EE516 Speech Processing and Recognition
Spring 2015

University of Washington
Dept. of Electrical Engineering

Homework 5: final project. Due Monday June 1st, 2015, 8:00am Electronically

Prof: J. Bilmes <bilmes@ee.washington.edu>
TA: Yuzong Liu <yzliu@uw.edu>

Monday May 18th, 2015

All homework is due electronically via the link https://canvas.uw.edu/courses/964524/assignments. Note that the due dates and times might be in the evening. Please submit a PDF file. Doing your homework by hand and then converting to a PDF file (by say taking high quality photos using a digital camera and then converting that to a PDF file) is fine, as there are many jpg to pdf converters on the web. Some of the problems below will require that you look at some of the lecture slides at our web page (http://j.ee.washington.edu/~bilmes/classes/ee516_spring_2015/).

There are 1 problems, for a maximum total of 350 points spanning 4 pages.

For the final project, You are to implement isolated word HMMs for doing an isolated word ASR system. This includes the expression of the Markov chain via a transition matrix, the initial state distribution, and the multivariate Gaussian distributions.

The first thing you should do is review lecture slides for lectures 7, 8, and 9 (this last one is a youtube lecture that will be ready by Friday night, May 22nd).

Problem 1. Final Project Problem (350 points)

The final project will consist of writing in Matlab from scratch a simple speaker dependent, isolated-word, whole-word model (i.e., one HMM/word), single Gaussian per state, diagonal covariance Gaussian, automatic speech recognition (ASR) system (see further definitions below), using the five-word vocabulary $\mathcal{W} = \{$“zero”, “one”, “two”, “three”, “four”$\}$.

You will need to use five separate HMMs, one for each word, that takes the following form: for each $w \in \mathcal{W}$, we have the probability distribution over a sequence of T MFCC vectors.

$$p(x_1, x_2, \ldots, x_T | w) = \sum_{y_1, y_2, \ldots, y_T} p(x_1, x_2, \ldots, x_T, y_1, \ldots, y_T | w) \tag{1}$$

$$= \sum_{y_1, y_2, \ldots, y_T} p(x_1|y_1) p(y_1) \prod_{t=2}^{T} p(x_t|y_t) p(y_t|y_{t-1}) \tag{2}$$

$$= \sum_{y_1=1}^{M_w} \sum_{y_2=1}^{M_w} \cdots \sum_{y_T=1}^{M_w} p(x_1|y_1) p(y_1) \prod_{t=2}^{T} p(x_t|y_t) p(y_t|y_{t-1}) \tag{3}$$

where y_i is a state variable at time t, x_t is an N-dimensional continuous feature vector (MFCCS and their deltas, so $N = 26$) at time t, $w \in \mathcal{W}$ is one of the five words, and T is the number of MFCC vectors (i.e., the number of frames in the utterance). Note that T is a constant for a given unknown utterance — i.e., you record some speech, convert it to MFCCS, and then T is the number of resultant frames. For different speech utterances, T will likely be different (i.e., “zero” is typically longer than “two”, and even two instances of the same word will not be exactly the same length). Lastly, M_w is the number of states in the HMM for word w.

Your HMMs will be over 26-dimensional MFCC feature vectors (the MFCCs and deltas) that you computed a part of homework 4. That is, $x_t \in \mathbb{R}^N$ where $N = 26$.

Here is a description of the other terms from above:

- **Matlab:** the programming language and system you'll use (it is generally not the fastest language, but for our purposes it will be fine, it also has functions such as `fft`, etc. that you can use).
- **speaker dependent:** The system is speaker dependent, meaning that it need not work well on anyone other than you. A speaker independent system is one that works regardless of who is speaking. Note that for the final demo, we will test your system on another person just to see if by chance you've achieved a bit of speaker independence (which would be nice but is not necessary).
- **isolated-word:** An isolated word ASR system is one where each word is presented in isolation from other words, which means that there is silence both before and after each word. In fact, in your system, you'll only be recording one word at a time, i.e., you'll start recording, the person will wait a bit, say a word, recording will stop, and the hope is that there will be silence in the recorded sound file both before/after the speech (you should think about how to make sure that this happens in your matlab recording system, if you record for a fixed amount of time, and the speech gets truncated at the end, it will be much harder to recognize).
- **whole-word model:** Our HMMs will not use phones, rather they will use states that are distinct for each word. In other words, the states do not have a phonetic interpretation, rather we will just use a separate HMM for each word, and there is no sharing of the states or the observation distributions across words. In larger speech systems, the states often have some sort of phonetic interpretation and observations are shared over different words (but we need not do this here). This will become more clear in the below.
- **single Gaussian per state:** Each state will have a single Gaussian density, i.e., $p(x_t|y_t) = \mathcal{N}(x_t|\mu_{y_t}, C_{y_t})$ where $\mathcal{N}(x_t|\cdot,\cdot)$ is an N-dimensional multivariate Gaussian, and μ_{y_t} is a mean vector, and C_{y_t} is a covariance matrix (which necessarily is positive definite), where y_t is a state index.
- **diagonal covariance Gaussian:** This is a simplification of the Gaussians, namely that C_{y_t} is a diagonal matrix. Since it is also positive definite, this means that C_{y_t} is a diagonal matrix with strictly positive entries along the diagonal and zeros everywhere else (this means, also, that you need **not** allocate storage for all N^2 entries, rather only the N entries along C_{y_t}'s diagonal).

  The distribution $p(x_t|y_t)$ is a Gaussian distribution with a diagonal covariance matrix, so this means that:

$$p(x_t|y_t) = \frac{1}{\sqrt{|2\pi C_{y_t}|}} \exp\left[-\frac{1}{2}(x_t - \mu_{y_t})^\intercal {C_{y_t}}^{-1}(x_t - \mu_{y_t})\right] \tag{4}$$

$$= \frac{1}{(2\pi)^{N/2}\sqrt{|C_{y_t}|}} \exp\left[-\frac{1}{2}(x_t - \mu_{y_t})^\intercal {C_{y_t}}^{-1}(x_t - \mu_{y_t})\right] \tag{5}$$

$$= \frac{1}{(2\pi)^{N/2}\sqrt{\prod_{i=1}^N C_{y_t}(i)}} \exp\left[-\frac{1}{2}\sum_{i=1}^N \frac{(x_t(i) - \mu_{y_t}(i))^2}{C_{y_t}(i)}\right] \tag{6}$$

  where $|C_{y_t}|$ is the determinant of matrix C_{y_t}, and $C_{y_t}(i)$ is the i^{th} diagonal entry in diagonal matrix C_{y_t}. As you can see, the diagonal covariance assumption leads to some nice computational savings for both computing the determinant and the matrix inverse.

  Since you will have five words, and M_w is the number of states in the HMM for word w, and since you will have one Gaussian per state, you will have total $\sum_{w \in \mathcal{W}} M_w$ number of Gaussians.

  You will need to test out different values for M_w for each w to see what works best, but I'd guess a reasonable starting point would be to set $M_w \approx 6$. Note, again, you need not use the same number of states for each word (longer words should probably have a few more states). Also remember, that the first state and the last state will probably model silence.

Your project will consist of the following stages:

1. Write code for the data structures and the algorithms of the HMM.

2. Write code that computes the $\alpha_t(y)$ recursion, as expressed in class. Note that sometimes people use the letter q as the state and other times people use the letter y for the state. In our case here, we're using y. This means that the state at time t, y_t is an integer from 1 to M_w. You'll probably want to store, for each HMM, the entire α matrix (which is going to be $M_w \times T$ for HMM w) for each word w.

3. Write code that computes the $\beta_t(y)$ recursion, as expressed in class. You'll probably want to store, for each HMM, the entire β matrix (which is going to be $M_w \times T$ for HMM w) for each word w. Note that you'll need a separate beta matrix for each instance of each word.

4. Note that you'll need a separate α and β matrix for each instance of each word (so if you have a five-word vocabulary, and say 10 instances of each word to train on using EM (see below) you'll need to compute 50 α and 50 β matrices, or 100 total matrices). You need not have them all in memory at the same time, during EM training (below), rather you can compute a pair of α and β matrices and keep them around, to compute the needed γ and ξ quantities, before deallocating them and moving on to the next utterance.

5. From either the α or the β matrix, you can compute the quantity $p(x_1, x_2, \ldots, x_T|w) = p(x_{1:T}|w)$.

6. Write code that computes the $\gamma_t(y)$ recursion, as expressed in class. This quantity you probably don't want to store in a matrix as it can be easily computed from the α and the β matrices.

7. Write code that computes the $\xi_t(i,j)$ quantity. This quantity also you probably don't want to store in a matrix as it can be easily computed from the α and the β matrices.

8. Collect a set of training utterances for each word. That is, you will have a training set $\mathcal{D}_w$ for each word where

$$\mathcal{D}_w = \left\{ x^1_{1:T^w_1}, x^2_{1:T^w_2}, \ldots \right\} \tag{7}$$

where T^w_i is the length, in number of frames, of the i^{th} instance of word w. The number of speech utterances $|D_w|$ for word w should probably be about 10, but you should experiment with this (you might need to use more to get your system to work better). The full set of training data is referred to as $\mathcal{D} = \cup_w D_w$.

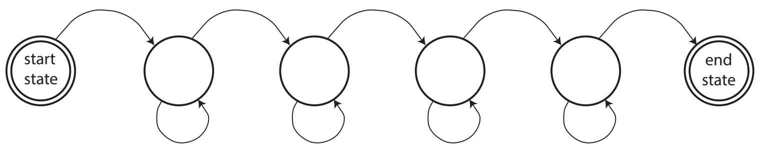
9. Machine learning: Implement the EM algorithm for HMMs, which is described in lecture 9 (this is to be placed on youtube). The EM algorithm allows you to compute the maximum likelihood estimate of the parameters. The HMM for word w can be written with its parameters as:

$$p(x_{1:T}|w, \lambda_w) \tag{8}$$

where λ_w are the parameters for HMM w. That is $\lambda_w = (\pi^w, A^w, \{b^w_i(x)\}_i)$ where

- π is the initial state distribution (a probability distribution over M_w values that in Equation (3) is written as $p(y_1)$),
- A is a transition matrix which is $M_w \times M_w$ (and that is written as $p(y_t|y_{t-1})$ in Equation (3), recall that $p(y_t|y_{t-1})$ is really shortcut notation, and what we really mean is $p(y_t|y_{t-1}) = p_{Y_t,Y_{t-1}}(Y_t = y_t|Y_{t-1} = y_{t-1}) = p_{Y_t,Y_{t-1}}(Y_t = j|Y_{t-1} = i) = a_{ij}$ when $y_t = j$ and $y_{t-1} = i$, and since a_{ij} is not t-dependent, this is a time-homogeneous Markov chain as discussed in class),

You probably want to use a strictly left-to-right sequential order transition matrix which means that $a_{ij} = 0$ whenever $j \notin \{i, i+1\}$ as shown in the following figure which corresponds to $M_w = 4$.



- and $b_i^w(x)$ is the diagonal covariance Gaussian mean vector and covariance matrix for word w's HMM state i (this corresponds to $p(x_t|y_t)$ in Equation (3)).

The EM algorithm computes:

$$\lambda_w^* \in \underset{\lambda_w}{\operatorname{argmax}} \sum_{j=1}^{|\mathcal{D}_w|} \log p(x^j_{1:T^w_j}|w, \lambda_w) \tag{9}$$

The parameters λ_w^* are the maximum likelihood parameter estimates. The exact equations for how to compute this, based on the α and β matrices, and the γ and ξ quantities are described in lectures 8 and 9.

10. Speech recognition: Speech recognition uses the trained HMMs you computed in the previous steps, and is done as follows. Given a new speech utterance that has been converted to a sequence of MFCC vectors $x_{1:T}$, speech recognition occurs as follows:

$$w^* \in \underset{w \in \mathcal{W}}{\operatorname{argmax}}\, p(x_{1:T}|w, \lambda_w^*) \tag{10}$$

11. Your matlab ASR system will run as follows. After prompting the user, you'll record some speech, convert it to MFCC vectors, run the above argmax, and then print out the word, and then prompt the user if they want to say another word (i.e., you don't need to have an always on mode, such as the Amazon Echo `http://www.amazon.com/oc/echo/`).