EECS445: Introduction to Machine Learning, Fall 2014
Homework #2

Due date: 6pm on 10/7 (Tuesday)

Reminder: While you are encouraged to think about problems in small groups, all written solutions must be independently generated. Please type or hand-write solutions legibly. While these questions require thought, please be as concise and clear in your answer as possible. Please address all questions to `http://piazza.com/class#fall2014/eecs445` with a reference to the specific question in the subject line (E.g. RE: Homework 2, Q2(c)). For any solutions that require programming, please submit commented code along with any figures that you are asked to plot.

Remarks about notation: As in the lectures, we will interchangeably use $\mathbf{x}_i$ and $\mathbf{x}^{(i)}$ to denote the i-th training example (and similar notation for other variables). Often, it's less confusing to use $\mathbf{x}^{(i)}$ than $\mathbf{x}_i$ (especially in handwriting) when your expression involves another subscripts to denote specific coordinates in the vector. The Bishop's book uses the former notation, but some other machine learning textbooks use the latter notation. In addition, to be consistent with the Bishop's book, we use t to denote the labels. You are free to use whichever notation as long as you are clear about it. However, please do not mix them in each problem.

Submission Instructions:
There are two things you need to turn in:
1. A hard-copy of your work.
2. A file that has all your code.

Your hard-copy must be turned in at the Learning Center (BBB 1637). As you enter the Learning Center, immediately to your right, there is a cabinet. The top drawer of the cabinet is labeled "EECS445 - Fall 2014". Drop your hand-written assignment there. Make sure your name is written on your assignment, and that all your pages are stapled together properly. Please write down the names of people with whom you discussed the problems. In addition, please write the date/time of submission.

What to include in your hard-copy:
1. Your detailed answers to all questions. In the front page, please write your name, uniquename, names of collaborators, date/time of submission.
2. Your printed-out code if it is a programming assignment.
3. All figures, plots, output values asked by the problem.

Your program file must be submitted online using CTools. For every sub-question, you need to make one executable file that can be executed by a single command. This file must be named as *q[#question]solution_[#subquestion].m* where *#question* is the number of the question, and *#subquestion* is the number of the sub-question. For example, if you are solving Q1(a), then the filename should be *q1solution_a.m*. As another example, if the question Q4 has only one programming sub-question, you can use *q4solution.m* (without specifying the sub-question)—if so, there should be only one file with prefix "*q4solution*".

Sometimes, the solution code for consecutive sub-questions could be redundant. In general, it's better to make things redundant than unclear. So, it's okay to copy and paste some code for different sub-questions. For example, Q1(a) could be subsumed by Q2(b). Even in this case, please submit *q1solution_a.m* and *q1solution_b.m* separately (through zip file to be submitted to Ctools).

[Note: for the *print-out version* of your homework, if some sub-questions are "subsumed" by later sub-questions, you can just print out the most comprehensive version. For example, if Q1(a) could be subsumed by Q1(b), then you can simply print-out the q1b. The main purpose is to reduce clutter in the print-out version. We will primarily base our grading on your online solution, but we may optionally give full points if your printed-out version of the code is correct (after visual inspection) without any doubt.]

You need to organize your code into separate folders, one folder for each question (for example, folder names should be *q1, q2*, etc.), and zip/gzip all your folders together (so that you have a single zip file that will include all the directories).

You zipped/gzipped file needs to be named *yourLastName_yourFirstName.zip* (or *yourLastName_yourFirstName.tgz*). Here, please use your official name, as registered by the University system. Finally, please upload this zip/gzip file to cTools to receive credit.

IMPORTANT: In grading, the grader will download the zip file and unzip it. Then he/she will go to each folder and run the *q[#question]solution_[#subquestion].m* file. You will only receive full credit if your file can execute and produce the correct result. So please make sure you include all function files and data files in the correct directory. Your grader is not responsible for debugging your code (e.g., having to figuring out which code to run due to unconventional filenames, or check data is loaded properly, etc.) to give partial credit.

1 [30 points] Gaussian Discriminate Analysis

Suppose we are given a dataset $\{(\mathbf{x}^{(i)}, t^{(i)}); i = 1, ..., N\}$ consisting of N independent examples, where $\mathbf{x}^{(i)} \in \mathbb{R}^M$ are M-dimensional vectors, and $t^{(i)} \in \{0, 1\}$. We will model the joint distribution of $(\mathbf{x}, t)$ according to:

$$p(t) = \phi^t (1-\phi)^{1-t}$$

$$p(\mathbf{x}|t=0) = \frac{1}{(2\pi)^{\frac{M}{2}}|\Sigma|^{\frac{1}{2}}} \exp(-\frac{1}{2}(\mathbf{x}-\mu_0)^T\Sigma^{-1}(\mathbf{x}-\mu_0))$$

$$p(\mathbf{x}|t=1) = \frac{1}{(2\pi)^{\frac{M}{2}}|\Sigma|^{\frac{1}{2}}} \exp(-\frac{1}{2}(\mathbf{x}-\mu_1)^T\Sigma^{-1}(\mathbf{x}-\mu_1))$$

Here, the parameters of our model are ϕ, Σ, μ_0, and μ_1. (Note that while there are two different mean vectors μ_0 and μ_1, there is only one covariance matrix Σ.)

(a) **[8 points]** Suppose we have already fit ϕ, Σ, μ_0, and μ_1, and now want to make a prediction at some new query point $\mathbf{x}$. Show that the posterior distribution of the label at $\mathbf{x}$ takes the form of a logistic function, and can be written as

$$p(t=1|\mathbf{x}; \phi, \Sigma, \mu_0, \mu_1) = \frac{1}{1+\exp(-\mathbf{w}^T\mathbf{x})}$$

where $\mathbf{w}$ is some appropriate function of ϕ, Σ, μ_0, and μ_1. (Note: To get your answer into the form above, for this part of the problem only, you may have to redefine the $\mathbf{x}^{(i)}$'s to be $M+1$-dimensional vectors by adding the extra coordinate $x_0 = 1$, like we did in class.)

(b) **[17 points]** For this part of the problem only, you may assume M (the dimension of $\mathbf{x}$) is 1, so that $\Sigma = \left[\sigma^2\right]$ is just a real number, and likewise the determinant of Σ is given by $|\Sigma| = \sigma^2$. Given the dataset, we claim that the maximum likelihood estimates of the parameters are given by

$$\phi = \frac{1}{N}\sum_{i=1}^{N} 1\{t^{(i)} = 1\}$$

$$\mu_0 = \frac{\sum_{i=1}^{N} 1\{t^{(i)} = 0\}\mathbf{x}^{(i)}}{\sum_{i=1}^{N} 1\{t^{(i)} = 0\}}$$

$$\mu_1 = \frac{\sum_{i=1}^{N} 1\{t^{(i)} = 1\}\mathbf{x}^{(i)}}{\sum_{i=1}^{N} 1\{t^{(i)} = 1\}}$$

$$\Sigma = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}^{(i)} - \mu_{t^{(i)}})(\mathbf{x}^{(i)} - \mu_{t^{(i)}})^T.$$

The log-likelihood of the data is

$$\ell(\phi, \mu_0, \mu_1, \Sigma) = \log \prod_{i=1}^{N} p(\mathbf{x}^{(i)}, t^{(i)}; \phi, \mu_0, \mu_1, \Sigma)$$

$$= \log \prod_{i=1}^{N} p(\mathbf{x}^{(i)} | t^{(i)}; \phi, \mu_0, \mu_1, \Sigma) p(t^{(i)}; \phi)$$

By maximizing ℓ with respect to the four parameters, prove that the maximum likelihood estimates of ϕ, Σ, μ_0, and μ_1 indeed as given in the formulas above. (You may assume that there is at least one positive and one negative example, so that the denominators in the definitions of μ_0 and μ_1 above are non-zero.)

(c) **[7 extra credit points]** Redo part (b) without assuming $M = 1$.

(d) **[5 points]** Using the M.L.E of the parameters given in part (b), compute each parameter's value. Load the file `q1_data.mat`, which contains the variables `q1x_train`, `q1y_train`, `q1x_test`, `q1y_test`. First, use training data to estimate parameter values. Report your estimation result. Then, use your result and your answer in (a) to classify test data. Report your test accuracy.
Helpful Matlab command: `sum, mean, repmat`.
Matlab tips: given two matrices, $\mathbf{X}$ with dimension of $n * m$ and $\mathbf{Y}$ with dimension of $n * 1$, in Matlab `X(Y==1,:)` will return rows in $\mathbf{X}$ with a corresponding y value of 1.

2 [35 points] Softmax Regression via Gradient Ascent

Gradient Ascent is an algorithm used to find parameters that maximize a certain expression (contrary Gradient *Descent*, that is used to *minimize* an expression). For some function $f(\mathbf{w})$, Gradient Ascent finds $\mathbf{w}^* = \arg\max_{\mathbf{w}} f(\mathbf{w})$ according to the following pseudo-code:

Algorithm 1: Gradient Ascent

$\mathbf{w}^* \leftarrow$ *random* ;
repeat
  $\mathbf{w}^* \leftarrow \mathbf{w}^* + \alpha \nabla_{\mathbf{w}} f(\mathbf{w}^*)$;
until *convergence*;
return $\mathbf{w}^*$

Further, Softmax regression is a multiclass classification algorithm. Given a dataset $\mathcal{D} = \{(\mathbf{x}^{(1)}, t^{(1)}), \ldots, (\mathbf{x}^{(N)}, t^{(N)})\}$, where $t^{(i)}$ can be one of $\{1, 2, \ldots, K\}$ (a K-class classification setting), the Softmax regression computes the probability on example $\mathbf{x}$ belonging to class k as:

$$p(t = k|\mathbf{x}, \mathbf{w}) = \frac{\exp\left(\mathbf{w}_k^T \phi(\mathbf{x})\right)}{\sum_{j=1}^{K} \exp\left(\mathbf{w}_j^T \phi(\mathbf{x})\right)}$$

Where $\mathbf{w}_k$ is a weight vector for class k. The above expression is **over-parametrized**, meaning that there is more than one unique $\{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_K\}$ that give identical probability measures for $p(t|\mathbf{x}; \mathbf{w})$. It is possible (and common) to obtain a unique solution by using only $k-1$ class weight vectors $\mathbf{w} = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_{K-1}\}$ by fixing $\mathbf{w}_K = 0$:

$$p(t = k|\mathbf{x}, \mathbf{w}) = \frac{\exp\left(\mathbf{w}_k^T \phi(\mathbf{x})\right)}{1 + \sum_{j=1}^{K-1} \exp\left(\mathbf{w}_j^T \phi(\mathbf{x})\right)}; \text{ for } k = \{1, 2, \ldots, K-1\}$$

$$p(t = K|\mathbf{x}, \mathbf{w}) = \frac{1}{1 + \sum_{j=1}^{K-1} \exp\left(\mathbf{w}_j^T \phi(\mathbf{x})\right)}$$

We define the likelihood over the i-th training example $p(t^{(i)}|\mathbf{x}^{(i)}; \mathbf{w})$ as:

$$p(t^{(i)}|\mathbf{x}^{(i)}; \mathbf{w}) = \prod_{k=1}^{K} \left[p(t^{(i)} = k|\mathbf{x}^{(i)}; \mathbf{w})\right]^{\mathbf{I}(t^{(i)}=k)}$$

Where $\mathbf{I}(.)$ is the indicator function. We define the likelihood (probability over training data) as:

$$L(\mathbf{w}) = \prod_{i=1}^{N} p(t^{(i)}|\mathbf{x}^{(i)}; \mathbf{w}) = \prod_{i=1}^{N} \prod_{k=1}^{K} \left[p(t^{(i)} = k|\mathbf{x}^{(i)}; \mathbf{w})\right]^{I(t^{(i)}=k)}$$

Finally, we define the log-likelihood as:

$$\begin{aligned} l(\mathbf{w}) = \log L(\mathbf{w}) &= \log \prod_{i=1}^{N} \prod_{k=1}^{K} \left[p(t^{(i)} = k|\mathbf{x}^{(i)}; \mathbf{w})\right]^{I(t^{(i)}=k)} \\ &= \sum_{i=1}^{N} \sum_{k=1}^{K} \log \left[p(t^{(i)} = k|\mathbf{x}^{(i)}; \mathbf{w})\right]^{I(t^{(i)}=k)} \end{aligned}$$

(a) **[18 points]** Derive the gradient ascent update rule over the log-likelihood of the training data. In other words, derive the expression for $\nabla_{\mathbf{w}_m} l(\mathbf{w})$ for $m = 1, \ldots, K-1$. Show that:

$$\begin{aligned} \nabla_{\mathbf{w}_m} l(\mathbf{w}) &= \sum_{i=1}^{N} \phi(\mathbf{x}^{(i)}) \left[\mathbf{I}(t^{(i)} = m) - \frac{\exp\left(\mathbf{w}_m^T \phi(\mathbf{x}^{(i)})\right)}{1 + \sum_{j=1}^{K-1} \exp\left(\mathbf{w}_j^T \phi(\mathbf{x}^{(i)})\right)}\right] \\ &= \sum_{i=1}^{N} \phi(\mathbf{x}^{(i)}) \left[\mathbf{I}(t^{(i)} = m) - p(t^{(i)} = m|\mathbf{x}^{(i)})\right] \end{aligned}$$

[Hints: $\log a^b = b\log(a)$*. Further, it helps to consider the two cases separately; a case for for* $t^{(i)} = k = m$*, and another for* $t^{(i)} = k \neq m$*, which is equivalent to using Kronecker delta*[1] δ_{km} *]*

(b) **[17 points]** Using the gradient computed in part (a), implement Gradient Ascent for Softmax Regression in Matlab. Use a learning rate $\alpha = 0.0005$. Load the file `q2_data.mat`, which contains the variables `q2x_train`, `q2t_train`, `q2x_test`, `q2t_test`. Train your classifier on the training data and report the accuracy on the test data. Recall that Softmax Regression classifies an example $\mathbf{x}$ as:

$$y = \arg\max_{y'} p(y'|\mathbf{x}; \mathbf{w})$$

While you must write your own Softmax Regression implementation, feel free to double-check your accuracy by making use of Matlab's `mnrfit()`. If your accuracy is much less than the accuracy produced by Matlab's implementation, that means you did something wrong! In theory, you should get identical accuracy to Matlab's implementation (if you have a good stopping criterion, or do a large number of iterations) as this log-likelihood function is concave. Another powerful tool that we recommend you to use and check your answer is LIBLINEAR which is more popular and widely used. You can find instructions and installation package here: `http://www.csie.ntu.edu.tw/~cjlin/liblinear/`

3 [35 points] Naive Bayes for classifying SPAM

In this problem, we will use the naive Bayes algorithm to build a SPAM classifier[2].

In recent years, spam on electronic newsgroups has been an increasing problem. Here, we'll build a classifier to distinguish between "real" newsgroup messages, and spam messages. For this experiment, we obtained a set of spam emails, and a set of genuine newsgroup messages. Using only the subject line and body of each message, we'll learn to distinguish between the spam and non-spam.

All the files for the problem are available on CTools. The text emails have been preprocessed to get them into a form usable by naive Bayes. You can look at two sample spam emails in the files `spam_sample_original*`, and their preprocessed forms in the files `spam_sample_preprocessed*`. The first line in the preprocessed format is just the label and is not part of the message. The preprocessing ensures that only the message body and subject remain in the dataset; email addresses (EMAILADDR), web addresses (HTTPADDR), currency (DOLLAR) and numbers (NUMBER) were also replaced by the special tokens to allow them to be considered properly in the classification process. (In this problem, we'll going to call the features "tokens" rather than "terms," since some of the features will correspond to special values like EMAILADDR. You don't have to worry about the distinction.) The files `news_sample_original` and `news_sample_preprocessed` also give an example of a non-spam mail.

The work to extract feature vectors out of the documents has also been done for you, so you can just load in the design matrices (called document-term matrices in text classification) containing all the data. In a document-term matrix, the i^{th} row represents the i^{th} document/email, and the j^{th} column represents the j^{th} distinct token. Thus, the (i, j)-entry of this matrix represents the number of occurrences of the j^{th} token in the i^{th} document.

For this problem, we've chosen as our set of tokens considered (that is, as our vocabulary) only the medium frequency tokens. The intuition is that tokens that occur too often or too rarely do not have much classification value. (Examples tokens that occur very often are terms like "the," "and," and "of," which occur in so many emails and are sufficiently content-free that they aren't worth modeling.) Also, terms were stemmed using a standard stemming algorithm; basically, this means that "price," "prices" and "priced" have all been replaced with "price," so that they can be treated as the same term. For a list of the tokens used, see the file `TOKENS_LIST`.

Since the document-term matrix is extremely sparse (has lots of zero entries), we have stored it in our own efficient format to save space. You don't have to worry about this format. The file read `Matrix.m`

[1] https://en.wikipedia.org/wiki/Kronecker_delta

[2] Credit: Question adopted from `http://www.stanford.edu/class/cs229/ps/ps2.pdf`

provides the `readMatrix` function that reads in the document-term matrix and the correct class labels for the various documents. Code in `nb_train.m` and `nb_test.m` shows how `readMatrix` should be called. The documentation at the top of these two files will tell you all you need to know about the setup.

(a) **[15 points]** Implement a naive Bayes classier for spam classification, using the multinomial event model and Laplace smoothing. You should use the code outline provided in `nb_train.m` to train your parameters, and then use these parameters to classify the test set data by filling in the code in `nb_test.m`. You may assume that any parameters computed in `nb_train.m` are in memory when `nb_test.m` is executed, and do not need to be recomputed (i.e., that `nb_test.m` is executed immediately after `nb_train.m`). Train your parameters using the document-term matrix in `MATRIX.TRAIN`, and then report the test set error on `MATRIX.TEST`.

Remark. If you implement naive Bayes in the straightforward way, you'll note that the computed $p(\mathbf{x}|\mathbf{t}) = \prod_{\mathbf{i}} \mathbf{p}(\mathbf{x_i}|\mathbf{t})$ often equals zero. This is because $p(\mathbf{x}|t)$, which is the product of many numbers less than one, is a very small number. The standard computer representation of real numbers cannot handle numbers that are too small, and instead rounds them off to zero. (This is called "underflow.") You'll have to find a way to compute naive Bayes' predicted class labels without explicitly representing very small numbers such as $p(\mathbf{x}|t)$. [Hint: Think about using logarithms.]

(b) **[10 points]** Intuitively, some tokens may be particularly indicative of an email being in a particular class. We can try to get an informal sense of how indicative token i is for the SPAM class by looking at:

$$\log \frac{p(x_j = i|t = 1)}{p(x_j = i|t = 0)} = \log \left(\frac{p(\text{token}_i|\text{email is SPAM})}{p(\text{token}_i|\text{email is NOTSPAM})} \right)$$

Using the parameters fit in part (a), find the 5 tokens that are most indicative of the SPAM class (i.e., have the highest positive value on the measure above). The numbered list of tokens in the file `TOKENS_LIST` should be useful for identifying the terms/tokens.

(c) **[10 points]** Repeat part (a), but with training sets of size ranging from 50, 100, 200, . . . , up to 1400, by using the files `MATRIX.TRAIN.*`. Plot the test error each time (use `MATRIX.TEST` as the test data) to obtain a learning curve (test set error vs. training set size). You may need to change the call to `readMatrix` in `nb_train.m` to read the correct file each time. Which training-set size gives the best test set error?