HW4

1 Normalization constant for a 1D Gaussian

The normalization constant for a zero-mean Gaussian is given by

$$Z = \int_a^b \exp\left(-\frac{x^2}{2\sigma^2}\right) dx \tag{1}$$

where $a = -\infty$ and $b = \infty$. To compute this, consider its square

$$Z^2 = \int_a^b \int_a^b \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) dxdy \tag{2}$$

Let us change variables from cartesian (x, y) to polar (r, θ) using $x = r\cos\theta$ and $y = r\sin\theta$. Since $dxdy = rdrd\theta$, and $cos^2\theta + \sin^2\theta = 1$, we have

$$Z^2 = \int_0^{2\pi} \int_0^{\infty} r \exp\left(-\frac{r^2}{2\sigma^2}\right) drd\theta \tag{3}$$

Evaluate this integral and hence show $Z = \sigma\sqrt{(}2\pi)$. Hint 1: separate the integral into a product of two terms, the first of which (involving $d\theta$) is constant, so is easy. Hint 2: if $u = e^{-r^2/2\sigma^2}$ then $du/dr = -\frac{1}{\sigma^2} re^{-r^2/2\sigma^2}$, so the second integral is also easy (since $\int u'(r)dr = u(r)$).

2 Normalization constant for a multidimensional Gaussian

Prove that the normalization constant for a d-dimensional Gaussian is given by

$$(2\pi)^{d/2}|\mathbf{\Sigma}|^{\frac{1}{2}} = \int \exp(-\tfrac{1}{2}(\mathbf{x}-\mu)^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}))d\mathbf{x} \tag{4}$$

Hint: diagonalize $\mathbf{\Sigma}$ and use the fact that $|\mathbf{\Sigma}| = \prod_i \lambda_i$ to write the joint pdf as a product of d one-dimensional Gaussians in a transformed coordinate system. (You will need the change of variables formula: see p511.) Finally, use the normalization constant for univariate Gaussians.

3 Bivariate Gaussians

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

$$\Sigma = \begin{pmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{pmatrix} \tag{5}$$

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

$$p(x_1, x_2) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \exp\left(-\frac{1}{2(1-\rho^2)}\left(\frac{(x_1-\mu_1)^2}{\sigma_1^2} + \frac{(x_2-\mu_2)^2}{\sigma_2^2} - 2\rho\frac{(x_1-\mu_1)}{\sigma_1}\frac{(x_2-\mu_2)}{\sigma_2}\right)\right) \tag{6}$$

4 Class conditional densities for binary data

Consider a generative classifier for C classes with class conditional density $p(\mathbf{x}|y)$ and uniform class prior $p(y)$. Suppose all the d features are binary, $x_j \in \{0, 1\}$. If we assume all the features are conditionally independent (the naive Bayes assumption), we can write

$$p(\mathbf{x}|y = c) = \prod_{j=1}^{d} \text{Ber}(x_j|\theta_{jc}) \tag{7}$$

This requires dC parameters.

1. Now consider a different model, which we will call the "full" model, in which all the features are fully dependent (i.e., we make no factorization assumptions). How might we represent $p(\mathbf{x}|y = c)$ in this case? How many parameters are needed to represent $p(\mathbf{x}|y = c)$?

2. Assume the number of features d is fixed. Let there be n training cases. If the sample size n is very small, which model (naive Bayes or full) is likely to give lower test set error, and why?

3. If the sample size n is very large, which model (naive Bayes or full) is likely to give lower test set error, and why?

4. What is the computational complexity of fitting the full and naive Bayes models as a function of n and d? Use big-Oh notation. (Fitting the model here means computing the MLE or MAP parameter estimates. You may assume you can convert a d-bit vector to an array index in $O(d)$ time.)

5. What is the computational complexity of applying the full and naive Bayes models at test time to a single test case?

6. Suppose the test case has missing data. Let $\mathbf{x}_v$ be the visible features of size v, and $\mathbf{x}_h$ be the hidden (missing) features of size h, where $v + h = d$. What is the computational complexity of computing $p(y|\mathbf{x}_v, \hat{\boldsymbol{\theta}})$ for the full and naive Bayes models, as a function of v and v?

5 Gaussian decision boundaries

(Source: [**?** , Q3.7])
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

$$R_1 = \{x : p(x|\mu_1, \sigma_1) \geq p(x|\mu_2, \sigma_2)\} \tag{8}$$

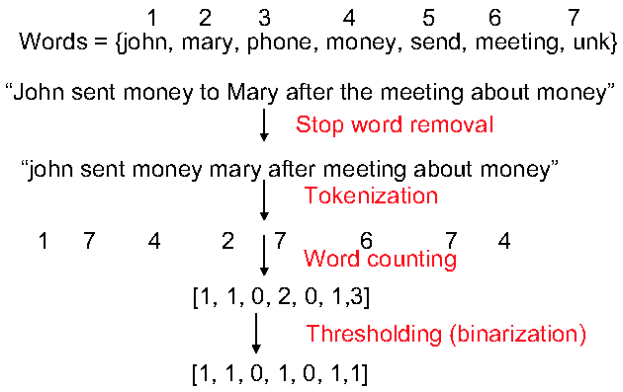
   Sketch the result. Hint: draw the curves and find where they intersect. Find *both* solutions of the equation

$$p(x|\mu_1, \sigma_1) = p(x|\mu_2, \sigma_2) \tag{9}$$

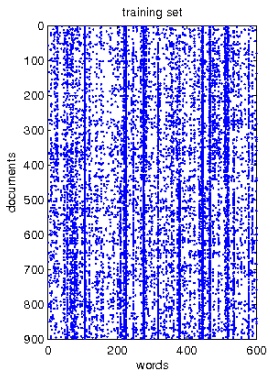
   Hint: recall that to solve a quadratic equation $ax^2 + bx + c = 0$, we use

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{10}$$

2. Now suppose $\sigma_2 = 1$ (and all other parameters remain the same). What is R_1 in this case?



(a)



(b)

Figure 1: (a) Examples of how to represent a sentence as a bag of words. (b) A small document $\times$ word co-occurrence matrix for 900 dicuments and 600 words. A black dot in row i column j means document i contains word j at least once. The first 450 lines correspond to the documents about X windows, the second 450 lines to documents about MS windows. Can you see a difference in the patterns between the first 450 rows and the second 450 rows? Figure produced by `docDataPlot`.

6 Naive Bayes classifier for email spam filtering (Matlab)

(Source: Jaakkola)
To solve this problem, use the file `naiveBayesExCode.zip`.
Consider the problem of classifying messages posted to online discussion boards. We consider two classes: messages about X Windows (class 1) and messages about microsoft Windows (class 2). (This is analogous to **email spam filtering**, which we will consider below; we start with this smaller problem to help you debug your code.) There are 900 documents from each class; we divided them into training and text sets of equal size. To save space (and time), we ran word detection on the documents, and the data available to you consists of binary feature vectors for each document. Upon loading `docdata.mat` the Matlab environment will contain

```
Name          Size              Bytes  Class     Attributes
  vocab       600x1             43914  cell
  xtest       900x600          147412  double    sparse
  xtrain      900x600          152884  double    sparse
  ytest       900x1              7200  double
  ytrain      900x1              7200  double
```

`xtrain(i,w)`=1 iff document i contains word w, otherise `xtrain(i,w)`=0. Note that `xtrain` and `xtest` are **sparse** matrices, since most entries are zero. You can visualize sparse matrices using the `spy` command: see Figure 1(b). The identity of the 600 words is stored in the cell array `vocab`. You can print out the first 10 words using

```
cellfun(@(x) sprintf('%s',x), vocab(1:10),'UniformOutput',false)
```

which produces

```
Columns 1 through 7
    'straight'    'magazines'    'issues'    'ray'    'enabled'    'head'    'improved'
  Columns 8 through 10
    'thread'    'libs'    'working'
```

1. Implement the following function

   ```
   function theta = NBtrain(X,Y)
   ```

```
% Posterior mean estimate of Naive Bayes parameters
% Input:
% X(i,j) = 1 if word j appears in document i, otherwise X(i,j)=0
% Y(i) = class label of doc i (assumed to be 1 or 2)
% Output:
% theta(j,c) = probability of word j appearing in class c
```

which computes the following posterior mean estimate

$$\hat{\theta}_{jc} = \frac{N_{jc} + 1}{N_c + 2} \tag{11}$$

where N_{jc} counts the number of times word j appears in class c, N_c is the total number of documents in class c, and we have assumed a Beta(1,1) prior. Turn in your code.

2. Implement a function to classify each document, assuming uniform class priors $p(Y = 1) = p(Y = 2) = 0.5$.

```
function y = NBapply(X,theta)
% X(i,j) = 1 if word j appears in document i, otherwise X(i,j)=0
% theta(j,c) = prob of word j in class c
% y(i) = most probable class for X(i,:)
```

Here $y(i) = \arg\max_c p(y = c|X(i,:))$ is the most probable class label for document i. Since $p(Y = c|\mathbf{x}) \propto p(\mathbf{x}|Y = c)$ is a small number, you will need to use logs to avoid underflow. (You don't necessarily need the logsumexp trick, because it suffices to compute the log likelihood $p(\mathbf{x}|y)$ rather than the normalized posterior $p(y|\mathbf{x})$, but you will need to use logs somehow!) Turn in your code.

3. Use `NBtrain` on the data in `xtrain,ytrain`. Compute the misclassification rates (i.e., the number of documents that you mis-classified) on the training set (by using `NBapply` on `xtrain,ytrain`) and on the test set (by using `NBapply` on `xtest,ytest`). Sanity check: You should get test error of **0.1867**.

4. The provided function `NBcv` computes the K-fold cross-validation error. (This calls your functions `NBtrain` and `NBapply`.) $K = 1$ means no cross-validation, that is error is simply computed on the whole training set. Use this to compute the 10-fold error rate on the training set. How does this compare to the (non cross validated) training and test error?

5. Plot (as histograms) the class-conditional densities $p(x_j = 1|y = c, \boldsymbol{\theta}_c)$ for classes $c = 1, 2$ and words $j = 1 : 600$. You should get the same result as Figure **??**.

6. What are the 5 most likely words in each class?

7. It is clear that the most probable words are not very discriminative. One way to measure how much information a word (feature) $X_j \in \{0, 1\}$ conveys about the class label $Y \in \{1, 2\}$ is by computing the **mutual information** between X_j and Y, denoted $I(X_j, Y)$, and defined as

$$mi(j) = I(X_j, Y) \quad = \quad \sum_{x=0}^{1} \sum_{c=1}^{2} p(X_j = x, Y = c) \log \frac{p(X_j = x, Y = c)}{p(X_j = x)p(Y = c)} \tag{12}$$

If we assume equal class priors, $p(Y = 1) = p(Y = 2) = 0.5$, then

$$p(X_j = 1, Y = c) = p(X_j|Y = y)p(Y = c) = \frac{\theta_{jc}}{2} \tag{13}$$

Use the provided function `MI` to compute the 5 words with the highest mutual information with the class label. (Use the θ's estimated on `xtrain,ytrain`.) List the words along with the corresponding values of `MI`. (As a sanity check, the first word should be "windows" with an MI of 0.2150.)

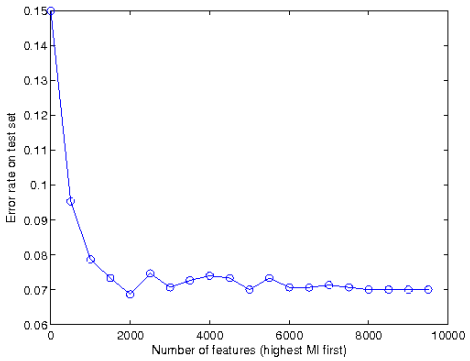


Figure 2: Test error rate vs number of features, in order of decreasing MI.

8. You will now apply the code you wrote in the previous question to a much larger data set (so your computer will need much more time and memory: on my laptop, with 1GB of RAM, it takes about 4–5 minutes to solve this part of the problem.) Consider the problem of classifying email messages into spam and non-spam (ham). There are 1000 spam emails and 2000 ham emails in our collection; we divided them into training and text sets of equal size and class ratio. We consider 10,000 words. Upon loading `emaildata.mat`, the Matlab environment will contain

```
Name          Size               Bytes  Class     Attributes
 vocab          1x10000           745408  cell
  xtest       1500x10000          4043516  double    sparse
  xtrain      1500x10000          4043516  double    sparse
  ytest          1x1500             12000  double
  ytrain         1x1500             12000  double
```

   Now `xtrain(i,w)` contains the *number of times* word w occurs in document i; thus it is not a binary matrix. `vocab` is a cell array, as before. Convert the X data (training and test) to binary (word present/ absent) form and fit a naive Bayes classifier using your code from above. What is the training and test error? Hint: if you run out of memory, use the 'clear' command to remove variables once they are no longer needed. Also, consider using single precision.

9. What are the 5 most likely words in each class? List them with their probability.

10. Use the provided function `MI.m` to compute the 5 words with the highest mutual information with the class label. (Use the θ's estimated on `xtrain,ytrain`.) List the words along with the corresponding values of `MI`. (As a sanity check, the first word should be "our" with an MI of 0.23376.)

11. Plot the error rate on the test set as a function of the number of features used, in order of decreasing mutual information. You should get a plot like Figure 2 (using a number of features in the range 5:500:10000). Turn in your code and plot. How many features should we use, and why?