Isolated-word speech recognition using hidden Markov models

Håkon Sandsmark

December 18, 2010

1 Introduction

Speech recognition is a challenging problem on which much work has been done the last decades. Some of the most successful results have been obtained by using hidden Markov models as explained by Rabiner in 1989 [1].

A well working generic speech recognizer would enable more efficient communication for everybody, but especially for children, analphabets and people with disabilities. A speech recognizer could also be a subsystem in a speech-to-speech translator.

The speech recognition system implemented during this project trains one hidden Markov model for each word that it should be able to recognize. The models are trained with labeled training data, and the classification is performed by passing the features to each model and then selecting the best match.

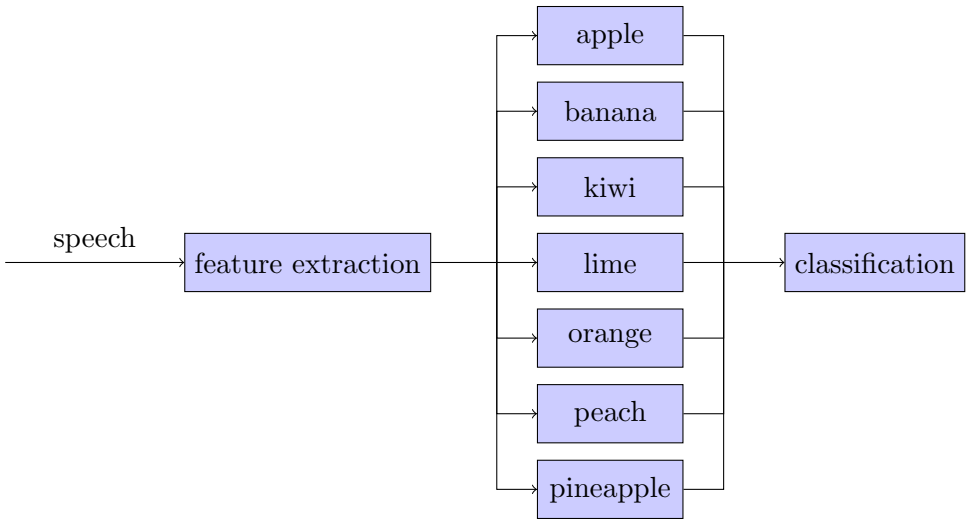


Figure 1: Flow chart of the system. The features extracted from the speech signal are passed to each word model and the best match is selected.

2 Background theory

2.1 Hidden Markov models

Basic knowledge of hidden Markov models is assumed, but the two most important algorithms used in this project will be described.

The observable output from a hidden state is assumed to be generated by a multivariate Gaussian distribution, so there is one mean vector and covariance matrix for each state. We will also assume that the state transition probabilities are independent of time, such that the hidden Markov chain is homogenous.

We will now define the notation for describing a hidden Markov model as used in this project. There is a total number of N states. An element $a_{ss'}$ in the transition probability matrix $\mathbf{A}$ denotes the transition probability from state s to state s', and the probability for the chain to start in state s is π_s. The mean vector and covariance matrix for the multivariate Gaussian distribution modeling the observable output from state s are μ_s and $\mathbf{\Sigma}_s$, respectively. For an observation $\mathbf{o}$, $b_s(\mathbf{o})$ denotes the probability density of the multivariate Gaussian distribution of state s at the values of $\mathbf{o}$. We will sometimes denote the collection of parameters describing the hidden Markov model as $\lambda = \{\mathbf{A}, \pi, \mu, \mathbf{\Sigma}\}$.

2.2 The forward algorithm

We want to calculate the probability density of an observation $\mathbf{o}_1, \ldots, \mathbf{o}_T$ for a specific model. This will be used to select the model (i.e. word) that most likely generated the speech signal.

$$
\begin{aligned}
f(\mathbf{o}_1, \ldots, \mathbf{o}_T; \lambda) &= \sum_{s_T} f(\mathbf{o}_1, \ldots, \mathbf{o}_T, s_T; \lambda) && (1)\\
&= \sum_{s_T} f(\mathbf{o}_T | \mathbf{o}_1, \ldots, \mathbf{o}_{T-1}, s_T; \lambda) f(\mathbf{o}_1, \ldots, \mathbf{o}_{T-1}, s_T; \lambda) && (2)\\
&= \sum_{s_T} b_{s_T}(\mathbf{o}_T) \sum_{s_{T-1}} f(\mathbf{o}_1, \ldots, \mathbf{o}_{T-1}, s_{T-1}, s_T; \lambda) && (3)\\
&= \sum_{s_T} b_{s_T}(\mathbf{o}_T) \sum_{s_{T-1}} f(s_T | \mathbf{o}_1, \ldots, \mathbf{o}_{T-1}, s_{T-1}; \lambda) f(\mathbf{o}_1, \ldots, \mathbf{o}_{T-1}, s_{T-1}; \lambda) && (4)\\
&= \sum_{s_T} b_{s_T}(\mathbf{o}_T) \sum_{s_{T-1}} a_{s_{T-1} s_T} f(\mathbf{o}_1, \ldots, \mathbf{o}_{T-1}, s_{T-1}; \lambda) && (5)
\end{aligned}
$$

The recursive structure is revealed as we reduced the problem from needing $f(\mathbf{o}_1, \ldots, \mathbf{o}_T, s_T; \lambda)$ for all s_T to needing $f(\mathbf{o}_1, \ldots, \mathbf{o}_{T-1}, s_{T-1}; \lambda)$ for all s_{T-1}. Let us introduce the forward variable to ease the notation.

$$\alpha_1(s) \equiv f(\mathbf{o}_1, S_1 = s; \lambda) \tag{6}$$
$$= b_s(\mathbf{o}_1)\pi_s \tag{7}$$
$$\alpha_t(s) \equiv f(\mathbf{o}_1, \ldots, \mathbf{o}_t, S_t = s; \lambda) \tag{8}$$
$$= b_s(\mathbf{o}_t) \sum_{s'} a_{s's}\alpha_{t-1}(s') \tag{9}$$

Then our solution can be expressed nicely as

$$f(\mathbf{o}_1, \ldots, \mathbf{o}_T; \lambda) = \sum_s \alpha_T(s). \tag{10}$$

Implemented naïvely top-down (backwards in time) this would not bring us any luck because of the exponentially recursive structure. The naïve algorithm is however easily convertible to an efficient variant using dynamic programming where we calculate the forward variables bottom-up (forwards in time). We simply calculate $\alpha_t(s)$ for all states s, first for $t = 1$ and then all the way up to T. This way all the forward variables from the previous time step are readily available when needed.

2.3 The Baum-Welch algorithm

We want to find the parameters λ that maximize the likelihood of the observations. This will be used to train the hidden Markov model with speech signals. The Baum-Welch algorithm is an iterative expectation-maximization (EM) algorithm that converges to a locally optimal solution from the initialization values.

The M-step consists of updating the parameters in the following intuitive way:

$$\pi_s := \overline{\pi_s} = \frac{\text{expected number of times in state } s \text{ at } t = 1}{\text{expected number of times at } t = 1} \tag{11}$$
$$a_{ss'} := \overline{a_{ss'}} = \frac{\text{expected number of transitions from } s \text{ to } s'}{\text{expected number of transitions from } s} \tag{12}$$
$$\mu_s := \overline{\mu_s} = \text{expected observation when in state } s \tag{13}$$
$$\mathbf{\Sigma}_s := \overline{\mathbf{\Sigma}_s} = \text{observation covariance when in state } s \tag{14}$$

The E-step thus consists of calculating these expectations for a fixed λ. Let $V_s^{(t)}$ denote the event of transition from state s at time step t, and $V_{s,s'}^{(t)}$ the event of transition from s to s' at t. Then we calculate these expectations by using indicator functions and linearity of expectation.

$$\overline{\pi_s} = \mathbb{E}\{\mathbf{1}[V_s^{(1)}]\} = P(V_s^{(1)}) \tag{15}$$

$$\overline{a_{ss'}} = \frac{\mathbb{E}\{\sum_t \mathbf{1}[V_{s,s'}^{(t)}]\}}{\mathbb{E}\{\sum_t \mathbf{1}[V_s^{(t)}]\}} = \frac{\sum_t P(V_{s,s'}^{(t)})}{\sum_t P(V_s^{(t)})} \tag{16}$$

$$\overline{\mu_s} = \frac{\mathbb{E}\{\sum_t \mathbf{1}[V_s^{(t)}]\mathbf{o}_t\}}{\mathbb{E}\{\sum_t \mathbf{1}[V_s^{(t)}]\}} = \frac{\sum_t P(V_s^{(t)})\mathbf{o}_t}{\sum_t P(V_s^{(t)})} \tag{17}$$

$$\overline{\mathbf{\Sigma}_s} = \frac{\mathbb{E}\{\sum_t \mathbf{1}[V_s^{(t)}](\mathbf{o}_t\mathbf{o}_t^{\mathrm{T}} - \overline{\mu_s\mu_s}^{\mathrm{T}})\}}{\mathbb{E}\{\sum_t \mathbf{1}[V_s^{(t)}]\}} = \frac{\sum_t P(V_s^{(t)})\mathbf{o}_t\mathbf{o}_t^{\mathrm{T}}}{\sum_t P(V_s^{(t)})} - \overline{\mu_s\mu_s}^{\mathrm{T}} \tag{18}$$

Note that the non-italic T denotes transpose and has nothing to do with time.

To be able to calculate these probabilities we first introduce the backward variable which is very similar to the forward variable previously defined.

$$\beta_T(s) \equiv 1 \tag{19}$$

$$\beta_t(s) \equiv f(\mathbf{o}_{t+1}, \ldots, \mathbf{o}_T | S_t = s; \lambda) \tag{20}$$

$$= \sum_{s'} a_{ss'} b_{s'}(\mathbf{o}_{t+1}) \beta_{t+1}(s') \tag{21}$$

The backward variable has its name because it is first calculated for the last time step and then backwards in time when implemented with dynamic programming (essentially the reverse procedure of the one described in detail for the forward variable).

Then we rename the probabilities to the same symbols as used by Rabiner and express them by forward and backward variables:

$$\gamma_t(s) \equiv P(V_s^{(t)}) = P(S_t = s | \mathbf{o}_1, \ldots, \mathbf{o}_T; \lambda) \tag{22}$$

$$= \frac{f(\mathbf{o}_1, \ldots, \mathbf{o}_T | S_t = s) P(S_t = s)}{f(\mathbf{o}_1, \ldots, \mathbf{o}_T)} \tag{23}$$

$$= \frac{f(\mathbf{o}_1, \ldots, \mathbf{o}_t, S_t = s) f(\mathbf{o}_{t+1}, \ldots, \mathbf{o}_T | S_t = s)}{f(\mathbf{o}_1, \ldots, \mathbf{o}_T)} \tag{24}$$

$$= \frac{\alpha_t(s)\beta_t(s)}{f(\mathbf{o}_1, \ldots, \mathbf{o}_T)} \tag{25}$$

And similarly (details omitted):

$$\xi_t(s, s') \equiv P(V_{s,s'}^{(t)}) = P(S_t = s, S_{t+1} = s' | \mathbf{o}_1, \ldots, \mathbf{o}_T; \lambda) \tag{26}$$

$$= \frac{\alpha_t(s) b_{s'}(\mathbf{o}_{t+1}) a_{ss'} \beta_{t+1}(s')}{f(\mathbf{o}_1, \ldots, \mathbf{o}_T)} \tag{27}$$

And finally we get the following:

$$\overline{\pi_s} = \gamma_1(s) \tag{28}$$

$$\overline{a_{ss'}} = \frac{\sum_t \xi_t(s,s')}{\sum_t \gamma_t(s)} \tag{29}$$

$$\overline{\mu_s} = \frac{\sum_t \gamma_t(s)\mathbf{o}_t}{\sum_t \gamma_t(s)} \tag{30}$$

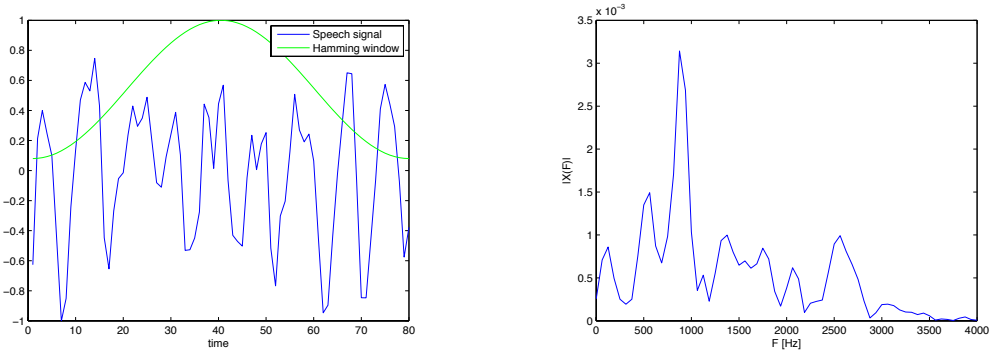
$$\overline{\mathbf{\Sigma}_s} = \frac{\sum_t \gamma_t(s)\mathbf{o}_t\mathbf{o}_t^{\mathrm{T}}}{\sum_t \gamma_t(s)} - \overline{\mu_s\mu_s}^{\mathrm{T}} \tag{31}$$

To summarize the E-step boils down to computing $\gamma_t(s)$ and $\xi_t(s,s')$ for all s, s' and t while the parameters λ are fixed, and then the M-step will update λ by using the calculations done in the E-step. This is iterated until satisfaction.

3 System design

3.1 Feature extraction

The source speech is sampled at 8000 Hz and quantized with 16 bits. The signal is split up in short frames of 80 samples corresponding to 10 ms of speech. The frames overlap with 20 samples on each side. The idea is that the speech is close to stationary during this short period of time because of the relatively limited flexibility of the throat. We will pick out our features from the frequency domain, but before we get there by taking the fast Fourier transform, we multiply by a Hamming window to reduce spectral leakage caused by the framing of the signal.

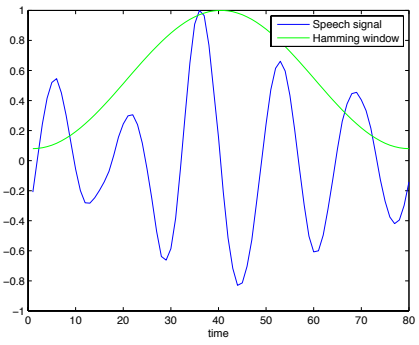


(a) Speech signal and Hamming window in time domain.

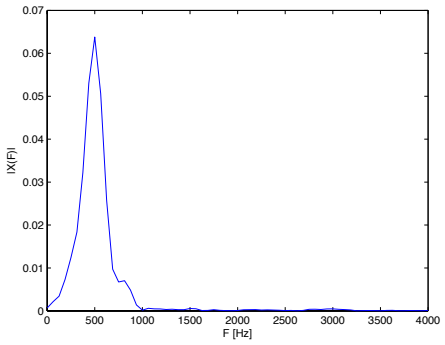
(b) Single-sided magnitude spectrum of the same speech signal multiplied by the Hamming window.

Figure 2: An 80 sample frame of an unvoiced part of a speech signal. Unvoiced speech, like 'sh', is more noisy and contains higher frequencies than voiced speech.

The D largest local maxima from the single-sided magnitude spectrum are are picked as features for each frame, and D is indeed an important parameter of the system that will be discussed later.



(a) Speech signal and Hamming window in time domain.



(b) Single-sided magnitude spectrum of the same speech signal multiplied by the Hamming window.

Figure 3: An 80 sample frame of a voiced part of a speech signal.

3.2 Training

The training is a combination of both supervised and unsupervised techniques. We train one hidden Markov model per word with already classified speech signals. One important choice is the number of different states in each model. The goal is that each state should represent a phoneme in the word. The clustering of the Gaussians is however unsupervised and will depend on the initial values used for the Baum-Welch algorithm.

For this project, totally random guesses (that obey the statistical properties) for $\mathbf{A}$ and π were used as initial values. For $\mathbf{\Sigma}_s$, the diagonal covariance matrix for the training data was used for all states. For each state a random training data point was chosen as μ_s. The training examples for each word are concatenated together, and Baum-Welch is run for 15 iterations.

3.3 Classification

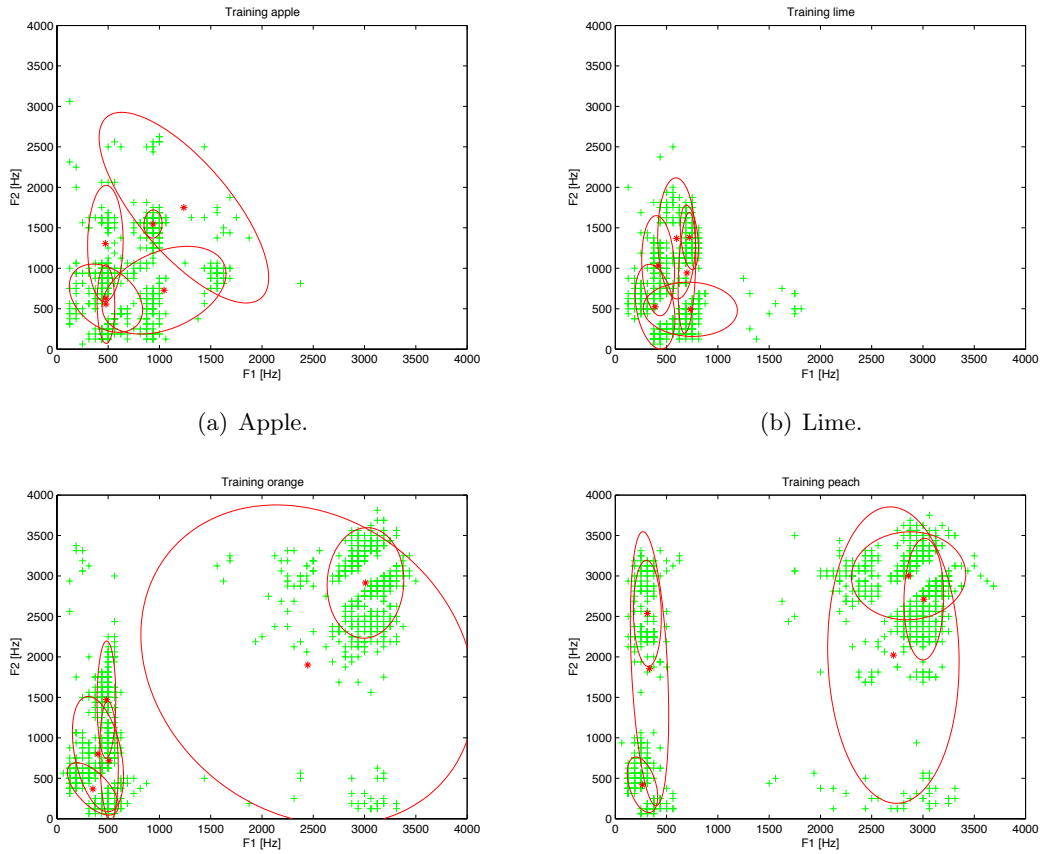
Let λ_i denote the parameter set for word i. When presented with an observation $\mathbf{o}_1, \ldots, \mathbf{o}_T$, the selection is done as follows.

$$\text{predicted word} = \arg\max_i f(\mathbf{o}_1, \ldots, \mathbf{o}_T; \lambda_i) \tag{32}$$

And we recognize that $f(\mathbf{o}_1, \ldots, \mathbf{o}_T; \lambda_i)$ is exactly what the forward algorithm computes.

4 Experimental setup and results

For each of the seven words, 15 utterances by this author were recorded. The performance of the system was measured by five-fold cross-validation on the recorded data set of 105 utterances. Experimentation indicated that the two most important parameters



(a) Apple.

(b) Lime.

(c) Orange.

(d) Peach.

Figure 4: Fitted Gaussians after ten iterations of the Baum-Welch algorithm. We have six states with one Gaussian each. The two most dominant frequencies (features) are shown. Each green plus is represents a frame from a training speech signal. The stars are the means of each Gaussian, and the ellipses indicate their 75% confidence interval. Notice the higher frequencies present in the words containing unvoiced phonemes ('peach' and 'orange') compared to the words that do not ('apple' and 'lime').

were the number of hidden states, N, and the number of frequencies extracted from each frame, D. The cross-validation was therefore run with different values for these parameters, and the results are shown in table 1.

5 Discussion

The results are quite good compared to the simple approach taken, especially in the feature extraction phase. More advanced features like Mel-frequency cepstral coefficients were considered, but we decided on simple frequencies due to the low misclassification

$N \backslash D$	2	3	4	5	6	7	8
2			21.9%		8.6%		
3	21.0%	15.2%	9.5%	12.4%	1.9%	14.3%	5.7%
4	16.2%	11.4%	8.6%	5.7%	3.8%	6.7%	4.8%
5	13.3%	8.6%	9.5%	4.8%	2.9%	5.7%	4.8%
6	12.4%	10.5%	3.8%	5.7%	7.6%	6.7%	10.5%
7	15.2%	12.4%	6.7%	10.5%	7.6%	2.9%	8.6%
8			12.4%			5.7%	

Table 1: Misclassification rates for five-fold cross-validation with different values for the number of hidden states, N, and the number of frequencies extracted from each frame, D. Each five-fold cross-validation procedure takes about 7 minutes with the 105 utterances on a 2 GHz Intel Core 2 Duo (serial execution).

rates achieved. It should be noted that this system would not perform well if trained and tested with different speakers. This is because of the different frequency characteristics of different voices, especially for speakers of different gender.

We also experimented with increasing the number of training iterations for the Baum-Welch algorithm, including setting a threshold on the likelihood difference between steps. That, however, proved to have little benefit in practice; neither the execution time nor the misclassification rate showed any mentionable improvements over just fixing the number of iterations to 15. The reason why the execution time did not show any significant improvements is because most of the execution time is spent during feature extraction, and not in training.

It is also interesting to note that when N is too small, there are many 'apple's misclassified as 'pineapple's, and vice versa, due to the loss of temporal information.

Another important parameter is the number of samples in each frame. If the frame is too small, it becomes hard to pick out meaningful features, and if it is too large, temporal information is lost. However, due to time constraints, we did not test anything else than 80 samples for this project.

The concatenation of the training examples trains a probability of transitioning from the 'last state' to the 'initial state' that is not needed for classification. Rabiner gives a modified Baum-Welch algorithm for multiple training examples such that concatenation is not necessary, but that was not implemented during this project as the concatenation seemed to work well.

6 Conclusion and future work

During this project a system for isolated-word speech recognition was implemented and tested. The cross-validation results are good for a single speaker. Two obvious extensions are better support for several speakers, and support for continuos speech. The first step towards the former would be more, and more robust, features. For the latter the simplest approach is probably to detect word boundaries and then proceed with an isolated-word

recognizer.

The Matlab implementation along with the data set is published as open source and can be found at `http://code.google.com/p/hmm-speech-recognition/`.

7 References

[1] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, pp. 257–286, Feb 1989.

[2] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1st ed. 2006. corr. 2nd printing ed., October 2007.

[3] X. Huang, A. Acero, and H.-W. Hon, *Spoken Language Processing: A Guide to Theory, Algorithm and System Development*. Prentice Hall PTR, May 2001.