

IT3105 AI Programming: Speech Recognition

Lecture Notes

Helge Langseth, IT Vest 310

Version 1.04 – April 13, 2011

This note aims at introducing the required background for building a basic Speech Recognition System (SRS). The topics that will be covered are *i*) basic properties of speech, and *ii*) a brief recap of how to build probabilistic temporal models, in particular Hidden Markov Models (HMMs) with the associated inference and learning algorithm.

The “classic” reference for everybody working with probabilistic speech recognition, and also a very important reference for this project, is *Lawrence R. Rabiner: “A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition”*, which can be downloaded from the course’s webpage. The paper gives a broad account of probabilistic speech recognition, with focus on the internals of HMM inference and learning. It contains more information than we will need in this project, but if you read (and understand!) pp. 257–266 (until “IV. Types of HMMs”), Subsection “IV.A. Continuous observation densities in HMMs” (p. 267) and Subsection “V.A Scaling” (pp. 272–273) you should be well covered for now. The main goal of this note is thus to get you up to speed so that you can understand the paper by Rabiner, and implement the techniques discussed there.

Those of you who have taken the “Methods in AI”-course should already be familiar with the HMMs, but if you have forgotten all about it, I recommend to look at Chapter 15 of *Russell & Norvig: “Artificial Intelligence, A Modern Approach”* again, as R&N give a good and fairly readable account of these models. Be warned that the notation and terminology differ somewhat between R&N’s book and Rabiner’s paper, though. Note also that when R&N’s Section 15.6 describes the practicalities of how to build an SRS, the approach taken there is a bit more labour-intensive than the one we will pursue, and I therefore recommend to not confuse yourself by reading R&N’s Section 15.6 now. If you are not familiar with HMMs I recommend to read Russell&Norvig, Chapters 13–15 before continuing on this note.

In addition to the already mentioned reference, there is a wealth of information about SRSs available online, including a number of fancy demonstrations that you may find motivating to take a look at.

In this project we will make an SRS that recognises *single-word-speech*, and we will assume that the speech stems from a limited vocabulary. Compared to what the state-of-the-art SRSs today can do, this is a very simplified problem. The reason for choosing to consider only a stripped-down version of the general speech recognition problem is that we can now avoid looking at refined acoustic models and language grammars; both of which constitute important parts of “real” SRSs. Basically, this means that our approach will be *data intensive*, and we will not spend time on the linguistic side of things. However, we will still meet some of the same challenges as state-of-the-art SRSs have to tackle, and I therefore hope that the problem will be an interesting challenge even though it is a bit simplified.

1 Sound signals and how to represent them

Let us start by looking at how speech is created. We focus attention to *vocalisation*, meaning sounds made by humans using the mouth, nose, and throat. A vocalisation is made up by a series of *utterances*; one utterance being the stream of speech between two periods of silence. As we are concerned by speech of single words, an utterance given to the SRS will always contain exactly one word (and separation of words is therefore not required). When a word is spoken, the task of generating the speech is divided into generating the separate *phonemes*. People do this by using lips, tongue, jaw, vocal cords, and lungs. When we move the lungs we create an air flow through the other articulators, which again makes the sound. As the air flows past the vocal cords, tension applied to the vocal cords causes the creation of a tone or pitch. Sound is transported through air as variations in pressure, and this is what is detected by the human ear. Humans can distinguish sounds in the range from 30 Hz to about 20 kHz, with best results for sounds at about 1 kHz.

Figure 1 plots some example utterances. The amplitude of the speech signal is plotted as a function of time, and the topmost plot shows the speech signal representing the vocalisation of “Start [pause] Stop [pause] Left [pause] Right”. The four plots below give the speech signal for each utterance separately. The scale on the x -axis is time in seconds, the y -axis gives the amplitude of the signal after normalisation. The important thing to take home from Figure 1 is that speech signals are *dynamic* – they change over time, and to be able to recognise one example of speech as being an utterance of a particular word, we must take this dynamic nature of the speech signal into account when we build our SRS.

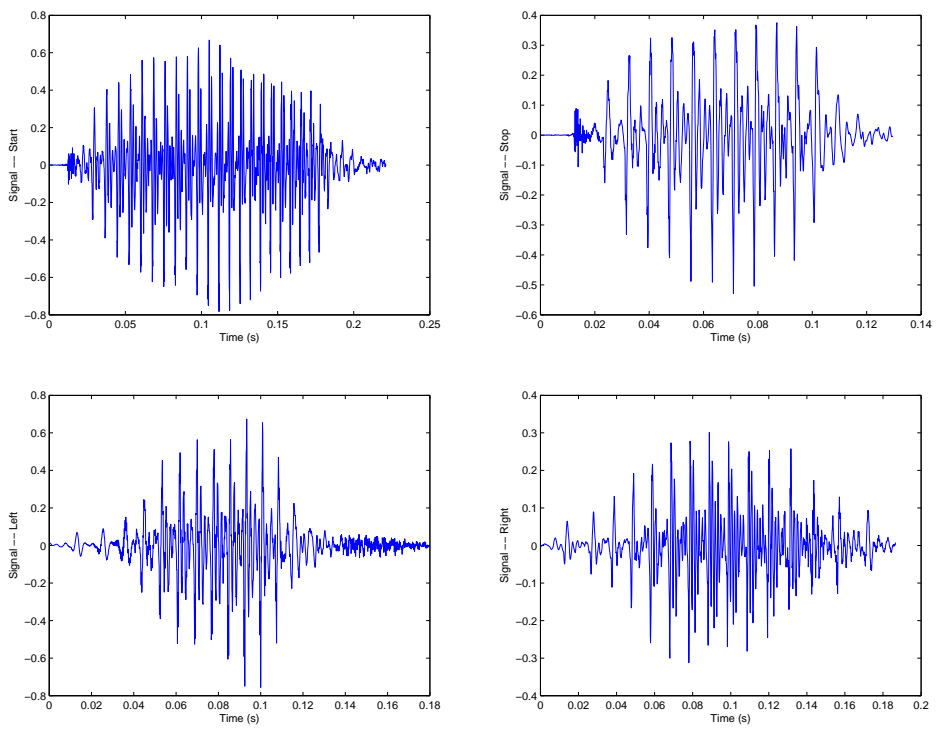
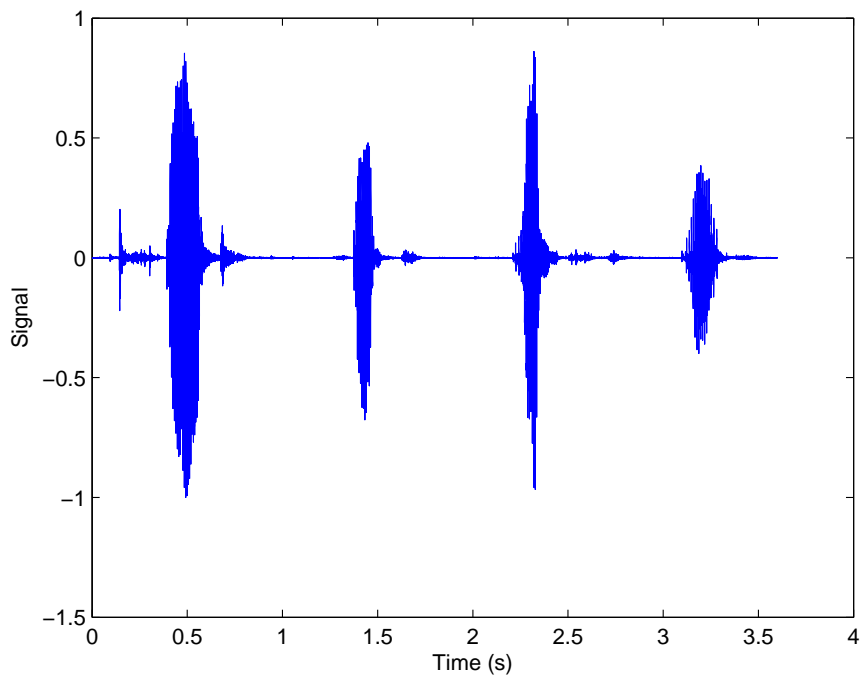


Figure 1: This is the raw signal of the the utterance of the four words “Start”, “Stop”, “Left”, and “Right”.

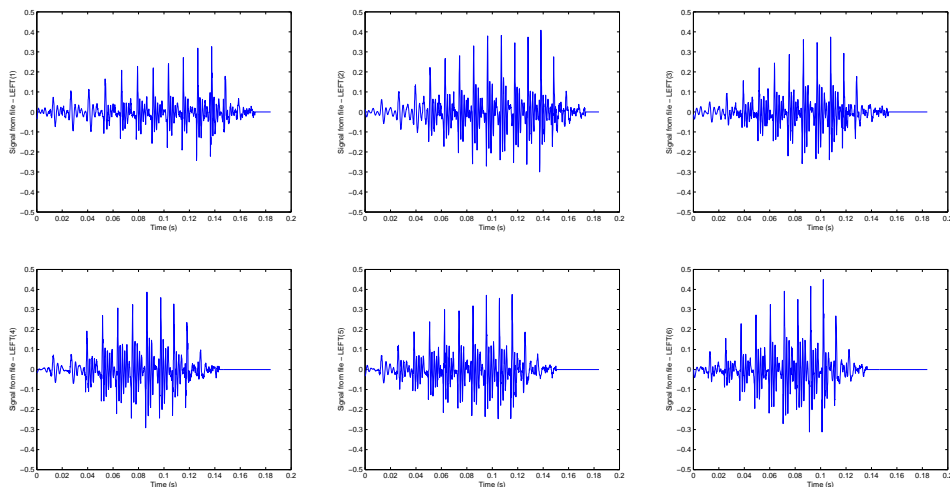


Figure 2: Several utterances of the word “Left”.

The representation of different words in Figure 1 look rather different, and this may give us the idea to use the sound signal itself as our representation of an utterance. Recognition would in this case amount to comparing a new speech signal to prototypical examples of earlier recorded speech, and classify the new utterance to the class being most “similar” in these terms. Before ascribing to that strategy, though, we must investigate the *robustness* of the speech signal representation. We say that a speech representation is robust if it is (almost) identical each time the same word is spoken. If we have a robust speech representation, this will obviously be beneficial when the speech is to be recognised. Consider Figure 2, where six utterances of the word “Left” are shown. Although the speech examples come from the same speaker, there are differences in the maximum amplitude and the “overall look” of the signals; symptoms of lacking robustness in this representation.

We investigate this further in Figure 3, where the (time-dependent) average signal of 35 utterances of the word “Left” is shown, together with the time-dependent 95% confidence interval of the 35 signals. As we see, the structure in the signal is completely overshadowed by the variability, and the raw signal thereby appears unsuitable as a representation in the SRS. We therefore proceed by trying to “robustify” the signal; a process described in Figure 4. At the beginning, the speech signal is continuous in time as shown in the topmost part of the figure. This signal is fed to the computer via a microphone, which captures the pressure wave of the original speech and translates it into an analogue electrical signal. At that point the sound is a real-valued function of continuous time, and the signal can in principle be saved exactly as it was using a tape system. However, although we have deliberately disregarded this problem so far, it is not feasible

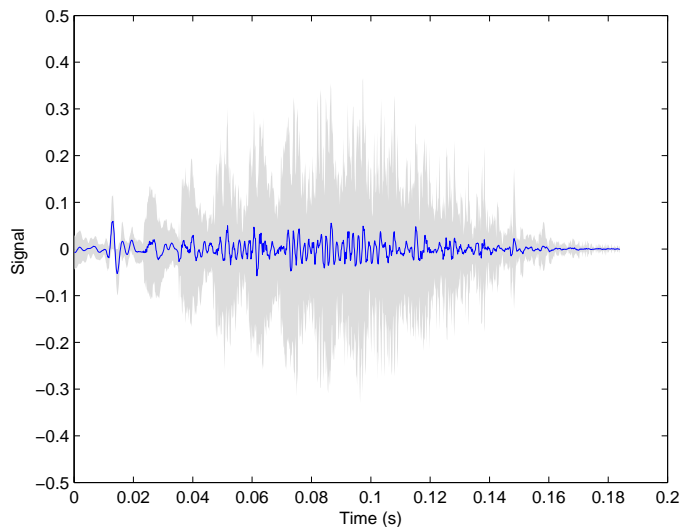


Figure 3: The (point-wise) mean signal and 95% confidence interval for 35 utterances of the word “Left”.

for a computerised system to work with a continuous-time signal. Rather, the SRS is given sound sampled in discrete time. Typically a speech signal is represented using between 8000 and 48000 real values per second (i.e., the signal is sampled with between 8 kHz and 48 kHz) where the low end is the frequency used in a telephone, the high end is the frequency used in digital audio tape). The sampling frequency we will work with in this project is 8 kHz. The discrete time signal is shown in the middle part of the figure.

Empirical studies have shown that important changes in the *content* of speech happen at no more than 100 Hz, meaning that we can “down-sample” the 8 kHz signal by a factor of 80 without losing important information. As indicated at the bottom of Figure 4, we do so by dividing the signal into partly overlapping intervals, called *frames*. One frame lasts 1/100th a second, and is therefore both sufficiently long for being interesting for analysis (in our case it will contain $8000/100 = 80$ values) as well as sufficiently frequent to capture the most interesting content changes in the speech signal.

A naïve approach to utilise the “framing” to robustify the speech signals is shown in Figure 5, where the signal average is calculated inside each frame as a proxy for the more variable raw signal. The same six examples of speech that were used in Figure 2 are also employed here. Unsurprisingly, the results in Figure 5 are more visually pleasing than the ones in Figure 2 (since much of the high-frequency “noise” is removed), but the results are still not convincing. The conclusion is that both the raw signal as well as the framed signals contain too much variability

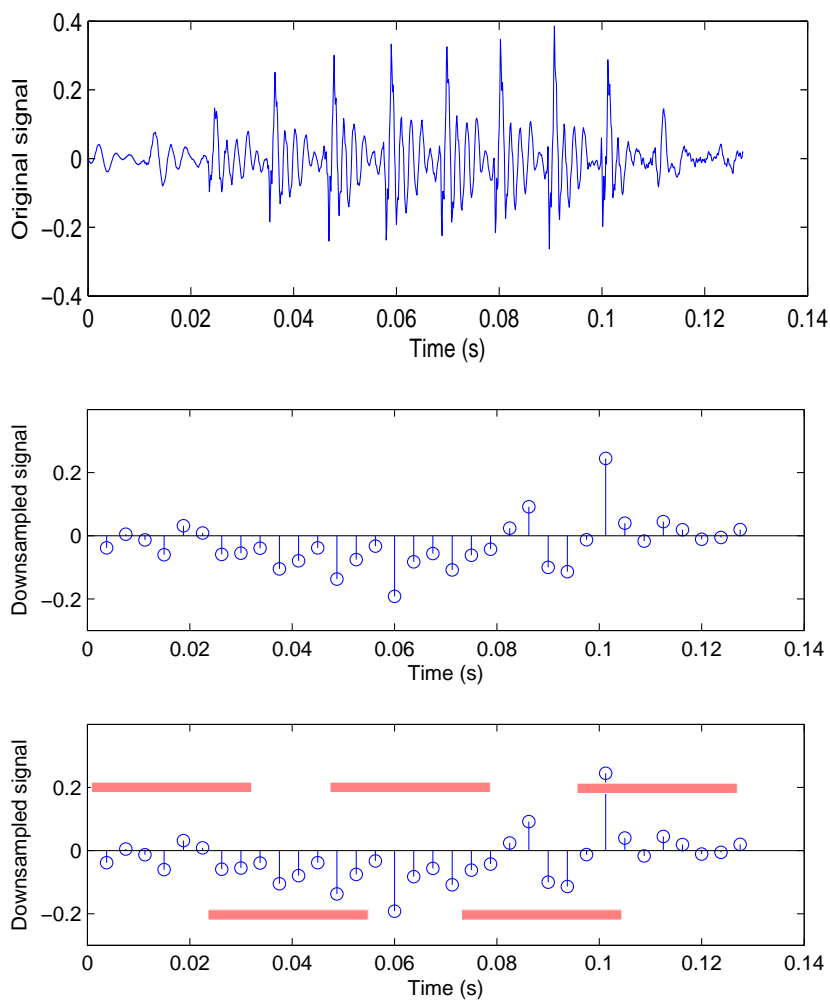


Figure 4: The continuous-time signal (on top) is represented in discrete time in the SRS. Typically, the speech signal is represented using between 8000 and 48000 real values per second (i.e., the signal is sampled with between 8 kHz and 48 kHz), as shown in the middle plot. This discrete-time sound signal is not robust, and should therefore be handled further. Typically, one makes so-called *frames* (a partly overlapping segmentation of the sound file), where each frame covers about 1/100th of a second of speech.

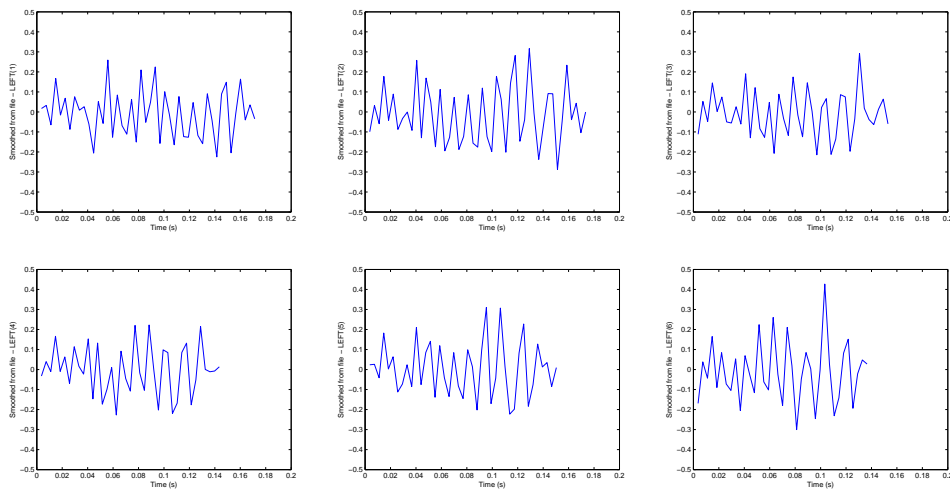


Figure 5: Six different utterances of the word “Left” by the same speaker. The raw data is smoothed by framing (using a Hamming window).

to be useful as input to a SRS.

A number of different ways to encode the speech signal exist, the most popular one among current SRSs is (apparently) the “Mel Frequency Cepstrum Coefficients” (MFCCs). MFCC is related to the Fourier coefficients, and to not complicate too much, we will start out using Fourier coefficients instead of MFCCs in this project. (However, if you fail to obtain results at the level you think is reasonable to expect, you may of course look more into MFCCs on your own.)

As you (hopefully) recall, the Fourier transform of a signal takes the signal from the *time domain* into the *frequency domain*. This means that our original signal is a function of time, but the result of the Fourier transform is a function of *frequencies*. Let us call the original signal $y(t)$ where t denotes time. For simplicity we assume that $t = 1, \dots, T$, meaning that the observation is T items long. We call the Fourier transform of the signal $Y(\omega)$ where ω denotes frequencies.¹

Formally, the discrete Fourier transformation of $y(t)$ is defined as

$$Y(\omega) = \sum_{t=0}^T y(t) \exp(-2\pi i \cdot \omega t / T),$$

so we can calculate $Y(\omega)$ for any ω that we desire (although the length of signal and the sampling frequency bound which frequencies it makes sense to consider).

¹I assume you are “familiar” with Fourier analysis. If you want a nice and very informal recap of the main ideas, you can for instance take a look at Open University’s video-lectures on the topic. They are available for free in the iTunes store; search for “open university Fourier” to locate this resource.

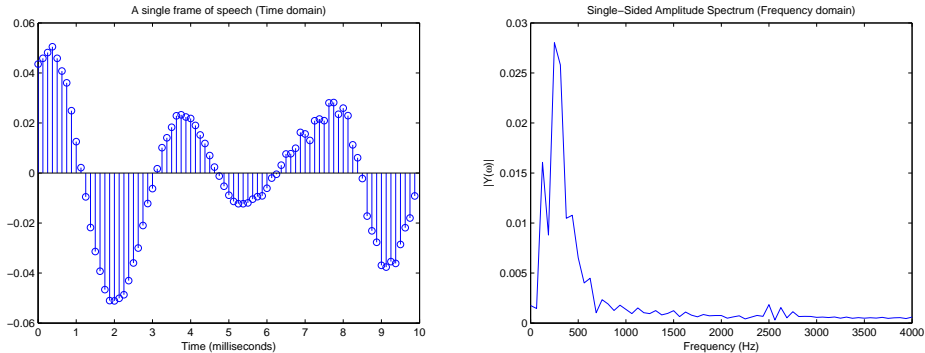


Figure 6: A single frame of speech (left panel), and the spectral density representation in the frequency domain (right panel). We emphasise that the sound signal is discrete-time, thus it is the discrete-time Fourier transform that is employed.

The Fourier transform has been used to generate Figure 6. The left part of the figure shows one frame of a speech signal in the time-domain, and the right-hand side represents the Fourier-transform of the signal. Note that since $Y(\omega)$ is a complex number, it is rather $\frac{|Y(\omega)|}{2\pi}$, called the spectral density, which is displayed. We interpret $|Y(\omega)|/(2\pi)$ as the contribution to the energy of the signal at frequency ω , and define the energy of the signal as $\frac{\sum_{\omega} |Y(\omega)|}{2\pi}$.

A common way of visualising a speech segment is by the so-called *spectrogram*. The spectrogram is found by segmenting the sound signal into frames, and for each frame to calculate the spectral density. We may think of a spectrogram as a stack of images like in the right panel of Figure 6. Hence, the result is a two-dimensional grid, where one dimension represents the time and the other frequency. The spectral density is shown using colour-codes. Figure 7 shows the spectrogram for the same sound files as were investigated in Figure 5. We see that the spectrograms appear to be almost identical for the different utterances, i.e., it appears to be a rather robust representation of speech.

As may have become apparent from the discussion so far, selecting an appropriate representation for speech in the SRS is not an easy task. We have seen results indicating that the spectral density is robust, and Figure 8 also suggests that the spectral density has some discriminative power: The figure shows the spectrogram for the utterance of the four words also used in Figure 1, and we can see that the different words appear differently in this representation. However, there are still open questions: Which frequencies are most informative? Should we just find a handful of frequencies that maximize $|Y(\omega)|$ or are we more interested in finding the “bumps” in the spectrum? Is the raw signal informative *in combination* with spectral information? There are also other short-time measurements that people

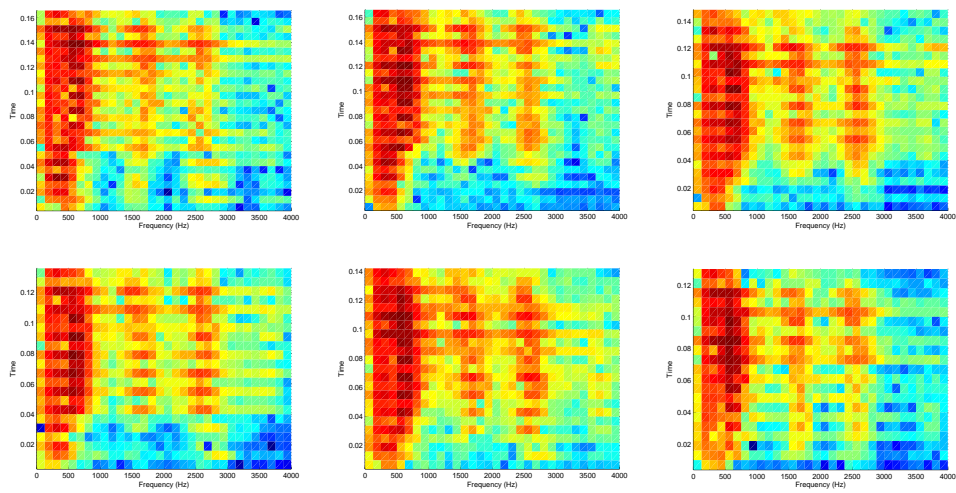


Figure 7: Six different utterances of the word “Left” by the same speaker. The plots give the spectrograms of the speech signals. Compare also to Figure 5.

use either together with or instead of spectral information. Examples of such measurements include

- The log-energy of the signal inside a frame;
- The average magnitude;
- The zero crossing-frequency.

In the first part of the project you will be asked to investigate the different representations further, and either qualitatively or quantitatively evaluate what representation is most useful.

2 Probabilistic models over time

2.1 Hidden Markov models

In the previous section we discussed how speech sounds are created, and how they can be represented for classification purposes. Let us look at this some more, but now with the purpose of *building a model* that can be used for classification.

By inspecting the sound signals, we have established two major facts:

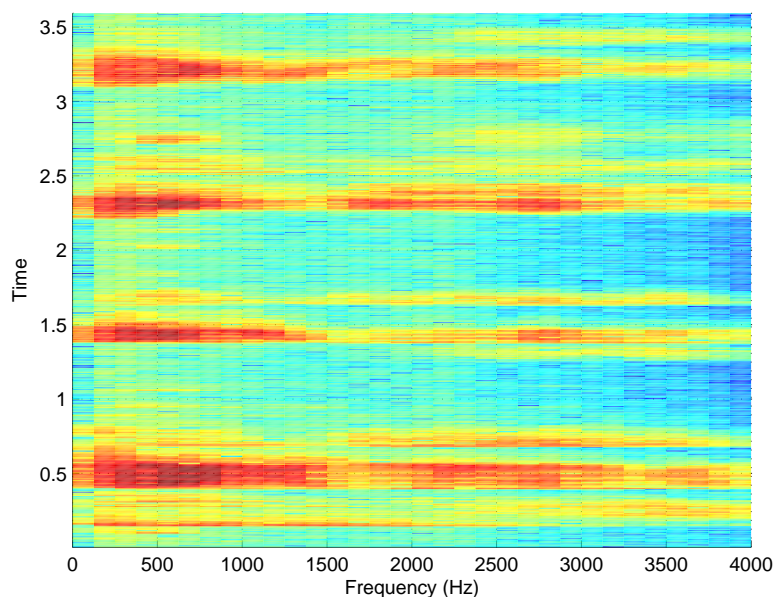


Figure 8: This is the spectrogram of the utterance of the four words “Start”, “Stop”, “Left”, and “Right”.

- Sound signals are *time variant*, so if we want to build an SRS we need a model that takes this dynamic aspect into account.
- Sound signals are noisy (literally): When a single speaker repeats the same word several times, the representations of these sounds may look quite different from each other, even if we put effort into making the representation robust. It is therefore imperative to use a modelling framework that is able to handle uncertainty efficiently.

In conclusion, the best-working SRSs around use probabilistic frameworks for modelling of data generated over time, i.e., a *time-series model*. This may be a surprise to you, as you may have thought about comparing representations of utterances like Figure 7 directly. Unfortunately, though, it is very difficult to compare spectrograms, for instance if different utterances of the same word differ in length. Therefore, we will do as the state-of-the-art systems, and use a *dynamic Bayesian network* as our representation. The class of dynamic Bayesian networks is rather broad, and by formalising some assumptions we can better choose one particular set of models, namely the *Hidden Markov Models*.

Let us start by looking at the speech generating process, where the speaker utters a sequence of phonemes, that together constitute a word. Let us assume for a

while that we are able to observe the phonemes directly, and denote by S_t the phoneme the user is uttering inside frame t . So, S_t is a single, discrete variable, and we will assume its domain to be N labelled states, meaning that S_t takes on one of the values in the set $\{1, 2, \dots, N\}$ at time t . An observation of a word would then be $\{s_1, s_2, \dots, s_T\}$, and we will use $\mathbf{s}_{1:T}$ as a shorthand for an observation of length T . If we want to describe formally the sequence of random variables $\{S_1, S_2, S_3 \dots\}$, we may find it suitable to make some assumptions:

1st-order Markov process: The phone uttered at time t obviously depends on the phonemes of earlier time-points (since they together will constitute a word). To simplify the calculation burden it is common to assume that this dynamics is appropriately described as a *1st order Markov process*, meaning that $P(S_t | \mathbf{S}_{1:t-1}) = P(S_t | S_{t-1})$. (This may not be totally true, but is assumed for convenience).

Stationary process: The dynamic model (aka. *transition model*), $P(S_t | S_{t-1})$, does *not* depend on t .

The dynamic model, $P(S_t | S_{t-1})$ is defined by the transition matrix, \mathbf{A} , where the matrix has the interpretation that $a_{ij} = P(S_t = j | S_{t-1} = i)$. Thus, \mathbf{A} is square, and has size $N \times N$. The model also must define in which state the sequence starts, and this is done by defining a distribution over $P(S_1)$. This distribution is called the *prior* distribution, and is denoted by $\boldsymbol{\pi}$, with $\pi_j = P(S_1 = j)$.

In all, this model is known as a Markov model, and is depicted in Figure 9. The left panel shows the Markov model as a Bayesian network, the right panel shows a different perspective, where each *state* of the variable is drawn as a circle. In this example model we have used $N = 2$ states. There are arrows between the states, and they are labelled with probabilities. The way to read this figure is to assume that at the beginning ($t = 1$) you are in one state (say, the first one for the sake of the argument, i.e., $s_1 = 1$, but this will be defined by the prior distribution). At each discrete time point ($t = 2, t = 3$, etc.), you will travel from the state you are in to one of the states that are reachable from the current state following an arrow. The probability that each arrow is chosen is defined by its label, so there is a probability a_{ij} to move to state j when being in state i . As some arrows go from one state and back to the same state, it is possible to remain in the same state over time (which is useful if it takes more than one frame to utter a particular phoneme). We mention both these representations here, because the former representation is used in the book by Russell&Norvig (and therefore also in the course “Methods in AI”), whereas the latter was used by Rabiner in his paper. As you see, they are interchangeable, and use the same parameterization.



Figure 9: Two different ways of presenting a Markov model.

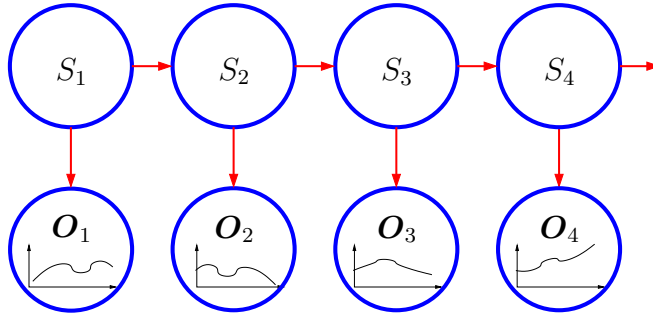


Figure 10: The HMM model structure shown as a (dynamic) Bayesian network. The variables S_t are discrete and one-dimensional, the variables \mathbf{O}_t are vectors of continuous variables used to represent the sound signal in a that frame.

Unfortunately, phonemes are not observable themselves. What we get instead is the phoneme partly disclosed by the sound observation from frame t (represented as discussed in the previous section). We will denote the representation of that sound-observation \mathbf{O}_t . This limited observability complicates matters slightly, but not as much as one could fear given that some assumptions are made:

Sensor Markov assumption: The observation we get at time t only depends on what is uttered at that point in time, $P(\mathbf{O}_t | \mathbf{S}_{1:t}, \mathbf{O}_{1:t-1}) = P(\mathbf{O}_t | S_t)$.

Stationary process for sensors: The observation model $P(\mathbf{O}_t | S_t)$ does not depend on t .

Accidentally, these assumptions together are exactly the assumptions underlying the *Hidden Markov Models*, and we shall therefore use the HMMs as the main reasoning engine for this project. Figure 10 shows the structure of a Hidden Markov Model. The interpretation of the model is related to how we assume that a single word is uttered: S_t is the unobserved phoneme the speaker is uttering in time frame t , and \mathbf{O}_t is the observation of that phoneme inside frame t .

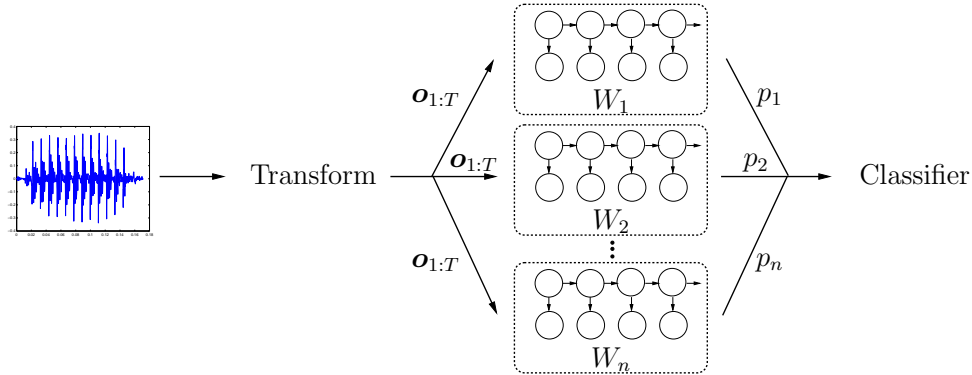


Figure 11: The top-level structure for the classifier has one HMM per word. Note that the same data is sent to all models, and that the probability $p_j = P(\mathbf{o}_{1:T}|W_j)$ is returned from the models.

2.2 Inference in HMMs

Let us assume that we have an observation of speech in our selected representation running over T frames. We call the observation in time-frame t \mathbf{o}_t , so the whole sequence is $\mathbf{o}_{1:T} = \{\mathbf{o}_1, \dots, \mathbf{o}_T\}$. Now the question is: “What can we do with this observation if we have a fully specified HMM model W to go with it?” The answer to this is rather complex, but the main thing we are interested in is to calculate the probability for this particular observation given the model, i.e., $P(\mathbf{o}_{1:T}|W)$. This is useful, because if a single model W as the one in Figure 10 is used to describe utterances of *one particular* word, the probability $P(\mathbf{o}_{1:T}|W)$ tells the probability for $\mathbf{o}_{1:T}$ to be observed if W represents the word that was spoken. Using Bayes rule, we can get an even more interesting value, namely the probability that W represents the spoken word given the produced observation:

$$P(W|\mathbf{o}_{1:T}) = \frac{P(\mathbf{o}_{1:T}|W) \cdot P(W)}{P(\mathbf{o}_{1:T})} \propto P(\mathbf{o}_{1:T}|W) \cdot P(W).$$

Here, $P(W)$ is the probability for hearing W before the speech starts, i.e., some notion of the frequency that the SRS hears this word with. Now, if we have one HMM per word, we can simply calculate $P(\mathbf{o}_{1:T}|W) \cdot P(W)$ in each model, and classify a sound signal to the word getting the highest score. The overall process is shown in Figure 11, where p_j is used as a shorthand for $P(\mathbf{o}_{1:T}|W_j) \cdot P(W_j)$.

The mechanisms for calculating $P(\mathbf{o}_{1:T}|W)$ is described in detail by Rabiner pp. 262–263, but the inference can be simplified a bit using matrix manipulations. The main trick is to focus on calculating $P(\mathbf{o}_{1:t}, S_t = j)$, a quantity he called $\alpha_t(j)$; we will use $\boldsymbol{\alpha}_t$ as a shorthand for the vector $[\alpha_t(1) \ \alpha_t(2) \ \dots \ \alpha_t(N)]^T$. Looking at $\boldsymbol{\alpha}_t$ is clever in two ways: Firstly, let us consider how we can calcu-

late α_{t+1} when α_t is known. To this end, Rabiner describes what is called the FORWARD ALGORITHM:

1. Initialisation: $\alpha_1(j) = \pi_j b_j(\mathbf{o}_1)$.
2. Induction: $\alpha_{t+1}(j) = \left[\sum_{i=1}^N \alpha_t(i) a_{ij} \right] b_j(\mathbf{o}_{t+1})$

Here, $b_j(\mathbf{o}_t) := P(\mathbf{o}_t | S_t = j)$ is the probability for observing \mathbf{o}_t given that the phoneme uttered in frame t is number j .

We can ease the implementation of this by first defining \mathbf{B}_t to be a matrix of size $N \times N$. \mathbf{B}_t has all values equal to zero except on the diagonal, where element (j, j) is $P(\mathbf{o}_t | S_t = j)$. Then, the induction step above is easier implemented as the matrix operation

$$\alpha_{t+1} = \mathbf{B}_{t+1} \mathbf{A}^T \alpha_t.$$

Hence, we can calculate the α_T -values very efficiently, using only $O(T)$ operations.

The other clever thing about looking at α is that Rabiner proposes to calculate the probability we are looking for by using that $P(\mathbf{o}_{1:T} | W) = \sum_{i=1}^N \alpha_T(i)$.

Although that equation is obviously true, it can be hard to take advantage of in practice. The reason is that due to numerical instability, the values in α_T may become very small as T increases, and eventually $P(\mathbf{o}_{1:T} | W)$ may be calculated as zero although that is not correct. To alleviate the problem, it is popular to focus on calculating $\log(P(\mathbf{o}_{1:T} | W)) = \log\left(\prod_{t=1}^T P(\mathbf{o}_t | \mathbf{o}_{1:t-1}, W)\right) = \sum_{t=1}^T \log(P(\mathbf{o}_t | \mathbf{o}_{1:t-1}, W))$, which means that we must calculate $P(\mathbf{o}_t | \mathbf{o}_{1:t-1}, W)$ for $t = 1, \dots, T$. This can be obtained fairly simply, by making some small adjustments to the above algorithm. In the following we focus on $P(S_t = j | \mathbf{o}_{1:t})$, and call this number $f_t(j)$. Note that $f_t(j) = \alpha_t(j) / \sum_j \alpha_t(j)$, so \mathbf{f}_t is a normalised version of α_t . The normalisation is useful for two reasons, firstly, it helps us avoid the problems with numerical instability that we would otherwise observe for large T , and secondly, the normalisation constant at time-step t can be shown to be equal to $P(\mathbf{o}_{t+1} | \mathbf{o}_{1:t})$, which is the number we are interested in!

Initialisation:

- $\mathbf{f}'_1 = \mathbf{B}_1 \boldsymbol{\pi}$
- $\ell_1 = \sum_{j=1}^N \mathbf{f}'_1(j)$
- $\mathbf{f}_1 \leftarrow \mathbf{f}'_1 / \ell_1$.

Induction: Do for $t = 1, \dots, T - 1$:

- $\mathbf{f}'_{t+1} = \mathbf{B}_{t+1} \mathbf{A}^T \mathbf{f}_t$

- $\ell_{t+1} = \sum_{j=1}^N \mathbf{f}'_{t+1}(j)$
- $\mathbf{f}_{t+1} \leftarrow \mathbf{f}'_{t+1} / \ell_{t+1}$.

Termination: Return $\log(P(\mathbf{o}_{1:T}|W)) = \sum_{t=1}^T \log(\ell_t)$.

You will also have to choose a family of distributions for your observations given utterance, i.e., the probability distribution for $P(\mathbf{o}_t|S_t = j)$. Recall that if $P(\mathbf{o}_t|S_t = j)$ is assumed to follow a multivariate Gaussian distribution with mean $\boldsymbol{\mu}_j$ and variance $\boldsymbol{\Sigma}_j$, then we can calculate the probability of a given observation as

$$P(\mathbf{o}_t|S_t = j) = \frac{1}{(2\pi)^{p/2} |\boldsymbol{\Sigma}_j|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{o}_t - \boldsymbol{\mu}_j)' \boldsymbol{\Sigma}_j^{-1} (\mathbf{o}_t - \boldsymbol{\mu}_j)\right),$$

where p is the dimension of the observation vector. We will write this as $P(\mathbf{o}_t|S_t = j) = \mathcal{N}(\mathbf{o}_j; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$ for short. However, it is generally recommended (also by Rabiner, p. 267) to use a *mixture of multivariate Gaussians* for the observation model. This means that we have a series of M different Gaussian distributions that *together* describe the observation model, and that the observation model is a weighted sum of these Gaussians, $P(\mathbf{o}_t|S_t = j) = \sum_{m=1}^M c_{jm} \cdot \mathcal{N}(\mathbf{o}_j; \boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm})$. In your project you can choose if you want to use a single multivariate Gaussian or a mixture of such distributions.

2.3 Learning in HMMs

The last part to have a functioning SRS is to decide upon the parameters of the distributions. We need

- The prior distribution $\boldsymbol{\pi}$.
- The dynamic model \mathbf{A} .
- The parameters in the observation model $\{\boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm}\}$ for $j = 1, \dots, N$, $m = 1, \dots, M$.

We can learn useful parameters from data using the algorithm described in Rabiner (pp. 264–267). Note that since we assume *continuous* features, we cannot use the equation (40c) for the observation model, but must rather use the technique described in equations 52–54.

Rabiner uses the FORWARD ALGORITHM already mentioned to calculate $\boldsymbol{\alpha}_t$ -values and a related BACKWARD step to calculate $\boldsymbol{\beta}_t$ -values, where $\beta_t(i) = P(\mathbf{o}_{t+1:T}|S_t = i)$, i.e., the probability of the observation sequence from $t + 1$

and until the end, given that the system is in state i at time t . As for α_t , Rabiner's algorithm for calculating β_t can be simplified using matrix manipulation; the inductive step can be written as

$$\beta_t = \mathbf{A}\mathbf{B}_{t+1}\beta_{t+1},$$

with the initialisation $\beta_T = [1 \dots 1]^T$.

As already mentioned, the calculation of α_t can be numerically unstable if T is large, and this gave rise to the *scaled* version of the FORWARD-algorithm. The same is true for β_t , so there is a scaled BACKWARD ALGORITHM, too. Note that we use the scaling factors from the FORWARD ALGORITHM, $\{\ell_t\}_{t=1}^T$, also in the scaled version of the BACKWARD ALGORITHM:

Initialisation:

- $\mathbf{r}'_T = [1 \dots 1]^T$.
- $\mathbf{r}_T \leftarrow \mathbf{r}'_T / \ell_T$.

Induction: Do for $t = T - 1, \dots, 1$:

- $\mathbf{r}'_t = \mathbf{A}\mathbf{B}_{t+1}\mathbf{r}_{t+1}$
- $\mathbf{r}_t \leftarrow \mathbf{r}'_t / \ell_t$.

After implementing the FORWARD and BACKWARD steps, we are ready to investigate the learning of the parameters in the Hidden Markov Model.

The main idea of the learning scheme is to realise that learning here is hard *because we do not observe all variables*. Had we seen the value of $\{S_1, \dots, S_T\}$ in addition to $\mathbf{o}_{1:T}$, then learning would simply be done by counting the observations and make simple matrix manipulations, and we would have obtained the parameters that maximise the likelihood of the observation in return. However, we do not observe $\mathbf{s}_{1:T}$, and need to compensate accordingly. The idea used by the EM algorithm (which the current learning algorithm is an instance of), is loosely speaking to “fill in” observations $\mathbf{s}_{1:T}$ based on the observations we have, and then estimate parameters as if these observations were real. The filling in is based on the *current* model, so after calculating new parameters (based on the filling in), we need to fill in again, now based on the updated model, and keep iterating like this until convergence.

The top level algorithm therefore runs like this when learning an HMM that should be tuned towards observations like $\mathbf{o}_{1:T}$:

1. “Guess” value for all parameters. This can, for instance, be done randomly, but make sure that constraints (like $\sum_j \pi_j = 1$ and that Σ_j is positive definite) are fulfilled.

2. Repeat

- (a) Do the FORWARD and BACKWARD passes using the current model and the observations $\mathbf{o}_{1:T}$ to get \mathbf{f}_t and \mathbf{r}_t for $t = 1, \dots, T$.
- (b) Calculate ξ_t and γ_t for all t using Rabiner's equations 37+38, but where the un-scaled parameters α_t and β_t are replaced by their scaled versions \mathbf{f}_t and \mathbf{r}_t . (These values represent the "filling in"-part of the algorithm.)
- (c) Re-estimate the prior distribution $\boldsymbol{\pi}$ and the transition model \mathbf{A} using Rabiner's equations 40a and 40b.
- (d) If you use a *mixture* for the observation model, you must re-estimate \mathbf{c} using equation 52.
- (e) Update values for $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ using equations 53 and 54.

Until the change in log likelihood calculated from the current model is only negligibly higher than in the previous iteration.

The SRS is fully specified after learning, and it should now be able to recognise speech in a controlled environment. It is expected that your system should correctly recognise at least 50% of the speech samples given to it, and if it does not obtain results of that quality, you may consider to tune the value of N (the number of states in the hidden variable), use a *mixture of Gaussians* for the observation model (and tune the number of mixture components), and the representation of speech.