CS532c Fall 2004: Homework 6
Out Nov 1, Due Nov 10

In this problem, we will use Markov and Hidden Markov models to identify the language of written sentences. For simplicity our representation of text will include only 27 symbols— the 26 letters of the Latin alphabet, and the space symbol. Any accented letter is represented as a non-accented letter, none-Latin letters are converted to their closest Latin letters, and punctuation is removed. This representation naturally looses quite a bit of information compared to the original ASCII text. This 'handicap' is in part intentional so that the classification task would be a bit more challenging. Most of the MATLAB code you will need here will be given. You will find the following routines useful (here and perhaps in some of your projects as well):

readlines.m Reads a named text le, returning a cell array of the lines in the le. To get line i of cell-array lines returned from, e.g., `lines = readlines('cnn.eng')`, use `linesfig`$\{$`i`$\}$.

text2stream.m Converts a string (a line of text) into a row vector of numbers in the range $\{1, \dots, 27\}$, according to the representation discussed above. So, for example, `numberline = text2stream(lines1)` would convert the first line of text from lines into a row vector of numbers. The conversion of the full output of readlines would have to be done line by line.

count.m Given text in a row vector representation and a width k, the function computes the count of all k-grams in the array. In other words, the function returns a kdimensional array representing the number of times each configuration of k successive letters occurs in the text.

totalcount.m This function allows you to compute the accumulated counts from each of the lines of text returned by `readlines`. Use this function to find the training counts for the different languages.

The data is stored in HW6public/Data. You may find the `addpath` command helpful.

1 Language identification using Markov models

[10 points per question, total = 70.]

Here we will construct a language classifier by using Markov models as class-conditional distributions. In other words, we will separately train a Markov model to represent each of the chosen languages: English, Spanish, Italian and German. The training data is given in the files `cnn.eng, cnn.spa, cnn.ita, cnn.ger`, which contain several news articles (same articles in different languages), one article per line. We will first try a simple independent (zeroth-order Markov) model. Under this model, each successive symbol in text is chosen independently of other symbols. The language is in this case identified based only on its letter frequencies.

1. Write a function `naiveLL(stream,count1)` which takes a 1-count (frequency of letters returned by count.m) and evaluates the log-likelihood of the text stream (row vector of numbers) under the independent (zeroth-order Markov) model.

   Extract the total 1-counts from the language training sets described above. Before proceeding, let's quickly check your function `naiveLL`. If you evaluate the log-likelihood of 'This is an example sentence' using the English 1-counts from `cnn.eng`, you'll get -76.5690, while the Spanish log-likelihood of the same sentence is -77.2706.

2. Write a short function `naiveC` which takes a stream, and several 1-counts corresponding to difierent languages, and finds the maximum-likelihood language for the stream. You could assume, e.g., that the 1-counts are stored in an array, where each column corresponds to a specific language. The format of the labels should be in correspondence with the `test_labels` described below.

   The files `song.eng, song.spa, song.ita, song.ger` contain additional text in the four languages. We will use these as the test set:

```
test_sentences = [ readlines('song.eng') ; ...
                   readlines('song.ger') ; ...
                   readlines('song.spa') ; ...
                   readlines('song.ita') ] ;
test_labels = [ ones(17,1) ; ones(17,1)*2 ; ones(17,1)*3 ; ones(17,1)*4 ]
```

   In order to study the performance of the classifier as a function of the length of test strings, we will classify all prefixes of the lines in the test files. The provided routine `testC.m` calculates the success probability of the classification, for each prefix length, over all the streams or strings in a given cell-array. You can call this function, as follows

```
successprobs = testC(test_sentences,test_labels,'naiveC',count1s);
```

   where `count1s` provides the array of training counts that your function `naiveC` should accept as an input.

3. Plot the success probability as a function of the length of the string. What is the approximate number of symbols that we need to correctly assign new piece of text to one of the four languages?

In order to incorporate second order statistics, we will now move on to modeling the languages with first-order Markov models.

4. Write a function `markovLL(stream,count2)` which returns the log-likelihood of a stream under a first-order Markov model of the language with the specifed 2-count. For the initial state probabilities, you can use 1-counts calculated from the 2-counts.

   Quick check: The English log-likelihood of 'This is an example sentence' is -63.0643, while its Spanish log-likelihood is -65.4878. We are again assuming that you are using the training sets described above to extract the 2-counts for the di erent languages.

   Write a corresponding function `markovC.m` that classifies a stream based on Markov models for various languages, specifed by their 2-counts.

5. Try to classify the sentence 'Why is this an abnormal English sentence'. What is its likelihood under a Markov model for each of the languages ? Which language does it get classified as ? Why does it not get classified as English?

As we discussed in class, it is common to use a Dirichlet prior to regularize the counts. The resulting MAP estimate is

$$\hat{\theta}_i = \frac{\hat{n}_i + \alpha_i}{\sum_{j=1}^{m}(\hat{n}_j + \alpha_j)} \tag{1}$$

which will be non-zero whenever $\alpha_i > 0$ for all $i = 1, \ldots, m$. Setting $\alpha_i = 1/m$ would correspond to having a single prior observation distributed uniformly among the possible elements $i \in \{1, \ldots, m\}$. Setting $\alpha_i = 1$, on the other hand, would mean that we had m prior observations, observing each element i exactly once.

6. Add pseudocounts (one for each con guration) and reclassify the test sentence. What are the likelihoods now. Which language does the sentence get classified as ?

7. Use `testC.m` to test the performance of Markov-based classification (with the corrected counts) on the test set. Plot the correct classification probability as a function of the text length. Compare the classification performance to that of `naiveC.m`. (Turn in both plots).

2 Hidden Markov Models

(30 points) We will now turn to a slightly more interesting problem of language segmentation: given a mixed-language text, we would like to identify the segments written in diﬁerent languages. For simplicity, we will consider a single sentence composed of just 2 languages, Spanish and German (as in homework 5).

1. (5 points) A simple approach would be to classify each character individually, based on its likelihood under each class-conditional density (using naiveC). Why would we expect the resulting segmentation not to agree with the true segmentation? What would the resulting segmentation look like? What is the critical piece of information we are not using in this approach ?

2. (10 points) A better approach is to use an HMM, where the hidden state represents which of the languages we are currently in. The goal of this part is to train an HMM using EM applied to the gerspa sentence used in homework 5 (contained in `segment.mat`). You can use the provided function `dhmm_em` (d stands for discrete) for this, as follows:

```
transmat0 = xxx % your initial guess of the transition matrix
obsmat0 = xxx % your initial guess of the observation matrix
prior0 = [0;1]; % always start in state 2
maxIter = 30; % max num. iterations you want to wait
[LLtrace, prior, transmat, obsmat] = dhmm_em(data.gerspa, ...
    prior0, transmat0, obsmat0, ...
    'adj_prior', 0, 'max_iter', maxIter);
```

   Here, transmat(i,j) = $P(X_t = j|X_{t-1} = i)$, and obsmat(i,o) = $P(Y_t = o|X_t = i)$. We clamp the prior state distribution to $P(X_1 = 2) = 1.0$, since we cannot learn this from only one training sequence (so the optional argument 'adjust prior' is set to 0). Since the sentence starts out with German, and we define the initial state to be 2, we are effectively defining state 2 to be German (thus breaking the symmetry in the hidden label space). The output of the function are the new parameters, and the trace of log-likelihood vs iteration.

   Your goal is to choose a good set of initial parameters, transmat0 and obsmat0, so that EM converges to a good local optimum. You can determine the quality of your solution by plugging the parameters into the Viterbi algorithm, and using it to decode the sentence from homework 5. You should be able to get as low as 64 classification errors. Hint: use your 1-counts from question 1 to initialise obsmat, and use a self-transition probability that reflects sentence length to initialise transmat. (Do not cheat by using the parameters from homework 5 as your initial guess! You must show your initial parameter guesses, and justify why they are sensible.)

3. (10 points). Use the provided function `fwdback.m` to calculate the per character posterior probabilities over the states. These are the $\gamma_t(i) = P(X_t = i|y_{1:T})$ probabilities described in lectures (t gives the character position in text and i specifies the state). Plot these probabilities as a function of the character position in the text sequence and turn in the plot. You might want to re-scale the axis using `axis[0 3000 0 1.1]`.

4. (5 points). Find the sequence of most probable states, i.e., for each time point t, find $\arg\max_i P(X_t = i|y_{1:T})$. Compare this sequence with the most probable sequence of states, as computed using `viterbiPath`. Are they different? If so, in which positions?