# An Isolated-Word, Speaker-Dependent Speech Recognition System

## 1. Introduction

This paper describes experiments for an isolated-word, small-vocabulary, speaker-dependent, speech recognition system. We specifically look at two vocabulary sets:

$$Set_1 = \{\textbf{one}, \textbf{two}, \textbf{three}, \textbf{four}, \textbf{five}\} \tag{1}$$

$$Set_2 = \{\textbf{dog}, \textbf{god}\} \tag{2}$$

The first vocabulary set $Set_1$ might be useful for an automated telephone service, where a user is guided through a set of menus by saying one of the five words in the set; many credit card companies and other consumer service organizations have such systems these days. Of course, in a real-world application, we would want to make such a system *speaker-independent*, rather than *speaker-dependent*; however, such a system is beyond the scope of this paper.

The second vocabulary set $Set_2$ tests the speech recognition system for two words that sound very similar when spoken. While many of the techniques used in this paper are borrowed from the speech recognition literature [1], we do make some simplifications, especially in the feature-extraction step.

## 2. Speech recognition system

### A. Overview

Figure 1 below illustrates the overall speech recognition system, while Figure 2 illustrates the signal-to-symbol preprocessing and conversion. Note that for $Set_1$, $M = 5$, while for $Set_2$, $M = 2$. As can be seen from Figure 1, words in the speech recognition system are modeled statistically with discrete-output hidden Markov models (HMMs); therefore, the one-dimensional, real-valued speech signal at the input must be segmented into isolated words and converted to a sequence of discrete observables $\{O_t\}$. For now let us assume the speech signal at the input has already been segmented into an isolated word. Then, the signal-to-symbol conversion begins by normalizing the sampled sound signal to a maximum range of $\pm 1$. Next, the sampled speech signal is partitioned into shorter $k$-length sequences with 50% overlap. Each of these $k$-length sequences is then multiplied by the $k$-length *Hamming window function*, in order to reduce spectral leakage. Note that the Hamming coefficients $H[i]$ are defined by,

$$H[i] = 0.54 - 0.46\cos\left(\frac{2\pi i}{k-1}\right),\ i \in \{0, 1, \ldots, k-1\}. \tag{3}$$

After windowing with the Hamming function, we apply the Fast Fourier Transform (FFT), take the absolute value of the resulting spectrum, and retain the first $k/2$ coefficients, since the last $k/2$ coefficients will contain no additional information (for real-valued signal). Thus, this procedure converts the one-dimensional sampled speech signal into a sequence of vectors $\{v_t\}$ of length $k/2$, where the vectors $\{v_t\}$ contain the short-term spectral magnitude content of the speech signal over time.

To convert the sequence of vectors to a sequence of symbols, a VQ-codebook of $L$ prototype vectors is assumed to have been generated during training using the iterative LBG VQ algorithm [2]; during run-time, therefore, the streaming sequence of spectral vectors $\{v_t\}$ can be converted to a sequence of discrete observables $\{O_t\}$ through vector quantization with the VQ codebook that was computed during training. Finally, in order to classify the unknown speech signal at the input, we evaluate $P(O|\lambda_i)$, $i \in \{1, 2, \ldots, M\}$, where $\lambda_i$ denotes the trained HMM corresponding to the $i$th word in the vocabulary set, $O = \{O_t\}$, $t \in \{1, 2, \ldots, T\}$, and $T$ denotes the length of the observation sequence corresponding to the spoken word at the input. We classify the speech signal at the input as the $l$th word in the vocabulary set, corresponding to that HMM which yields the largest probability such that,

$$P(O|\lambda_i) < P(O|\lambda_l),\ \forall i \neq l. \tag{4}$$

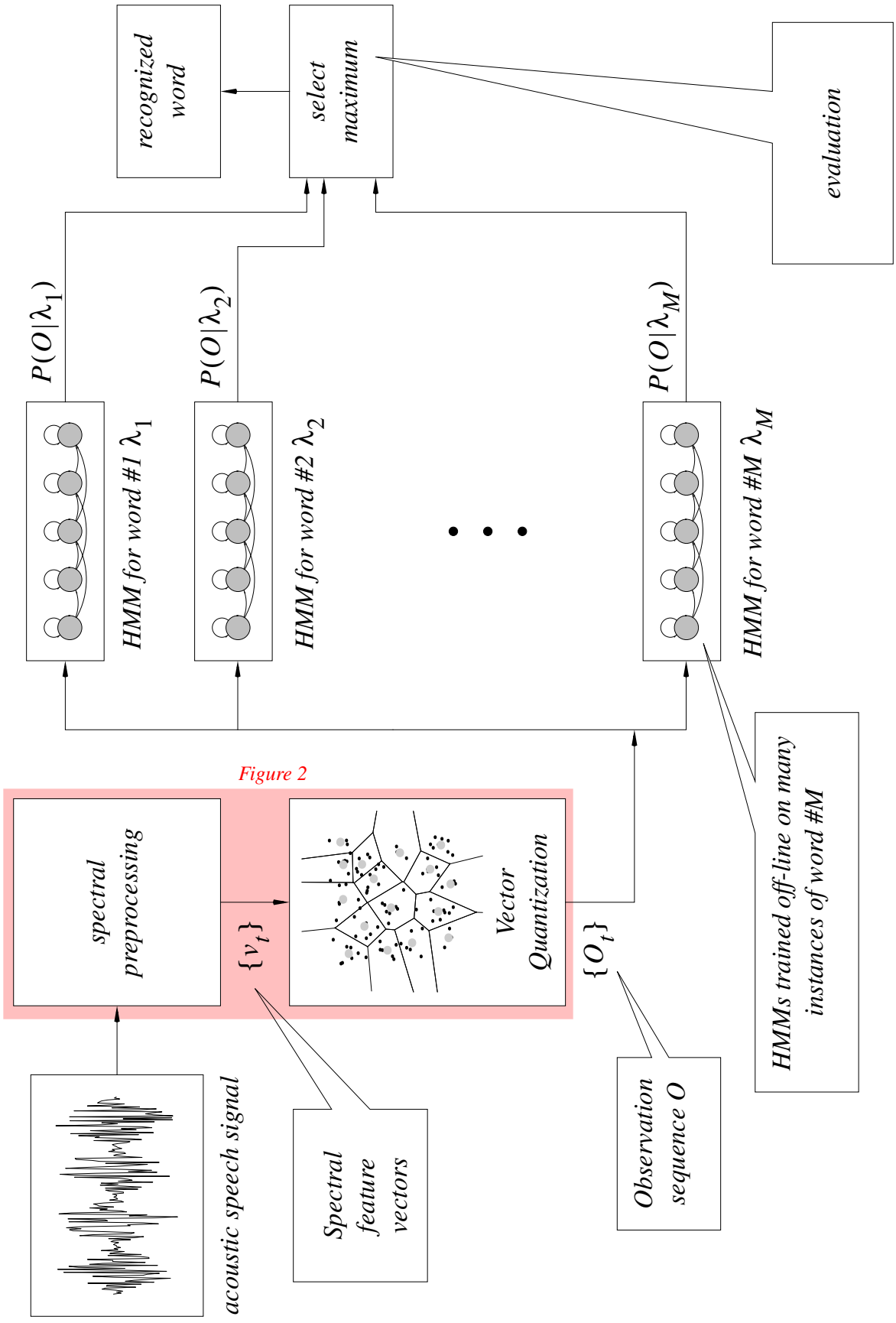


Figure 2

**Figure 1: Overall isolated-word speech recognition system**

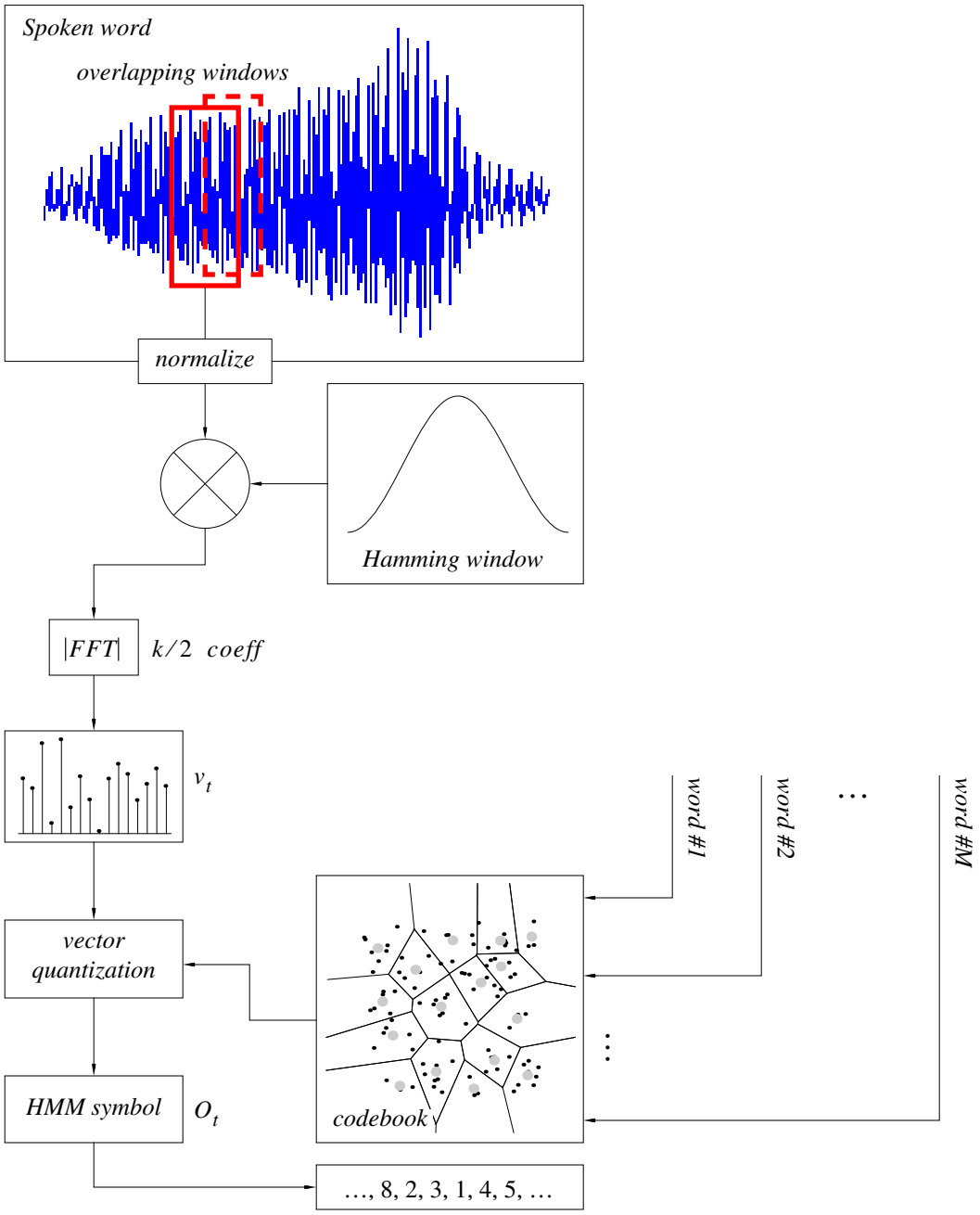


***Figure 2: Conversion of speech signal to a sequence of discrete symbols.***

## B. Data collection

Data was collected through the sound input of a Titanium G4 laptop. For each word instance over both vocabulary sets, a male speaker's voice saying the same word repeatedly was recorded for approximately two minutes at a sampling frequency of 44.1kHz and a resolution of 16 bits; this process yielded between 115 and 120 spoken instances of each word. The resulting speech files were then down-sampled to 8kHz "wav" files[1], which then served as our data for the experiments described below.

*1. See http://mil.ufl.edu/~nechyba/eel6825/course_materials.html to listen to these "wav" files.*

### C. Word segmentation

Each two-minute "wav" file was segmented into isolated words by analyzing the power in the speech signal over 32ms segments (256 samples at 8kHz), with 16ms overlap (128 samples at 8kHz). Specifically, we normalized each speech file to a maximum range of $\pm 1$ and then computed the short-time power $P$ of the signal

$$P = \sum_{j=t}^{j+79} x[j]^2 \,,\; j \in \{0, 128, 256, \ldots\} \,, \tag{5}$$

where $x[j]$ denotes the $j$ th sample in each sound file. We recognized spoken words in the signal for segments where,

$$P > \theta_{threshold} \,,\; \theta_{threshold} = 10^{-3} \,. \tag{6}$$

The specific value of $\theta_{threshold}$ was determined by trial and error, and worked well in segmenting out the words from the two-minute speech files.

### D. Training and testing

After word segmentation, we converted each spoken word into a sequence of spectral vectors following the procedure depicted in Figure 2 and summarized in Section 2-A above (with $k = 256$). Thus each vector $v_t$ was of length 128, while the average length of the resulting vector sequences was 22.6 for $Set_1$ and 22.4 for $Set_2$ (approx. 360 msec/word instance). For each vocabulary set, we now split the data into two groups — one for training and the other for testing. For each word over both vocabulary sets, we reserved the last 40 spoken word instances for testing, using the first 75 to 80 spoken word instances for training (the number available for training is variable, because the number of instances for each word varied slightly). The test data was not used in any part of the training process described below.

For each vocabulary set, we generated a unified VQ codebook of prototype vectors over all words in that vocabulary set using all available training data of spectral vector sequences. We employed the iterative LBG VQ algorithm, generating VQ codebooks of sizes $\{2, 4, 8, 16, \ldots\}$. Given the amount of available data and the number of parameters in a hidden Markov model with $L$ observables, we settled on $L = 8$ prototype vectors for the results reported below.

Finally, we trained one HMM $\lambda_i$ per spoken word using the Baum-Welch algorithm on the quantized observation sequences in the training data. As is discussed in greater detail below, we varied the number of states in the HMMs to achieve the best classification performance over the test data.

## 3. Results

### A. Vector quantization

In Figures 3 and 4, we draw the $L = 8$ prototype vectors for vocabulary $Set_1$ and $Set_2$, respectively. The $i$ th element of each prototype vector, $i \in \{0, 1, \ldots, k/2 - 1\}$, indicates that prototype vector's magnitude at frequency $f$,

$$f = \frac{8000 i}{256} = 31.25 i \,\text{kHz} \tag{7}$$

For example, the first prototype vector in Figure 3 (red), has its largest component at,

$$f = 31.25 \times 13 = 406 \,\text{Hz}. \tag{8}$$

Note that most of the frequency content of the recorded speech samples appears to be concentrated in approximately the 0Hz to 1kHz range. Had the speaker of the speech data been a person with a higher-pitched voice, we would expect a broader range of non-zero frequency components.

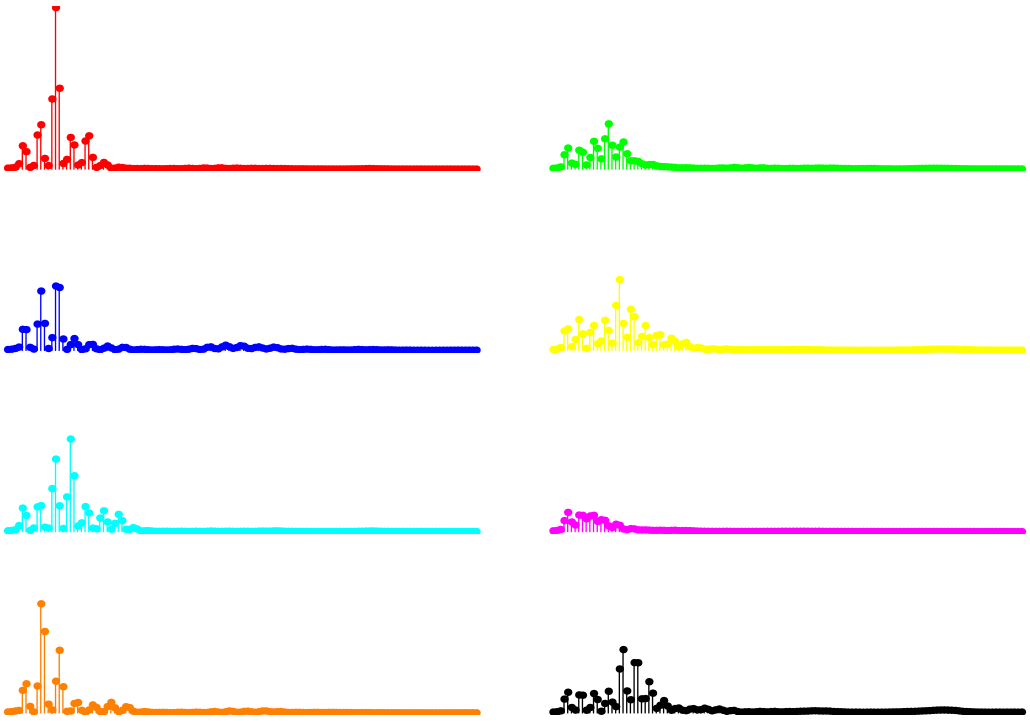

***Figure 3: L = 8 prototype vectors for vocabulary set #1 (one, two, three, four, five).***

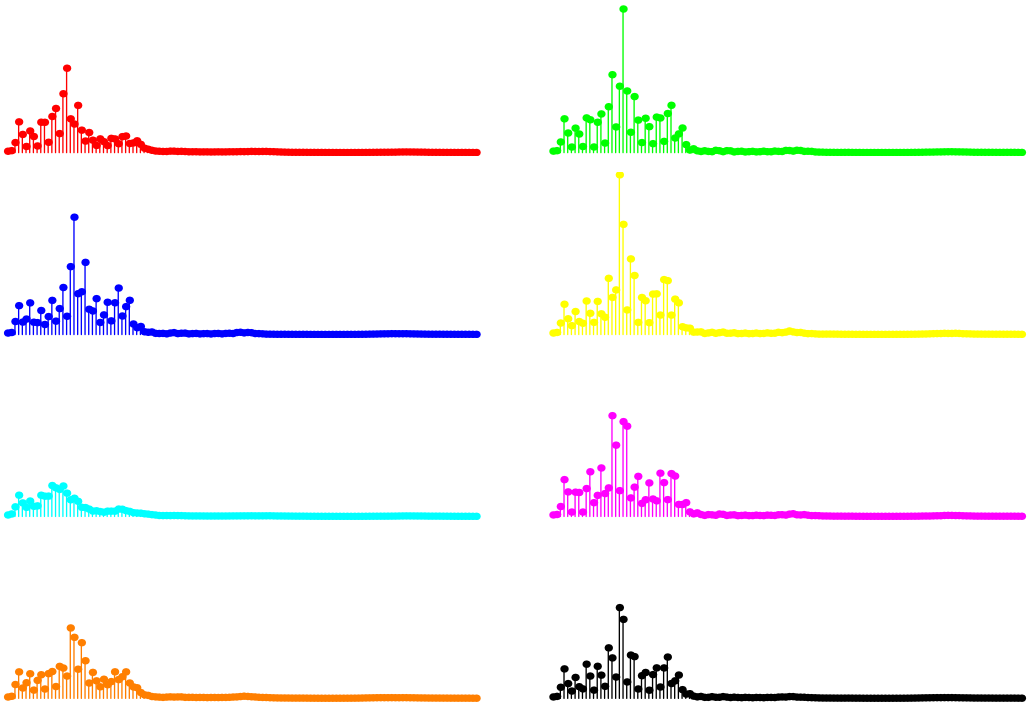

***Figure 4: L = 8 prototype vectors for vocabulary set #2 (dog, god).***

## B. Trained HMMs

In Figures 5 and 6, we illustrate the trained HMMs (6 states[1]/8 observables) for vocabulary $Set_1$ and $Set_2$, respectively. Note that the colors in Figures 5 and 6 correspond to the prototype-vector colors in Figures 3 and 4, respectively.

Before we see how these HMMs perform in classifying the test data, we point out an interesting feature of the two sets of HMMs. For the HMMs corresponding to $Set_1$ (Figure 5), both the first and last states of each HMM exhibit a large probability for the purple observable. Note from Figure 3, that the purple observable corresponds to the prototype vector with the smallest elements (i.e. least power). This should be expected, since the power of an isolated word utterance at the beginning and end will be smaller than in the middle of that utterance. Note that the same observation holds for the HMMs corresponding to $Set_2$ (Figure 6), except that now the high-probability observable for the first and last states of each HMM corresponds to the cyan prototype vector in Figure 4.

Table 1 below reports the classification performance of the trained HMMs in Figures 5 and 6 over the test data. Remember that the test data was not used in any phase of the training procedure, and consists of 40

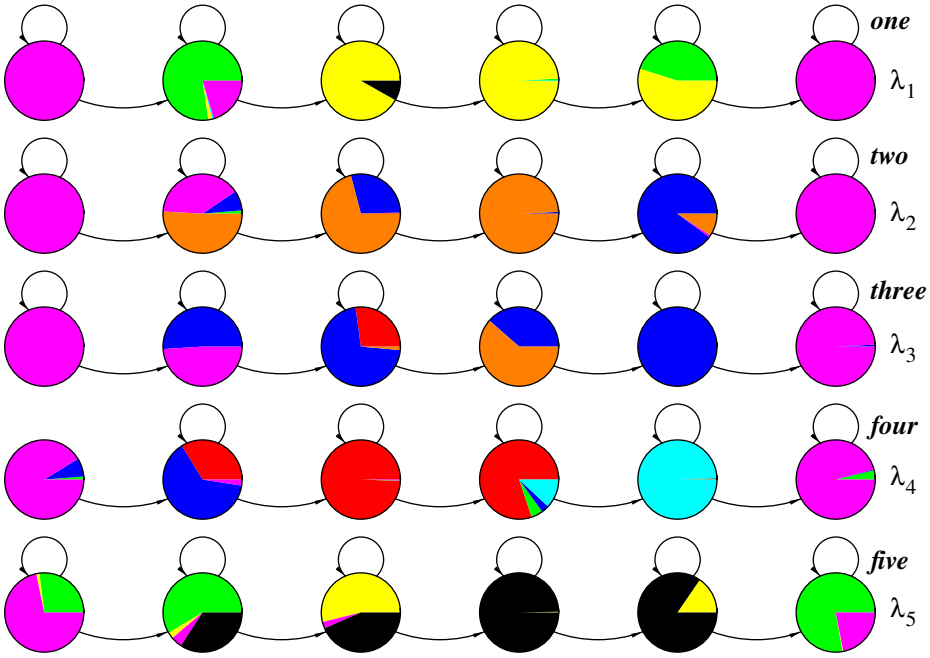


***Figure 5: 6 state/8 observable HMMs for vocabulary set #1 (one, two, three, four, five).***

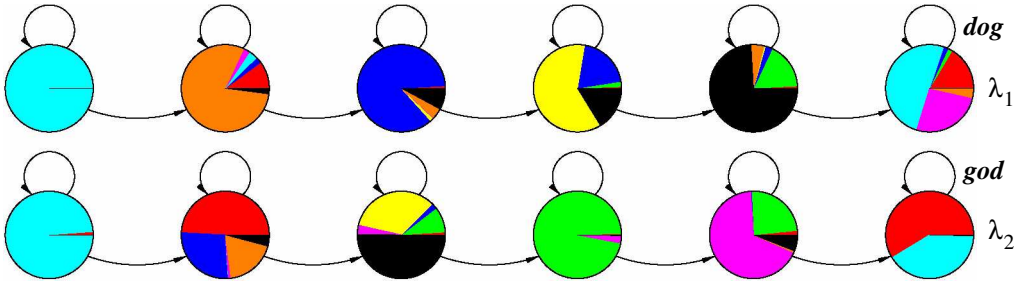


***Figure 6: 6 state/8 observable HMMs for vocabulary set #2 (dog, god).***

*1. Later in this paper, we explain our selection of six-state HMMs more fully. In short, six-state HMMs gave the best classification performance over the test data sets.*

instances of each spoken word in both vocabulary sets. We used a floor of $10^{-9}$ during evaluation of the test data over the trained HMMs; that is, prior to evaluation, we replaced each zero element in the state transition matrix $A$, the output probability distribution matrix $B$ and the initial state probability vector $\pi$ of each HMM by $10^{-9}$, and then renormalized to meet probabilistic constraints.

**Table 1: Classification performance over test data**

| $Set_1$ ***words*** | ***classification error*** |
|---|---|
| *one* | *1/40 (2.5%)* |
| *two* | *1/40 (2.5%)* |
| *three* | *1/40 (2.5%)* |
| *four* | *0/40 (0.0%)* |
| *five* | *1/40 (2.5%)* |
| ***totals*** | ***4/200 (2.0%)*** |

| $Set_2$ ***words*** | ***classification error*** |
|---|---|
| *dog* | *1/40 (2.5%)* |
| *god* | *0/40 (0.0%)* |
| ***totals*** | ***1/80 (1.3%)*** |

As can be observed from Table 1, the classification error over the test data is pretty good — 2.0% error for vocabulary $Set_1$, and 1.3% error for vocabulary $Set_2$. With more advanced feature extraction, the few classification errors could probably be reduced further or even eliminated.

## 4. Discussion

### A. Detailed examples

In this section, we illustrate a few detailed classification examples for the *dog*/*god* vocabulary $Set_2$. Figure 7 shows three different test cases for each word (i.e. *dog* and *god*); for each example, we plot the original speech signal, the corresponding observation sequence, and the relative evaluation probabilities $P(O|\lambda_1)$ and $P(O|\lambda_2)$ (green denotes the *dog* class, while red denotes the *god* class). Note that the top two examples are misclassified and poorly classified, respectively.

By comparing the observation sequences with the HMM models, it should be intuitively obvious why each test instance results in either misclassification, poor classification or good classification. Consider, for example, the test instance *dog*, #097 (upper left corner of Figure 7); the long subsequence of green observables in that observation sequence most likely tilts the relative probability values in favor of the *god* model, due to the high probability of the green observable in state four of the *god* HMM.

### B. Different random parameter settings

Next, we examine how different random initial parameter values can influence the parameters of the corresponding trained HMMs. In Figure 8, for example, we illustrate three different trained HMMs $\lambda_a$, $\lambda_b$ and $\lambda_c$ for the *god* training data set, where the difference between the three HMMs is due to different random initializations of the HMMs at the beginning of training (i.e. the Baum-Welch algorithm)[1]. We note that each of the three HMMs corresponds to a different locally maximal solution on the log-probability hyper-surface in HMM-parameter space.

While at first blush the three HMMs in Figure 8 might appear to be very different, Table 2 below shows that they evaluate to similar log probabilities over the entire *god* training data set $\mathbf{O}_{train}$, although $\lambda_a$ is margin-

*1. See http://mil.ufl.edu/~nechyba/eel6825/course_materials.html to view movies corresponding to the training of these HMMs.*

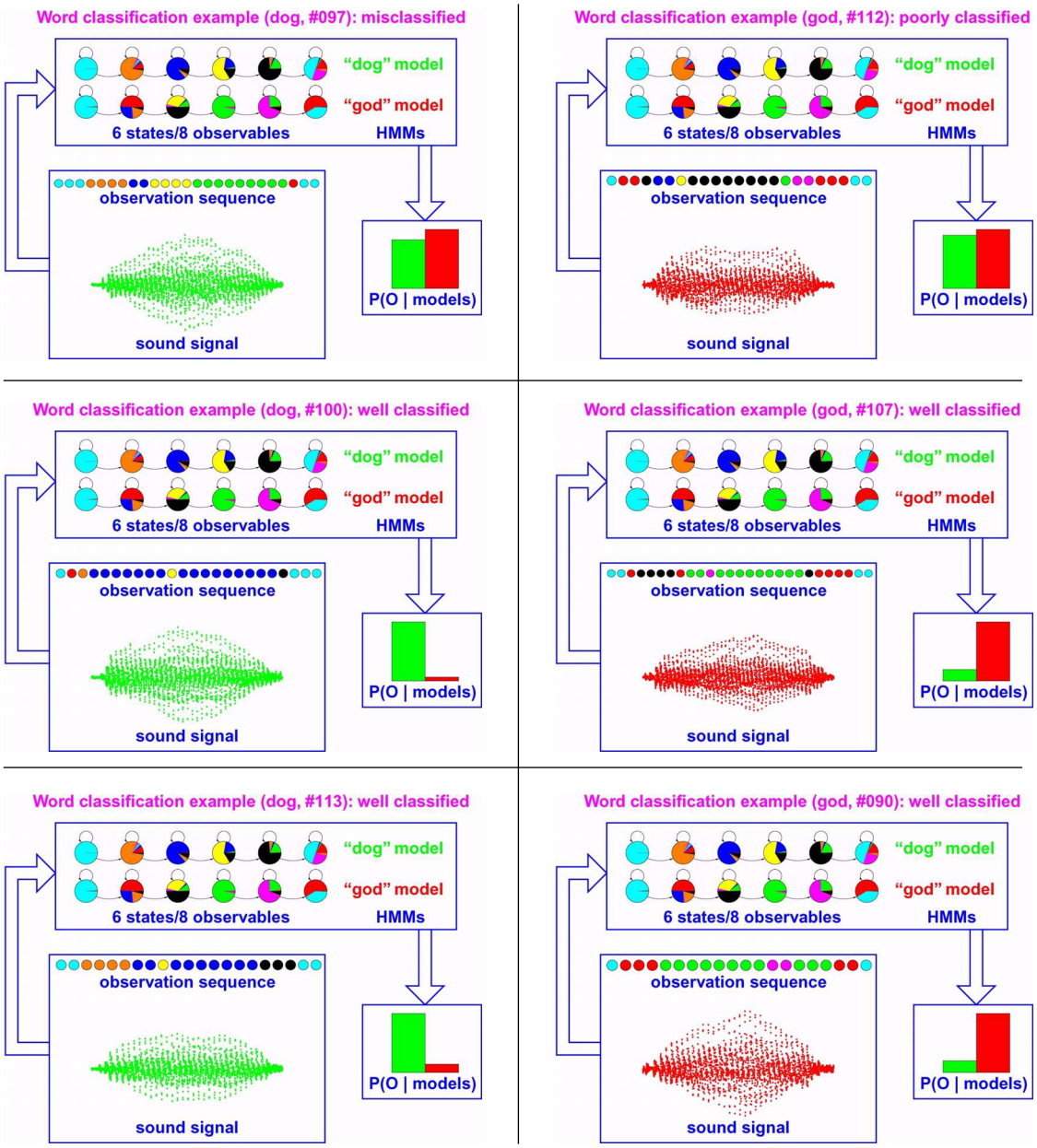


***Figure 7: Detailed classification examples for vocabulary set #2 (dog/god).***

ally better than $\lambda_b$ and $\lambda_c$. Qualitatively, we note that the role of the first state in $\lambda_a$ appears to have been assumed by the first two states in $\lambda_b$; consequently, state 3 in $\lambda_b$ corresponds approximately to state 2 in $\lambda_a$, state 4 in $\lambda_b$ corresponds approximately to state 3 in $\lambda_a$, etc. Similar comparisons can be made between $\lambda_a$ and $\lambda_c$, and $\lambda_b$ and $\lambda_c$, respectively.

### C. Varying the number of states

Next, we study how the number of states in our HMM word models impacts classification performance. In Figure 9(a), we illustrate HMM word models for $Set_2$ (i.e. *dog*/*god*), varying the number of states from two to eight. In Figure 9(b), we plot the classification error for these models over the test data set as the number of states is varied from two to nine. Note that the lowest classification error occurs for HMM models with six, seven or eight states, while the highest classification error occurs for HMM models with three states. The plot

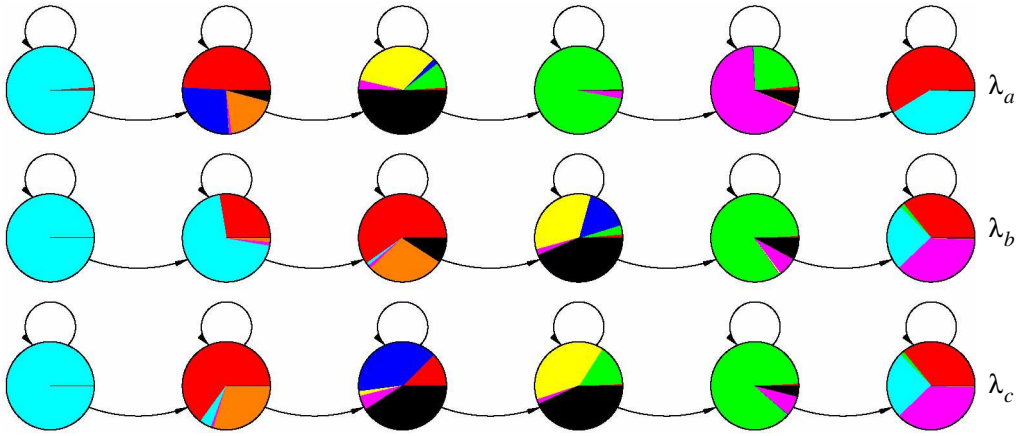


***Figure 8: Three "god" HMMs with different random initial parameter values.***

**Table 2: Training data evaluation probabilities for HMMs in Figure 8**

| $l$ | $\log P(\mathbf{O}_{train}\|\lambda_l)$ |
|---|---|
| *a* | *-854.1* |
| *b* | *-902.5* |
| *c* | *-889.8* |

in Figure 9(b) explains our preference for six-state HMMs in our word modeling task; six-state HMMs appear to be the most compact model to give the smallest classification error (1.3%).

In Figure 9(c) and (d) we illustrate the difference in classification performance between three-state and six-state HMMs for the specific test instance *dog*, #099. Note that this test instance is misclassified as the word *god* by the three-state HMMs, but is correctly classified by the six-state HMMs. This example, as well as others not shown, suggests that the six-state HMMs are able to incorporate temporal properties of our two classes, while HMMs with fewer number of states lack sufficient temporal structure to encode those same temporal properties.

### D. Viterbi analysis

In this section, we apply the Viterbi algorithm (i.e. decoding the most likely state sequence) to further explain some of the classification results of the previous sections. In Figure 10(a) and (b) we plot the most likely state sequence $q^*$ for the observation sequence and the two HMMs (three-state and six-state) in Figure 9(c) and (d), respectively. Note from Figure 10(a) that the three-state HMM appears to encode very little temporal information, since, for the observation sequence in Figure 9, $q^*$ resides almost entirely in state 3 (the red line color in Figure 10(a) indicates misclassification of test instance *dog*, #099, for the three-state HMM). For the six-state HMM, however, $q^*$ spends at least some time in each of the six states (see Figure 10(b); the green line color indicates correct classification of the test instance *dog*, #099). This example reinforces our conclusions in the previous section regarding different number of states in our HMM word models.

Next, we apply the Viterbi algorithm to recover the most likely state sequences corresponding to all 40 *dog* test instances and the six-state *dog* and *god* HMMs in Figure 6. In Figure 11(a) we plot the results for the *dog* HMM, while in Figure 11(b) we plot the results for the *god* HMM. Given these plots, we make a couple of observations. First, note how, in the aggregate, the most likely state sequences certainly appear different for the two HMM word models over the *dog* test data. Second, note that there is one instance of a state sequence for the *god* model transitioning from state six to state five. Given the left-to-right structure of the trained

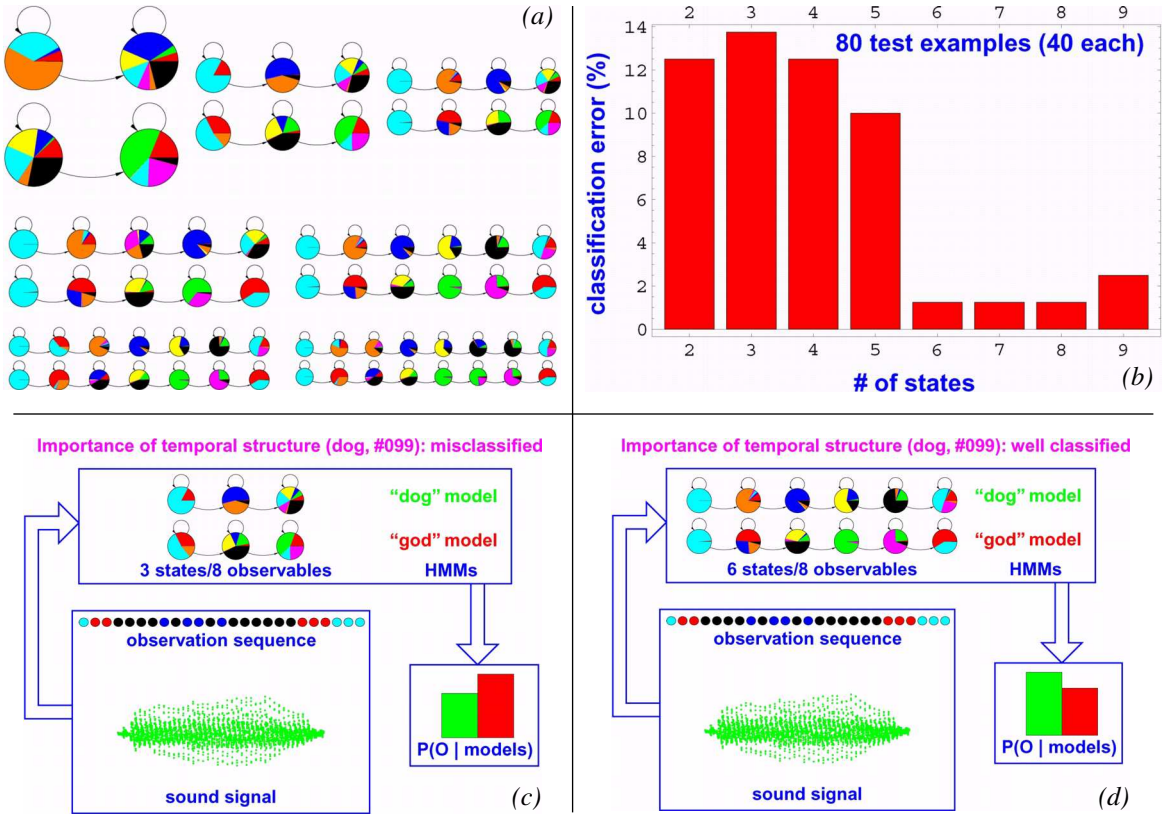


***Figure 9: Varying the number of states for example #2 (dog/god).***

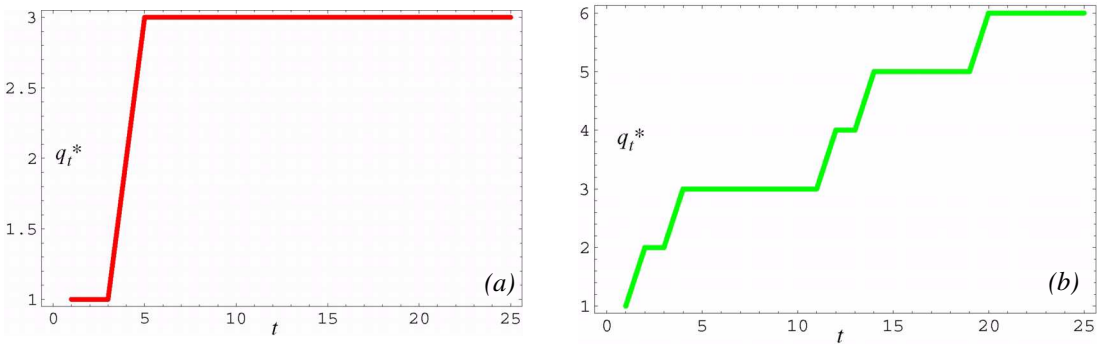


***Figure 10: Most likely state sequences for "dog" HMMs in (a) Figure 9(c) and (b) Figure 9(d).***

HMMs, how is this possible? The answer is that by flooring the HMMs (see Section 3-B), backward state transitions are in fact possible (although very unlikely).

## E. Data compression analysis

Finally, we analyze how much information is lost in our signal-to-symbol conversion. We will proceed by first computing the approximate number of bytes required to represent the uncompressed training data sets; then, we will do the same analysis for the converted observation sequences in the training data and compare the two numbers. The average length of each spoken word instance is approximately 360 msec; at a sampling frequency of 8kHz and a sampling resolution of 16 bits, that corresponds to,

$$\frac{360}{1000} \times 8000 \times 16 \text{ bits/word} \tag{9}$$

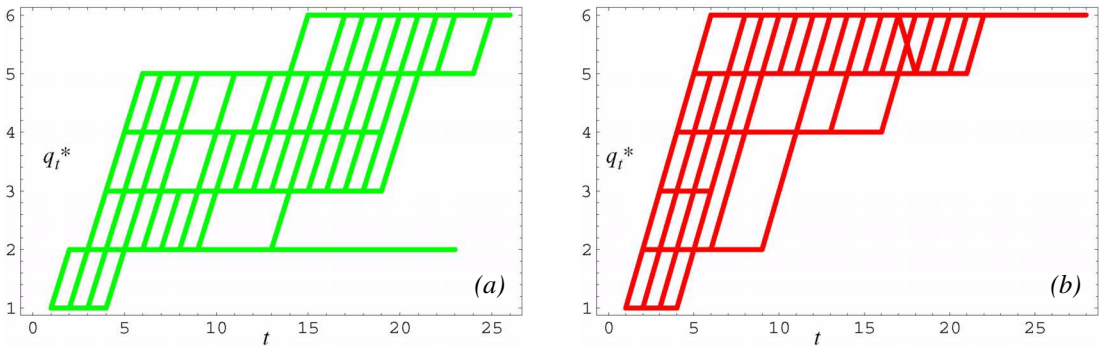


***Figure 11: Most likely state sequences for "dog" test data and (a) "dog" HMM and (b) "god" HMM***

or approximately 5,760 bytes/word. Given that the observation sequences are of average length 22.5, and that each observable can be represented by 3 bits (for 8 observables), an observation sequence can be represented by,

$$22.5 \times 3 \text{ bits/observation sequence,} \tag{10}$$

or approximately 8.5 bytes/observation sequence. Nominally, the VQ codebook requires,

$$8 \times 128 \times 4 \;=\; 4,096 \text{ bytes} \tag{11}$$

assuming 4 bytes/floating-point number. However, since we observed previously that most of the prototype vectors have approximately zero elements for frequencies above 1000Hz (three-fourth of all vector elements), the actual number of bytes required to represent the VQ codebook is closer to 1,024 bytes. Therefore, for $n$ word utterances in the training set, the total number of bytes required for the uncompressed sound files will be approximately equal to $5,760n$ bytes, while the total number of bytes required for the observation sequences will be approximately equal to $(1,024 + 8.5n)$ bytes. Consequently, our compression ratio $\gamma$ is given by,

$$\gamma \approx \frac{5,760n}{1,024 + 8.5n}, \text{ where } n \text{ = number of training instances,} \tag{12}$$

which is plotted in Figure 12 as a function of $n$. Note that for $Set_1$, we have approximately 400 training instances (80/word), while for $Set_2$, we have approximately 160 training instances (80/word). Thus, the approximate compression ratios $\gamma_1$ and $\gamma_2$ for our two case studies are 520:1 for $Set_1$ and 380:1 for $Set_2$. What is remarkable about these numbers is that despite a huge loss of information in the signal-to-symbol conversion process, we are still able to get very good classification performance over the test instances (for both case studies).

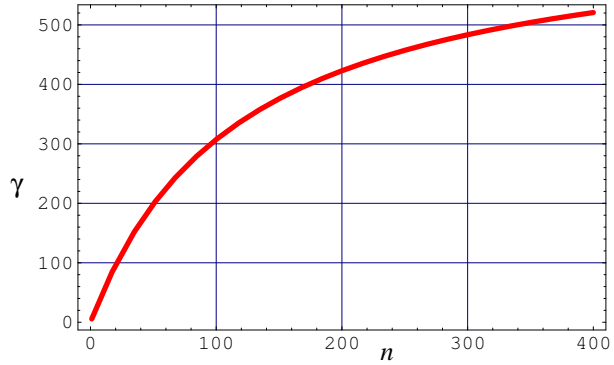


***Figure 12: Data compression ratio as a function of the number of training instances (n).***

## 5. Conclusion

In this paper, we trained and tested an isolated-word, speaker-dependent speech recognition system using discrete-output hidden Markov models (HMMs). We were able to achieve low classification error over test data for two different case studies, despite relatively elementary feature extraction and a large data compression ratio for the signal-to-symbol conversion process. Furthermore, we were able to show that classification performance over the test data changed as a function of the number of states in the HMMs, suggesting that the HMMs encode significant temporal structure in modeling individual words.

## 6. References

[1] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proc. of the IEEE*, vol. 77, no. 2, pp. 257-86, 1989.

[2] Y. Linde, A. Buzo and R. M. Gray, "An Algorithm for Vector Quantizer Design," *IEEE Trans. on Communication*, vol. COM-28, no. 1, pp. 84-95, 1980.

## 7. Appendix

Word segmentation and spectral feature vector extraction were performed in *Mathematica*; vector quantization and HMM training, evaluation and decoding were done using C-coded executables[1]; visualization of results was done using *Mathematica*[2].

*1. See http://mil.ufl.edu/~nechyba/eel6825/source_code.html*

*2. See http://mil.ufl.edu/~nechyba/eel6825/course_materials.html*