CS177, Homework 3

Due: Wednesday, April 22

As always, please upload electronic files to EEE as a single archive, named after yourself.

Reading

Unfortunately our text does not have good coverage of classification or naïve Bayes. However, there are several online sources of reading listed on the course webpage. Take time to review your class notes, as well.

Assignment Summary

In this assignment you will implement an algorithm for estimating the probabilities used by a naïve Bayes classifier and will apply this to a data set which came from a set of real email. The assignment involves developing two functions in MATLAB and performing some experiments using those functions.

1. Write a function called nbayes learn.m that takes in training data and returns the probability tables for a naïve Bayes classifier (as discussed in class).

2. Write a function called nbayes predict.m that takes a set of test data vectors and returns a set of class label predictions {spam,non-spam} for each vector.

Templates for both of these functions is available on the website. Please comment your code. You should upload your Matlab code for these two functions along with a one-page discussion of your experiments (in a file called `experiments.txt` or `.doc` or `.pdf`) to the Homework 3 folder on EEE. You do not need to turn in hardcopies.

Data

For your assignment you will be working with a data set that was created a few years ago at Hewlett Packard Research Labs as a testbed data set to test different spam email classification algorithms. The HP researchers gathered a set of both spam and non-spam emails and computed a set of "feature values" (e.g., based on the presence of certain words) for each email. The idea is for an algorithm to try to predict whether an email is spam or not based on the features. On the class web page you find pointers to three files:

1. **Features:** The first file, called `spam_features.txt` is an ascii file containing $n = 4601$ rows and $d = 57$ columns. Each row corresponds to a particular email and each column indicates the presence or absence of that feature in the email. The j^{th} column thus gives the outcomes of a binary variable $X_j \in \{1, 2\}$ for the set of emails.

2. **Labels:** The second data file, `spam_labels.txt` consists of $n = 4601$ values which are the class labels for each of the emails. The label 2 corresponds to emails that were identified as "spam" and the label 1 corresponds to emails that were identified as "non-spam".

If you are interested in finding out more about where the data came from, you can read more here

- http://mlearn.ics.uci.edu/databases/spambase/spambase.DOCUMENTATION which describes in detail how the data was collected, what the features mean, basic statistics about the data, and so on. Note that the original features are real-valued (many of them are expressed as the percentage of words in an email that correspond to a particular word). So to create discrete-valued variables that we can use in a naïve Bayes classifier, we converted each of the original real-valued features to a binary random variable by making all values less than or equal to the median (across all 4601 rows for that feature) be 1 and all values above be 2.

Training a naïve Bayes classifier from data

You will implement a MATLAB function `nbayes_learn.m` for learning the classifier from data. The basic steps your function will need to perform are

1. Separate out examples into the two classes based on the labels stored in the class labels file.
2. Let n_1 and n_2 be the number of labeled examples that are in class 1 and class 2 respectively and let $n = n_1 + n_2$ be the total number of examples in the file. Estimate the marginal probability $p(C = i)$ for each class using the empirical frequency as follows:

$$p(C = i) = \frac{n_i}{n}, \qquad\qquad i = 1, 2$$

3. For each combination of class value i, feature X_j and feature value k, estimate the conditional probability $p(Xj = k|C = i)$ using the empirical frequency. Since there are a limited number of examples, use the following "smoothed" estimate as discussed in class:

$$p(X_j = k|C = i) = \frac{n_{i,j,k} + .5}{n_i + 1}, \qquad\qquad i = 1, 2, \quad j = 1, \ldots, d \quad k = 1, 2$$

   In this formula, $n_{i,j,k}$ is the number of training examples (rows) that were of class i and for which feature X_j (column j) took on value k. Again, n_i is the total number of rows of class i.

At this point, the training of the model is complete since we know all the probabilities necessary to specify the classifier. You should store the values in a MATLAB data structure `params` as described in the function templates on the website. A good thing to do is check that your calculations are correct by making sure that your probability distributions all sum to 1. For example, make sure that $\sum_k p(x_j = k|C = i) = 1$ holds for each choice of i and j.

Using the classifer to make predictions

Next you will use the classifier learned from the training data to make predictions about new test data vectors. Write a function `nbayes_predict.m` that finds the most likely class label, i.e., the class for which the probability $p(C = i|x_1, x_2, x_3, \ldots x_d)$ is highest. In this case, you will get the set of feature measurements $x_1, \ldots x_d$ but the class label C is unknown. In order to calculate $p(C = i|x_1, x_2, x_3, \ldots)$, you will need to invoke the assumption of that the x_j are conditionally independent given the class variable C. Given this assumption, for each class value i, we can compute

$$p(C = i|x_1, x_2, x_3, \ldots, x_d) = \frac{p(C = i) \prod_{j=1}^{d} p(x_j|C = i)}{p(x_1, x_2, \ldots, x_d)}$$

If we only care about which of the two classes ($C = 1$ or $C = 2$) has higher conditional probability, you don't need to compute the denominator, since it is the same in both cases. (For the last task of the homework, however, you will need to compute it.) For each of the test data vectors input to your function, evaluate the probability and predict the class with the larger probability. **Note:** If you calculate the product above directly, you may run into problems with very small probabilities getting rounded down to 0. We will discuss a solution to this problem in class, by doing the computation with log probabilities. Write your function so that it can take a set of test examples (a matrix where each row contains the 57 feature values) along with `params` and performs this calculation on each row of data, returning the set of predicted class variables, one for each row of input.

Testing the classifier performance

We would like to see how accurate our classifier is at making good predictions. To calculate the accuracy, we can run the classifier on a set of examples where we know the true labels, and then calculate the percentage of the test examples for which the classifier made a correct prediction. Perform the following set of experiments and include the requested items in your report:

- Train the classifier using only the first 1500 training examples and labels and then use the remaining rows (1501–4601) in order to compute the accuracy. To calculate the accuracy, run `nbayes_predict.m` on the remaining rows and compare the predicted class labels it returns to the true class labels, reporting accuracy as a percentage of correct predictions.

- Repeat this experiment but now train using only the first 50 examples and test on rows 1501–4601. Again report your accuracy.

- Repeat again, but now train using only the first 10 examples and test on rows 1501–4601. How does the accuracy vary across these 3 different training conditions?

- Suppose a classifier randomly guessed the class labels with equal probabilities of 0.5. How accurate would you expect it to be? How accurate would a classifier be that always predicted the class label which occurred most commonly in the training set (i.e., if the majority of examples 1–1500 were class i, it would always report class i regardless of input)? How do these two simple strategies compare to the performance of your system measured in parts 1-3?

- Write a third function which returns the probability of the class label C being spam and non-spam. Compute these probabilities for the first 20 examples in the data file. You do not have to turn in code for this function, just the table of values. You will probably want to use the code you have already written for `nbayes_predict.m` as a template and modify it slightly to return the class probabilities rather than just the most likely class.