example input of the Bag-of-Sounds example):

... A B A A C A A A E A D A B A A C F A G A H A …

then the following would be part of the first matrix generated by the classifier. The first matrix, again, looks at one phoneme and the phoneme that appears immediately afterwards:

phoneme	A	B	C	D	E	F	G	H
A	.3333	.1667	0.1667	.0833	.0833	0	.0833	.0833
B	1	0	0	0	0	0	0	0

The format repeats for the rest of the phonemes, which in this case is C to H. The probability for row i and column j is calculated as the number of times phoneme j was seen immediately after phoneme i, divided by the total number of times phoneme i has another phoneme that comes right afterwards.

The second matrix would look at the phoneme two steps away:

phoneme	A	B	C	D	E	F	G	H
A	0.6667	0	0.1667	0	0.0833	0.0833	0	0
B	1	0	0	0	0	0	0	0

... and so on. The probability of row i, column j is calculated as the number of times phoneme j was seen two spaces after phoneme i, divided by the number of times phoneme i has a phoneme that comes two spaces afterwards.

Likewise, the third matrix, except it looks at the phoneme three steps away:

phoneme	A	B	C	D	E	F	G	H
A	.5455	.0909	0	.0909	.0909	.0909	0	.0909
B	0	0	1	0	0	0	0	0

... and so on.

When training, we created these matrices for each language, and stored them in separate files. Therefore every time we trained a language there were three files created, one for each of the probability matrices.

When testing, we created the three probability matrices for the observed sequence as well. Then, for each language, we loaded up the matrices from saved files, and compared the probabilities. The differences between the observed probability matrices and the saved matrices from training determined the score for the language: higher score was given to the language whose matrices were more similar to the observed matrices.

Hidden Markov Models (HMM) Classifier

The final classifier that we implemented for the system uses Hidden Markov Models. The system calculates a model that has transitions containing the probabilities to one phoneme state from the remaining phonemes, as well as probabilities of each of the emissions from a particular state. After the model is developed, an input of phoneme values is put into a vector which in turn

sums the HMM transition probabilities from each state. In the end, whichever language model has the highest probability will classify as the language being spoken.

For example, let's look at a 3-state observable Markov Model of the weather. An observable model means that we are looking at each physical (observable) event [12]. So let us assume that for one day, the weather can be observed as one of the following:

State 1: rainy
State 2: cloudy
State 3: sunny.

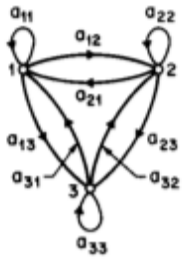
The weather on day *t* can be described using a matrix of state transition probabilities:

	rainy	cloudy	sunny
rainy	0.4	0.3	0.3
cloudy	0.2	0.6	0.2
sunny	0.1	0.1	0.8

This matrix is the transition model. If today the weather is sunny (state 3), the probability of the next 5 days being S = "sunny-sunny-rainy-rainy-cloudy" or {s3, s3, s1, s1, s2} could be evaluated as

P(S | Model) = P(s3, s3, s1, s1, s2 | Model)
= P(s3) * P(s3 | s3) * P(s1 | s3) * P(s1 | s1) * P(s2 | s1)
= pi3 * a33 * a33 * a31 * a11 * a12
= 1 * (0.8)(0.8)(0.1)(0.4)(0.3)
= 0.00768

where pi3 is the initial state probability for the first day, which is given.



An HMM has N states and M distinct observation symbols. In our implementation, we used code written by Tapas Kanungo from the University of Maryland [6]. We had M=128 symbols representing each phoneme from the codebook, and N=4 transition states for the model. The reason we used only 4 states was because of space allocation issues that will be discussed in the Issues section of this report.

For training, we extracted the phoneme values from the all the sound files in the training set, and passed them in to a method that would estimate the HMM using the Baum-Welch algorithm. The resulting HMM was then stored in a .hmm file.

The Baum-Welch algorithm computes the transition and emission probabilities of an HMM when it is given only the emission sequence. We choose lambda = (A, B, pi) so that P(S, lambda) is locally maximized. For each entry in the transition matrix, all the probability paths to that cell are summed and normalized:

$$\sum_{t=1}^{T-1} \xi_t(i, j) = \text{expected number of transitions from } S_i \text{ to } S_j.$$

The same is done for the probabilities emanating from that entry:

$$\sum_{t=1}^{T-1} \gamma_t(i) = \text{expected number of transitions from } S_i$$

Finally, divide:

$$\bar{\pi}_i = \text{expected frequency (number of times) in state } S_i \text{ at time } (t = 1) = \gamma_1(i)$$

$$\bar{a}_{ij} = \frac{\text{expected number of transitions from state } S_i \text{ to state } S_j}{\text{expected number of transitions from state } S_i}$$

$$= \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

$$\bar{b}_j(k) = \frac{\text{expected number of times in state } j \text{ and observing symbol } v_k}{\text{expected number of times in state } j}$$

$$= \frac{\sum_{t=1 \text{ s.t. } O_t = v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}.$$

This divides the expected transitions from one state to another by all the expected transitions from a state. As certain transitions are observed, the probabilities of observing more popular ones increase and form local maximums [12].

For testing, we compared the test phoneme values against each language HMM and computed the log[P(test sequence | model)] using the Forward algorithm. Since we had such a large sequence of phonemes being passed in, taking the logarithm of each probability was more practical because each probability was so small. We ended up with very large negative numbers, which is why for the final we normalized the language log probability scores to a scale of 0 to 100, 0 being the lowest match.

In the Forward Algorithm, if we define a variable a(i) = $P(o_1, o_2, \ldots o_t, q_t = s_i \mid \text{lambda})$, the probability of the observation sequence $\{o_1, o_2, \ldots, o_t\}$ and state s_i given the model lambda. We solve for a(i) inductively:

1) Initialization:

$$\alpha_1(i) = \pi_i b_i(O_1), \quad 1 \le i \le N.$$

2) Induction:

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(O_{t+1}), \quad 1 \le t \le T - 1$$

$$1 \le j \le N.$$

3) Termination:

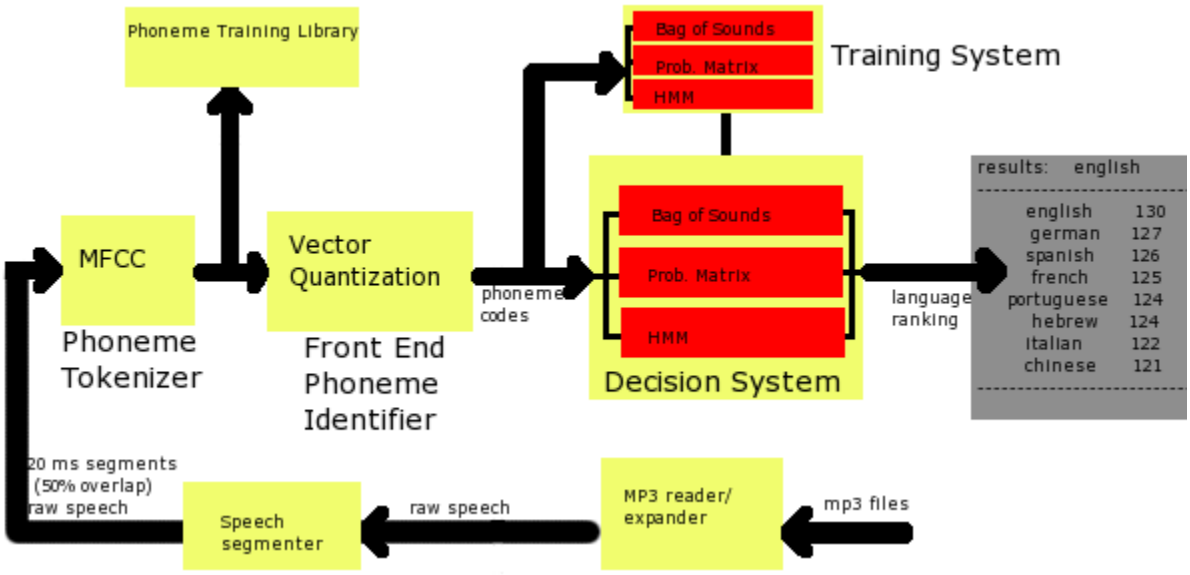
$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i).$$

Step 1) initializes the forward probabilities to a state s_i and initial observation o_1. The second

step, the induction step, shows that state s_j can be reached from the N possible states, and step 3 sums all the alpha variables together [12].

Code Organization

Our program was split into two parts, based on functionalities of the code. The flow graph below depicts the various programs that were used in this project, and how they are connected to each other.



Implemented System Flow Diagram

The systems shown above, each represented by a yellow box, were organized into two groups, the front-end and back-end. Originally the front-end was designed around the usage of the DSK, but due to the circumstances that led to us not utilizing the DSK, the main difference between the front-end and back-end became their overall purpose. For more information, see our Issues section.

Front-End Implementation

The front-end part of the project, as show in the flow graph, implements the following functions:

- Reading in MP3 files and expanding it to raw speech
    - This was done in MATLAB using a combination of the MATLAB utility mp3reader distributed by Columbia University [4] and a speech expansion code that we wrote. The mp3reader parsed the MP3 and was expanded to raw speech by our written code.
- Creating a series of 20ms segments with 50% overlap
    - Used our speech segmenter code written in C
- For each segment, create a MFCC vector

    - Each segment was passed in to our MFCC calculator code, which was written in C, which returned a single vector for each segment that described the observed phoneme within the segment
- When phoneme training, take all the vectors created from the previous part and form a phoneme library of a desired number of phonemes
    - Our vector quantization code, written in C, implemented a method of gathering all the MFCC vectors of the MP3 and creating the desired number of centroids. Each centroid represents one phoneme, which together make up the phoneme library.
- When classifier training or testing, identify which phoneme from our library is the closest to each MFCC vector
    - Given the MFCC vector, a phoneme classifier program (in C) determined the distances between the vector and each of the phonemes in our phoneme library. The phoneme that was closest to the vector was identified as the observed phoneme

Back-End Implementation

Unlike the front-end system, the back-end only has two main functions: to train and to test the three decision algorithms. This way, the program is clearly split into the preparation stage, implemented by the front-end's phoneme classification and identification, and the actual decision stage, implemented by the back-end's three classifier algorithms that determine which language was observed. The three classifiers used are the Bag-of-Sounds, the Probability Matrix, and the Hidden Markov Model algorithms.

When training the classifiers, the Bag-of-Sounds and the Probability Matrix algorithms take in one phoneme at a time from the front-end. Both algorithms add the phoneme to their calculations in real-time, reading in one phoneme at a time; however, the Hidden Markov Model, because of the way that the modeling is implemented, requires the entire phoneme sequence to be read in before a comprehensible HMM can be used. Each of them stores the outcome of their training in files. The Bag-of-Sounds records the most often seen phoneme or sequences of phonemes and stores them in a file for each language. The Probability Matrix creates three files for each language; the three files contain the three matrices used in the Probability Matrix algorithm, recording the relative probabilities of seeing a specific phoneme after another within one, two, or three steps. The Hidden Markov Model algorithm produces an HMM for each language, and stores them in .hmm files.

When testing, the format that the phonemes are transferred to the classifiers remains the same as in training. Each phoneme is passed to the Bag-of-Sounds and the Probability Matrix, while the whole sequence must be passed to the Hidden Markov Model. However, the difference is that the algorithms now use the stored files from training instead of creating them. Each algorithm reads in the files that were created during training and, depending on the classifier, somehow compares the observed phoneme sequence with the information given in the stored files. The results of each classifier is displayed in a sorted, ranked manner, each language

getting a certain number of points and the language with the highest number being determined as the language of choice.

In determining the final result, the scores of the languages are weighted equally, added up and then averaged. The final decision may or may not be the language of choice of any of the classifiers; since the choice is score-based, the language that has scored relatively well in all three classifier may easily win over the language that scored highest in only one classifier. This way we don't rely on what one classifier decides as its language of choice, but rather on what the overall system, comprised of the three classifiers, decides as the most likely language observed.

Training

Phoneme Training

Before training the system to learn the characteristics of a particular language, the system must be able to correctly classify speech into phonemes. As discussed in the vector quantization algorithm, we do not know which MFCC vectors correspond to which phonemes a priori. Thus, the centroids of each phoneme must be calculated before any other classification can be done.

First, it must be determined how many phonemes exist in the speech that will be trained with and tested. Each natural language varies with how many phonemes it uses, and which phonemes appear. For instance, English uses about 44 to 48 phonemes, depending on classification method, while Mandarin Chinese contains between 39 to 43. The number of phonemes chosen must be sufficient to distinguish the union of the phoneme sets for all the languages considered [2].

Based on the success of other phoneme classifiers, 128 phoneme centroids were chosen. To determine the location of these 128 centroids, the phoneme training speech was required to have a wide range of languages and speakers. The training set chosen was a multilingual translation of Edgar Allen Poe's The Raven, spoken by 22 speakers in 12 languages [3]. This allowed for an even selection of speaker's gender and speaking characteristics, and used the languages:

- English
- Dutch
- Finnish
- French
- German
- Italian
- Latin
- Portuguese
- Russian

- Spanish
- Swedish
- Yiddish

Although this set is not the same that was used for language training, it provides a wide range of phoneme representation.

Classification Training

Each of the three language classifiers described above must be trained with sample speech of each language. The languages used to train this system, and thus the languages that the system is (theoretically) capable of identifying, are the following:

- English
- French
- Spanish
- Portuguese
- Italian
- German
- Chinese (Mandarin)
- Hebrew

The training speech for each of these languages used samples of chapters from audio books [3]. Each training set consisted of 150 to 500 MB of MP3 files (sampled at various rates). When converted to raw speech at 16 kHz, this becomes 1.5 to 7 GB each.

This training set required several hours to convert to raw speech, and several minutes per language to train.

Testing Sets and Testing Results

Training analysis was performed on independent chapters from audio books from those used to train. Each chapter was between 10 and 20 minutes of clean speech. A total of 48 chapters were used for testing purposes. Some of these chapters were from the same books as the training set, and some were from new books.

The following table shows the result of testing for each type of classifier. Note that these results were computed by taking the ratio of the number of correct results over the number of total tests. If a classifier gave the correct language as its second result, then it received half-credit.

Bag-of-Sounds	Prob. Matrix	HMM	Total
47.9%	56.3%	45.8%	51.0%

As shown, the system gives the correct result about half of the time. Note that random guessing would result in an accuracy of 12.5% (guessing out of eight languages). Top-of-the-line systems are capable of achieving an accuracy of about 97% for less than a minute of speech [10].

One note of interest is that the accuracy of our system varies greatly from language to language. For instance, none of the algorithms were able to correctly classify any of the Italian testing data. In fact, if we remove Italian from the testing set, we see an across-the-board improvement:

Without Italian:

Bag-of-Sounds	Prob. Matrix	HMM	Total
53.5%	62.8%	51.2%	57.0%

Another characteristic is that the ensemble averaging seems sub-par. This is noted since a single classifier (Prob. Matrix) is more accurate than the full ensemble. There are many cases in which the ensemble should be more accurate than any individual classifier (for instance when every classifier gives the correct result in second-place, but different answers for first place).

One future tweak might be to do more background research into ensemble averaging, which has become a field in itself.

Performance Speed

While this system does not run on the DSK, it is still designed with memory and processor constraints in mind. The advantage of this design mentality is to allow large training sets to be processed in a reasonable amount of time.

Note that only three components of the full system require a full speech file to be stored in memory at once:

- MP3 expander
- Vector Quantizer
- HMM Classifier

Note that the first two of these are run off-line, and the last was third-party software. The MP3 expander was the only component of the system implemented in Matlab, and by far the least

efficient. For example, when performing the system tests mentioned in the previous section, about 30 hours were required to expand the original MP3's into raw speech format, while only one hour was required to classify the speech. This disparity was caused by the dlmwrite function in Matlab, which does not appear to scale well with vector size.

While the final program is not designed to run in real time, it would certainly be capable of it. In a single second of processor time, the system is capable of classifying 296,377 samples. Since speech is sampled at 16 kHz, this allows the system to classify over 18 seconds of speech in a second of processor time, which is more than sufficient to keep up with an incoming stream.

Another considerations to performance was the third-party software used for FFT and DCT libraries. This software was developed at CMU by the Spiral group, and is platform-specific code that is consistently more efficient than Intel's hand-tuned libraries [5].

Demo

The demo of this code took place in the lab, where we recorded short segments of speech from classmates and professors, which was then classified. Our initial demo resulted in relatively poor results, most likely due to the noisy lab and short recording time. We then recorded Daphne Tsatsoulis reciting The Raven, translated to German and Spanish, in a quieter environment. The result of this classification was successful on the Spanish recitation, but not for the German.

The success of the classification of Spanish speech was a testament to the ensemble decision system used, since individually none of the classifiers ranked Spanish as their first choice. One hypothesis for the failure of German classification was that Daphne's German dialect and accent is atypical, which may throw off the classification system.

Issues

DSK

As stated previously, the original organization of the front-end was designed for the usage of the DSK. Our initial goal was to run the majority of the front-end system in the DSK. Specifically, the PC would send segments of an MP3 file to the DSK, which would run the front-end part of the program, and return a phoneme vector. The PC side would then use these to either train and create a library of phonemes, or classify them to the library phonemes.

The problems that we encountered trying to send and receive packets between the DSK and the PC prevented us from utilizing the DSK at all. Even with the help of the TA recommended for his proficiency of using the DSK we were not able to resolve this issue, and considering the large amount of time already spent attempting to utilize it, we gave up trying to implement

the usage of the DSK in our project. We do not think using the DSK would have affected our overall project greatly, especially since the code that we had planned to put on the DSK were already written in C. If the connection issue had been fixed, only minor changes would have been required to run the code on the DSK.

Vector Quantization

We encountered one strange error when using our Vector Quantization algorithm. This error occurred when we tried to cluster the vectors after we split a centroid. In several instances it was trying to take the mean of the vectors, but there were no vectors within the region. This was a real surprise, since the algorithm divides up the vectors in such a way that there should have been an even division among the vectors. Even if a large number of vectors were focused around one point, the number of centroids should have corresponded to that, leaving still an even distribution of vectors. Also, since the centroids before the split are taken to be the mean of the clustered vector, the idea that the centroids favor one side of the region to another is out of the question.

The only explanation that we came up with was that when dealing with outliers, there were centroids that were assigned to one outlier only. Therefore, when we tried to split the centroid and cluster its vectors, obviously the outlier would have to go to one only, and the other centroid would end up with no vectors of its own. In order to resolve this issue we simply assigned a value of zero to the centroid when this error occurred. This would ignore the outliers that were affecting our centroids, making sure our mapping was not hindered by random outliers.

HMM memory allocation

While we were training our system to compute our language HMMs, we originally wanted to use more states for each language model, however, since the code we used required the input of the entire phoneme training sequence, we would run out of memory from all the memory allocations that was going on. In the code, they build a matrix that stores pointer values, resulting in N*M*sizeof(int) malloc operations. Since the phoneme sequences it is reading in is in the order of millions, it was not very possible to run such memory-costly operations with the machines we had, so we could only lower the number of states we used.

Future Ideas

There are many improvements or alternate approaches that could be made for future implementations. They are listed in ascending order of the magnitude of work required to implement:

One possibility for errors in the phoneme classifier may be due to training on different languages than the testing set. For the Romance/European languages, this would most likely have less of an effect, but it is likely that Chinese, for instance, may contain phonemes that were not found

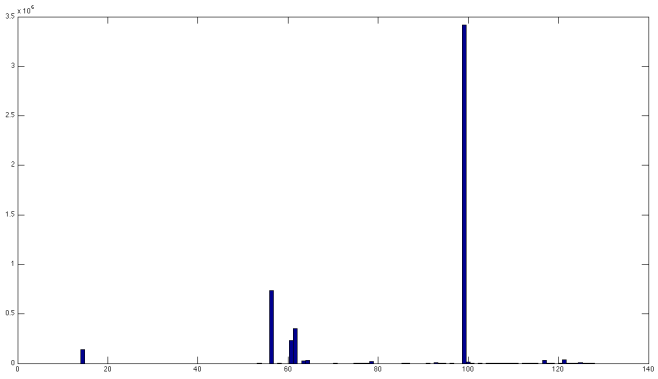
in our training set. Training the phoneme classifier on the same languages that the rest of the system uses may lead to better synergy of classification and better results.

Another issue might be due to the timescale associated with a single phoneme. While 20 ms is a common length for some speech processing applications, other systems benefit from longer segments. We did not test the effect of phoneme length on accuracy of classification, so this change may produce fruitful results.

While the use of audio books allows for many hours of clean speech in many languages, there are some limitations. Specifically, the speech is typically only performed by relatively few speakers, and often contains proper nouns. There is no guarantee of diversity in speaker characteristics, and the recording process is not uniform. By using a professional speech database for training, a system may be able to represent a better model of each language, which would result in more accurate classifications.

As discussed above, there can be errors that occur due to different ways of speaking the same language. For instance, when one is speaking English in a formal setting (e.g. giving a speech), a different vocabulary is employed than when chatting with friends. Alternatively, the same language may have many dialects. Rather than grouping all of these different instances under the same umbrella of “English”, a classifier could treat each as separate languages. Then, when reporting results, print out the same string (i.e. “English”) for all of this subcategories. While the subcategories may be very similar to each other, and be misclassified, the end result is treated the same so this could result in better accuracy overall.

One trend we noticed was that a particular subset of phonemes occurred more frequently than the others. For example, our initial phoneme training resulted in the following histogram, when classifying the training set:



Phoneme count vs. phoneme identifier

One likely possibility is that the common phonemes are representations of silence, which occurs much more often in speech than any other specific sound. Since this gives no indication about language, perhaps these phonemes could be filtered out before any classification, which would

result in more accurate classification when values of adjacent phonemes are determined (which is used for all the classifiers above).

As described in [2], there are multiple ways to classify phonemes from raw speech. We chose vector quantization due to ease of implementation and reliability. Another method offered is Parallel Voice Tokenization (PVT). This repeats the phoneme classification for each language specifically. The operating principle is that since each language may use a different set of phonemes, then better classification results from a phoneme classifier that is trained only on a single language. Thus, if a phoneme sequence is developed for each language, and then the back-end classifiers are run on the language-specific phoneme sequences, they may result in better probability estimates and thus better classifications.

Finally, a performance increase might result from using a different implementation of Hidden Markov Models. We used a third-party library [6], which was implemented inefficiently. As a result, it was only feasible to create very small-order models (4 states). This leads to approximations in the classification. If a more efficient implementation was used, then maybe more states could have been modeled, leading to a more complex and accurate ranking process.

Final Schedule and Work Break-down

Weeks of..

2/13-2/20:

All:	Research and finalization of design

2/27:

Eric:	Write implementation of Mel Frequency Cepstrum Coefficients for DSK. Start on FFT algorithm. Agree with Jinyoung on output format for tokenization.

3/6:

Eric:	Finish MFCC implementation, test/debug with sample speech.
Jinyoung:	Begin work on Vector Quantization.

Milestone: Finish implementations for MFCC on DSK.

3/13:

Bridget:	Begin work on C implementation of MP3 reader
Eric:	Download audio books in various languages, define language set. Develop phoneme classifier that uses the results of vector quantizer.
Jinyoung:	Finish implementation of Vector Quantizer trainer, structure of code-book.

3/20:

Bridget:	Find third-party MP3 reader MATLAB code.

Eric: Attempt debugging of DSK version of front-end.

3/27: Update in class

Eric: Attempt debugging of DSK version of front-end. Write MP3 reader in MATLAB.

4/3:

Eric: Move system from DSK to PC standalone. Finish front-end, write bag-of-sounds classifier. Helped Bridget outline HMM classifier.

Jinyoung: Start re-implementation of an expanded version of the previous group's strategy.

All: Meet with TA in lab to attempt to resolve DSK problem.

4/10:

Bridget: Found third-party HMM code with Eric. Write HMM classifier.

Eric: Write main program wrapper code. Compiled phoneme training sets. Trained phoneme classification system. Begin compiling training sets for language classifiers.

Jinyoung: Finish probability matrix classifier.

4/17:

Eric: Train bag-of-sounds classifier

Jinyoung: Train probability matrix classifier.

4/24:

Bridget: Finish HMM classifier, train HMM classifier. Test overall system.

Eric: Integrate HMM into rest of system. Develop ensemble averaging scheme. Test overall system.

Conclusion

Spoken language identification is naturally a difficult problem. While it is a relatively new research topic, current top-level implementations have evolved to use sophisticated probabilistic and pattern recognition techniques. As such, adapting this problem to be a semester-long undergraduate project requires many simplifications. Thus comparing the results of this project to those introduced at academic conferences is less than fair.

On the other hand, there are many improvements that could be made to this system while still keeping it relatively straight-forward and reasonable to implement. Although our system improved greatly upon the previous group's results, there are still many things that a future 18-551 group could do to improve our system. This means that even though this topic has now been attempted twice in this class, it can still be a valid project for a future group.

References

[1] Lee, C.-H. “Principles of Spoken Language Recognition.” Chapter 39, *Springer Handbook of Speech Processing.* Benesty, Sondhi, Huang. 2008. 785-795.

[2] Li, H., Ma, B., Lee, C.-H. “Vector-Based Spoken Language Classification.” Chapter 42, *Springer Handbook of Speech Processing.* Benesty, Sondhi, Huang. 2008. 825-840.

[3] "Free Audio Books", Books Should Be Free: Free Audio Books from the public domain. <http://www.booksshouldbefree.com/language/>, (c) 2011

[4] Ellis, Dan. "mp3read and mp3write for Matlab". LabROSA. <http://labrosa.ee.columbia.edu/matlab/mp3read.html>, (c) 2010

[5] "Hardware/Software Generation for DSP Algorithms". Spiral. <http://spiral.net>, (c) 2011

[6] Kanungo, Tapas. "Home Page for Tapas Kanungo". UMDHMM Toolkit. <http://www.cfar.umd.edu/~kanungo>, (c) 2011

[7] Roque, Abel V., Velik Bellemin, and William Y. Liu. *Group #5: Language Identification*. Tech. Pittsburgh: 18-551, 2002.

[8] Harper, M. P., Maxwell, M. “Spoken Language Characterization.” Chapter 40, Springer Handbook of Speech Processing. Benesty, Sondhi, Huang. 2008. 797-809

[9] Zissman, Marc A. “Comparison of Four Approaches to Automatic Language Identification of Telephone Speech.” IEEE Transactions on Speech and Audio Processing, Vol. 4, No. 1. January 1996.

[10] "NIST LRE09 Evaluation Results." *Information Technology Laboratory Homepage*. Web. <http://www.itl.nist.gov/iad/mig//tests/lre/2009/lre09_eval_results/index.html>.

[11] Moore, Andrew W. “Clustering with Gaussian Mixtures.” *Computer Science. Carnegie Mellon University.* Web. <http://www.autonlab.org/tutorials/gmm14.pdf>.

[12] Rabiner, Lawrence R. “A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition.” IEEE Vol. 77, No. 2. February 1989.