CS340 Fall 2006: Homework 5

Out Wed 11 Oct, back Wed 18 Oct

1 Posterior predictive distribution for the Dirichlet-multinomial model

Let $X \sim Multinomial(\theta_i)$ where $\theta_i \overset{\text{def}}{=} P(X = i)$ for $i = 1 : K$. We will use a conjugate (Dirichlet) prior with $\theta_{1:K} \sim Dir(\alpha_{1:K})$. Suppose we have seen a data set $D_{old} = (x_1, \ldots, x_{N_{old}})$ of iid data with sufficient statistics (counts) N_k^{old}, and we want to predict the probability of seeing a future data set $D_{new} = (x_1, \ldots, x_{N_{new}})$ of iid data with sufficient statistics (counts) N_k^{new}. Compute

$$p(D_{new}|D_{old}, \vec{\alpha}) = \int p(D_{new}|\vec{\theta}) p(\vec{\theta}|D_{old}, \vec{\alpha}) d\vec{\theta} \tag{1}$$

Your answer should be a function of $\vec{\alpha}$,N_k^{old} and N_k^{new}. Hint 1: recall that the marginal likelihood is given by

$$P(D|\vec{\alpha}) = \frac{\Gamma(\sum_k \alpha_k)}{\Gamma(N + \sum_k \alpha_k)} \prod_{k=1}^{K} \frac{\Gamma(N_k + \alpha_k)}{\Gamma(\alpha_k)} \tag{2}$$

where $\{N_k\}$ are the sufficient statistics (counts) of D.
Hint 2: Bayes rule says $p(D_{new}|D_{old}) = p(D_{new}, D_{old})/p(D_{old})$.
Hint 3: remember that Bayesian updating can be done recursively, so an old posterior can become a new prior.

2 Fun with entropy

Consider the joint distribution $p(X, Y)$

		x			
		1	2	3	4
	1	1/8	1/16	1/32	1/32
y	2	1/16	1/8	1/32	1/32
	3	1/16	1/16	1/16	1/16
	4	1/4	0	0	0

1. What is the joint entropy $H(X, Y)$?

2. What are the marginal entropies $H(X)$ and $H(Y)$?

3. The entropy of X conditioned on a specific value of y is defined as

$$H(X|Y = y) = -\sum_x p(x|y) \log p(x|y) \tag{3}$$

   Compute $H(X|y)$ for each value of y. Does the posterior entropy on X ever increase given an observation of Y?

4. The conditional entropy is defined as

$$H(X|Y) = \sum_y p(y) H(X|Y = y) \tag{4}$$

Compute this. Does the posterior entropy on X increase or decrease when averaged over the possible values of Y?

5. What is the mutual information between X and Y?

3 Mutual information

Prove that

$$I(X, Y) = H(X) - H(X|Y) = H(Y) - H(Y|X) \tag{5}$$

4 Conditioning reduces entropy

Prove

$$H(X|Y) \leq H(X) \tag{6}$$

with equality iff X and Y are independent. Hint: use the fact that $I(X, Y) \geq 0$.

5 Language identification using Markov models

In this problem, we will will construct a language classifier by using n'th order Markov models as class-conditional distributions. In other words, we will train a separate Markov model to represent each of the chosen languages (English, German, Spanish, and Italian), and then compute the likelihood of a novel sentence under each of these models. The training data is given in the files `cnn.eng, cnn.ger, cnn.spa, cnn.ita`, which contain several news articles (same articles in different languages), one article per line.
We will compute the statistics of individual letters or sequences of letters for each language; these are called n-gram statistics. If $n = 1$, we are using unigrams (marginal letter frequencies), so $c1(i)$ is the number of times letter i appears; if $n = 2$ (a first order Markov model), we are counting bigram frequencies, so $c2(i, j)$ is the number of times letter i is followed by letter j; and so on.
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, non-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 to make the classification task a bit more challenging.
The following matlab functions are provided.

stream = text2stream(string) Converts a string (a line of text) into a row vector of numbers in the range $\{1, \ldots, 27\}$.

streams = readlines(filename) Reads a named text file, returning a cell array of the lines in the file, with letters converted to numbers as above.

t = totalcount(streams, n) Computes n-gram statistics from a cell array of streams, as returned by readlines. eg. c1=totalcount(streams,1) gives the unigram statistics c1(i) and c2=totalcount(streams,2) gives the bigram statistics c2(i,j).

counts1, counts2 = getCounts()] Computes the unigram and bigram counts. $counts1(i, c)$ is the number of times word i occurs in language c. $counts2(i, j, c)$ is the number of times word i is followed by word j in language c.

Now answer the following questions. (Note: the numbers you get may differ slightly from those below, because different operating systems seem to treat the foreign characters differently.)

1. Write a function `ll = naiveLL(stream, count1)` which takes a text stream (represented as a vector of numbers) and the unigram counts and evaluates the log-likelihood of the text stream using the maximum likelihood estimates $\hat{\theta}_i$ derived from the counts:

   $$p(x_{1:T}|\hat{\theta}) = \prod_{t=1}^{T}\prod_{i=1}^{K}\hat{\theta}_i^{I(x_t=i)} \quad (7)$$

   $$\log p(x_{1:T}|\hat{\theta}) = \sum_{t=1}^{T}\sum_{i=1}^{K} I(x_t = i)\log\hat{\theta}_i \quad (8)$$

   $$= \sum_{i=1}^{K} M_i(x_{1:T})\log\hat{\theta}_i \quad (9)$$

   where $x_{1:T}$ is the test stream, $M_i(x_{1:T})$ is the number of times symbol i occurs in the test stream, K is the number of symbols in the alphabet, and the MLE derived from counts1 is

   $$\hat{\theta}_i = N_i/N \quad (10)$$

   where N_i is the number of times symbol i occurs in the training streams (i.e., $N_i = count1(i)$). Turn in your code.

   As a check, 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 -77.1211. You can test as follows:

   ```
   eng = 1; ger = 2; spa = 3; ita = 4;
   ll = naiveLL(text2stream('this is an example sentence'), counts1(:,eng))
   ll = naiveLL(text2stream('this is an example sentence'), counts1(:,spa))
   ```

2. Write a function `c=naiveC(stream, counts1)` where stream is a numeric vector and counts1(:,l) are the counts for language l. Return the most probable class (language) c, where 1=english, 2=german, 3=spanish, 4=italian. Turn in your code for naiveC. It should use naiveLL as a subroutine.

3. Now we want to apply the trained model to classify some novel text. The files `song.eng, song.ger, song.spa, song.ita` 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 ]
   ```

   We will study the performance of the classifier as a function of the length of test strings by classifying 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',counts1);
   ```

   This calls your function `naiveC` with each line of test_sentences and with the `counts1` argument, and compares it to test_labels. Use this function to plot the success probability as a function of the length of the string. It should look like Figure 1(left). Turn in your code and plot.

4. We will now move on to modeling the languages with first-order Markov models.
   Write a function `markovLL(stream,count2,count1)` which returns the log-likelihood of a stream under a first-order Markov model of the language with the specifed bigram and unigram statistics i.e., convert the bigram statistics count2(i,j) into conditional probabilities $P(X_t = j|X_{t-1} = i)$ and the unigram statstics[1] count1(i) into marginal probabilities $P(X_1 = i)$, and use these MLEs as a plug-in to compute the log-likelihood. The result should be an equation similar (but not identical!) to Equation 9. Hint: you make find the provided function $A = mk\_stochastic(M)$ useful; this converts a matrix of counts $M(i,j)$ into a stochastic matrix $A(i,j)$, where $A(i,j) = M(i,j)/(\sum_{j'} M(i,j'))$.

   Obviously if $N_{ij} = 0$ in the training, then when estimating $\log \hat{\theta}_{ij}$ you will get a "log of zero" warning. You can use **warning off** to turn off such messages; remember to put **warning on** after the offending line. If an $i \rightarrow j$ transition occurs in the test sentence, then the log likelihood will be $\log 0 = -\infty$. However, for the stream 'This is an example sentence' you should find the English log-likelihood is -63.0745, while its Spanish log-likelihood is -65.6447:

```
ll = markovLL(text2stream('this is an example sentence'), counts2(:,:,eng), counts1(:,eng));
ll = markovLL(text2stream('this is an example sentence'), counts2(:,:,spa), counts1(:,spa));
```

   (Your numbers might differ in the second or third decimal place, but should be basically the same.) Note that these numbers are higher than the unigram model, indicating a better fit to the training data. Turn in your code for MarkovLL.

5. Write a function `c=markovC(stream, counts2, counts1)` where stream is a numeric vector and counts2(:,:,l) are the counts for language l. Return the most probable class (language) c, where 1=english, 2=german, 3=spanish, 4=italian. Turn in your code for MarkovC.

6. Try to classify the sentence 'Why is this an abnormal English sentence'. What is its log-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? (Your code may try to take log of 0; explain why it does this. We will fix this problem below.)

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

$$p(X_t = j|X_t = i) = \hat{\theta}_{j|i} = \frac{n_{j|i} + \alpha_{j|i}}{\sum_{j'=1}^{m}(n_{j'|i} + \alpha_{j'|i})} \tag{11}$$

   where $\theta_{j|i}$ is the probability of word j after seeing i, and $n_{j|i}$ are the counts of j following i. Add pseudocounts of $\alpha_{j|i} = 1$ to the empirical bigram counts and reclassify the test sentence. (There is no need to add pseudocounts to the unigram counts.) Which language does the sentence get classified as now?

8. Use

```
probs = testC(test_sentences, test_labels, 'markovC', counts2, counts1);
```

   to test the performance of Markov-based classification (with the corrected counts) on the test set (this may take a few seconds). testC calls your function `markovC` with each line of test_sentences and with the `counts2` and `counts1` arguments, and compares it to test_labels. (`testC.m` uses `varargin` to handle a variable number of arguments; this is standard matlab practice when passing a function to a function.) This returns the success probability for each stream length. Plot the correct classification probability as a function of the text length. It should look like Figure 1(right).

[1] You can derive the unigram counts from the bigram counts by marginalizing, but there is an ambiguity which arises depending on whether you sum counts2 over the first or 2nd dimension. Example: 2-counts: "ab", "ba", each once. Was the sequence "aba" or "bab"? For the first, counts1(a)=2, for the second, counts1(a)=1. Hence we require you pass in count1 as a separate argument.

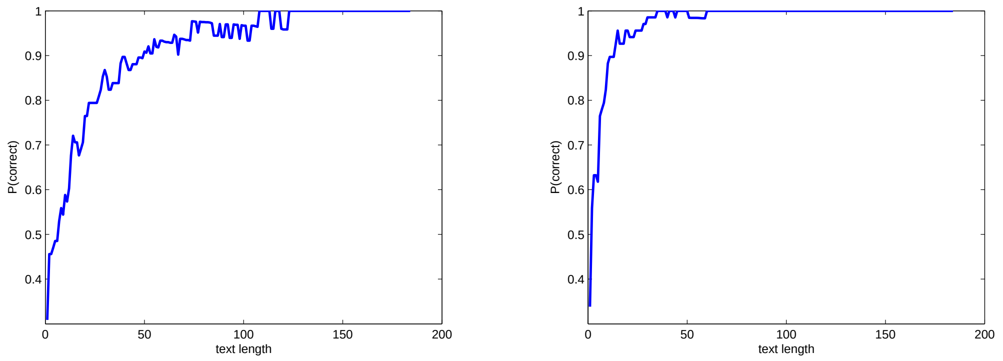

Figure 1: Performance vs number of test symbols for (left) Naive Bayes model and (right) first order Markov model. We see that the Markov model correctly classifies the document more quickly than the naive model (i.e., it needs to see less data). For example, to reach 95% correct, the naive model needs about 55 characters, whereas the Markov model nees only 18 characters.