CS 340 Fall 2007: Homework 6

1 Bivariate Gaussians

Let $X \sim \mathcal{N}(\mu, \Sigma)$ where $X \in \mathbb{R}^2$ and

$$\Sigma = \begin{pmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{pmatrix} \tag{1}$$

where ρ is the correlation coefficient. Show that the pdf is given by

$$p(x_1, x_2) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \exp\left(-\frac{1}{2(1-\rho^2)}\left(\frac{(x_1-\mu_1)^2}{\sigma_1^2} + \frac{(x_2-\mu_2)^2}{\sigma_2^2} - 2\rho\frac{(x_1-\mu_1)}{\sigma_1}\frac{(x_2-\mu_2)}{\sigma_2}\right)\right) \tag{2}$$

Hint: The determinant for a 2×2 matrix

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \tag{3}$$

is given by

$$|A| = \text{det}A = ad - bc \tag{4}$$

and its inverse is given by

$$A^{-1} = \frac{1}{|A|}\begin{pmatrix} d & -b \\ -c & a \end{pmatrix} \tag{5}$$

2 Mutual information for correlated normals

(Source: Exercise 9.3 of [**?**])
Find the mutual information $I(X, Y)$ where

$$\begin{pmatrix} X \\ Y \end{pmatrix} \sim \mathcal{N}_2\left(\mathbf{0}, \begin{pmatrix} \sigma^2 & \rho\sigma^2 \\ \rho\sigma^2\sigma^2 & \sigma^2 \end{pmatrix}\right) \tag{6}$$

Evaluate $I(X, Y)$ for $\rho = 1$, $\rho = 0$ and $\rho = -1$ and comment. Hint: The (differential) entropy of a d-dimensional Gaussian is

$$h(\mathbf{X}) = \tfrac{1}{2}\log\left[(2\pi e)^d \det\Sigma\right] \tag{7}$$

3 A measure of correlation (normalized mutual information)

(Source: Exercise 2.20 of [**?**])
Let X and Y be identically distributed (so $H(X) = H(Y)$) but not necessarily independent. Define

$$r = 1 - \frac{H(Y|X)}{H(X)} \tag{8}$$

1. Show $r = \frac{I(X,Y)}{H(X)}$

2. Show $0 \leq r \leq 1$
3. When is $r = 0$?
4. When is $r = 1$?

4 Gaussian decision Boundaries

Suppose we have two 1D normal distributions with the same variance, but with different means: $N(\mu_1, \sigma^2)$ and $N(\mu_2, \sigma^2)$. Explain the effect on the decision boundary of changing the class prior $p(Y = 1)$.

5 More Gaussian decision boundaries

Let $p(x|y = j) = \mathcal{N}(x|\mu_j, \sigma_j)$ where $j = 1, 2$ and $\mu_1 = 0, \sigma_1^2 = 1, \mu_2 = 1, \sigma_2^2 = 10^6$. Let the class priors be equal, $p(y = 1) = p(y = 2) = 0.5$.

1. Find the decision region
$$R_1 = \{x : p(x|\mu_1, \sigma_1) \geq p(x|\mu_2, \sigma_2)\} \tag{9}$$
Sketch the result. Hint: draw the curves and find where they intersect. Find *both* solutions of the equation
$$p(x|\mu_1, \sigma_1) = p(x|\mu_2, \sigma_2) \tag{10}$$
Hint: recall that to solve a quadratic equation $ax^2 + bx + c = 0$, we use
$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{11}$$
2. Now suppose $\sigma_2 = 1$ (and all other parameters remain the same). What is R_1 in this case?

6 Gibbs sampling from an Ising model (Matlab)

The goal of this exercise is to draw samples from an Ising model, at different temperatures. Specifically, the goal is to reproduce Figure 1. To make this, you can modify the function `gibbsDemoDenoising`. Simplify this function by removing all the parts that depend on local evidence (we want to sample from the prior and ignore the likelihood). You can choose an initial sample randomly, rather than by thresholding pixels.

7 Language identification using Markov models (Matlab)

(Source: Jaakkola, modified by Murphy)
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. (The data files are in the `markovLanguageData` subdirectory.)
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.

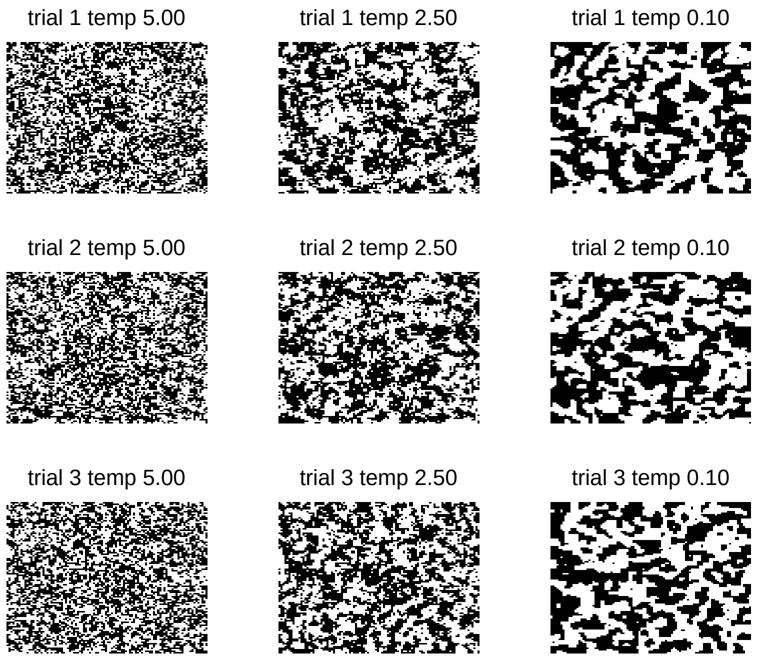

Figure 1: Samples from an 100×100 Ising model at decreasing temperatures. We use a coupling strength of $W = 1/T$, where T is the temperature. We use $T = 5$, $T = 2.5$ and $T = 0.1$ from left to right. We show 3 samples at each temperature, drawn after 50,000 single site updates using Gibbs sampling, applied to a random initial state. Produced by `gibbsDemoIsing`.

For simplicity, our representation of text will include only 27 symbols: the 26 letters of the Latin alphabet, and the space symbol. All characters outside of this set are replaced by a space symbol. 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.
You will need the following functions.

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

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.

(counts1, counts2) = getCountsMarkovLanguageEx() Computes the unigram and bigram counts from all the training files. $counts1(i, c)$ 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.

1. Write a function `ll = naiveLL(stream, counts1)` which takes a text stream (represented as a vector of numbers) and the unigram counts and evaluates the log-likelihood of the text stream by plugging in the maximum likelihood estimates $\hat{\pi}_i$ derived from the counts:

$$p(x_{1:T}|\hat{\theta}) = \prod_{t=1}^{T}\prod_{i=1}^{K} \hat{\pi}_i^{I(x_t=i)} \tag{12}$$

$$\log p(x_{1:T}|\hat{\boldsymbol{\pi}}) = \sum_{t=1}^{T}\sum_{i=1}^{K} I(x_t = i)\log \hat{\pi}_i \tag{13}$$

$$= \sum_{i=1}^{K} M_i(x_{1:T})\log \hat{\pi}_i \tag{14}$$

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

$$\hat{\pi}_i = N_i/N \tag{15}$$

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 -76.9307, while the log-likelihood of the same sentence under the Spanish parameters is -77.5193. You can test as follows:

```
eng = 1; ger = 2; spa = 3; ita = 4;
str = 'this is an example sentence';
stream = text2stream(str);
ll = naiveLL(stream, counts1(:,eng)) % -76.9307
ll = naiveLL(stream, counts1(:,spa)) % -77.5193
```

2. Write a function `yhat=naiveLanguageClassifier(stream, counts1)` where stream is a numeric vector and counts1 is as above. Return the most probable class (language) yhat, where 1=english, 2=german, 3=spanish, 4=italian. Turn in your code. It should use `naiveLL` as a subroutine. You can assume the prior over classes is uniform.

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 `testLanguageClassifier.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:

```
probs = testLanguageClassifier(test_sentences, test_labels, ...
              'naiveLanguageClassifier',counts1);
```

This calls your function `naiveLanguageClassifier` 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 2(left). Turn in your code and plot.

4. We will now move on to modeling the languages with first-order Markov models.
   Write down an equation similar to Equation 14 for the log-likelihood of a test sentence given known parameters $\boldsymbol{\pi}$ (initial state distribution) and A (transition matrix).

5. Write a function `markovLL(stream,counts2,counts1)` that estimates $\boldsymbol{\pi}$ and A from the specifed bi-gram and unigram statistics,[1] and returns the log-likelihood of the sentence. Use the MLE for $\pi_i = P(X_1 = i)$ (derived from counts1 as above), but use the posterior mean estimate for $A(i,j) = P(X_t = j|X_{t-1} = i)$, derived from counts2 and a Dirichlet prior with pseudocounts of $\boldsymbol{\alpha} = \mathbf{1}$ to avoid problems with 0 counts. Hint: you may 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 $A(i,j) = M(i,j)/(\sum_{j'} M(i,j'))$. Hint 2: as a sanity check, you should get the following

   ```
   str = 'this is an example sentence';
   stream = text2stream(str);
   markovLL(stream, counts2(:,:,eng), counts1(:,eng)) % -64.2469
   markovLL(stream, counts2(:,:,spa), counts1(:,spa)) % -66.7015
   ```

   Note that these numbers are higher than the unigram model, indicating a better fit to the training data. Turn in your code.

6. Write a function `c=markovLanguageClassifier(stream, counts2, counts1)`. Return the most probable class (language), where 1=english, 2=german, 3=spanish, 4=italian. (Use a uniform prior over classes.) Turn in your code.

7. Use

   ```
   probs = testLanguageClassifier(test_sentences, test_labels, ...
         'markovLanguageClassifier', counts2, counts1);
   ```

   to test the performance of Markov-based classification on the test set. testLanguageClassifier calls your function `markovLanguageClassifier` with each line of test_sentences and with the `counts2` and `counts1` arguments, and compares it to test_labels. 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 2(right). We see that this model needs much less data at test time to come up with the right classification. Turn in your code and plot.

[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.

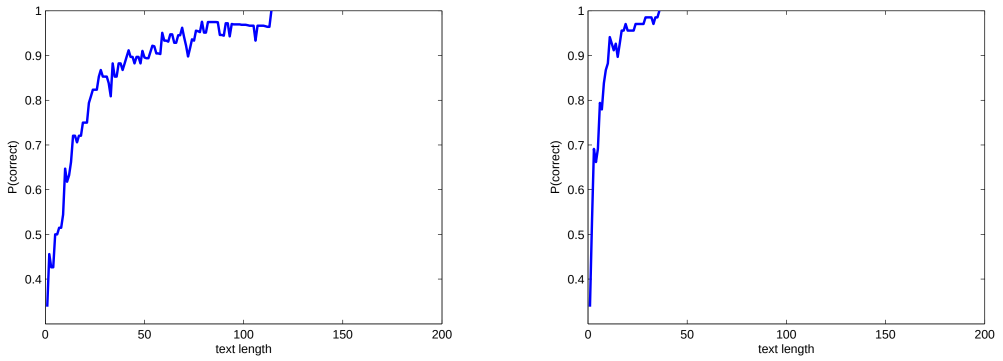

Figure 2: Performance vs number of test symbols for (left) Naive Bayes model and (right) first order Markov model. We see that the Markov model correctly classifies the document more quickly than the naive model (i.e., it needs to see less data). For example, to reach 95% correct, the naive model needs about 59 characters, whereas the Markov model nees only 17 characters.