CS4442b/9542b: Artificial Intelligence II, Winter 2014
Assignment 1: Spam/Ham Classification
Due: February 10

Instructions:

- Submit your assignment through WebCT by 11:55pm on the due date.
- Include all matlab code you write and a soft copy of answers to questions. You do not need to include code/data that I give you.
- Make sure to use the function names, input/output parameters that I specify. If you write any helper functions, you can give them any names you wish.
- This assignment has extra credit questions. Extra credit counts only towards the assignments, it does not transfer to quizzes.

Data: In this assignment you will develop and test several Spam/Ham classifiers. I collected a dataset, extracted 99 features, divided them into testing/training sets. Download file *A1.mat* into matlab using command "load A1". This file has four matrices: $HamTrain(1082 \times 99)$, $SpamTrain(1080 \times 99)$, $HamTest(465 \times 99)$, $SpamTest(530 \times 99)$. The first two should be used for training, the last two for testing. Throughout the assignment, I refer to Ham as class 1 (and its class label is 1), and to Spam as class 2 (and its class label is -1, i.e." bad").

1. (10%) Let C be a column vector of classification results for Ham/Spam problem. Namely $C(i) = 1$ means that example i is a Ham, and $C(i) = -1$ means that example i is a Spam. Write a matlab function$[CONF, error] = summarizeResults(C1, C2)$, which takes as an input $C1$, a classification vector of examples whose true class is 1, and $C2$, a classification vector of examples whose true class is 2. The output should be a confusion matrix $CONF$ and classification error. $Error$ is the fraction of misclassified examples (i.e. number of correctly classified examles divided by the total number of examples). $CONF$ is a 2 by 2 matrix where $CONF(i, j)$ is the number of examples of class i that are classified as class j. Thus diagonal entries in $CONF$ are the correct classifications, and off-diagonal entries are mis-classifications. For example, if input vectors are $C(1) = \begin{bmatrix} -1 \\ 1 \\ 1 \\ 1 \\ -1 \end{bmatrix}$, and $C(2) = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix}$.

   Then the output should be: $CONF = \begin{bmatrix} 3 & 2 \\ 1 & 2 \end{bmatrix}$, $err = \frac{3}{8}$.

   Try to write this function without loops using Matlab $find$ command.

2. (15%)

   (a) Write function $[C1, C2] = classifyKnn(Train1, Train2, Test1, Test2, k)$ which performs kNN classification. The first four inputs are the training examples for two classes ($Train1$ and $Train2$) and the testing examples for two classes ($Test1$ and $Test2$). The

last input is the parameter k for kNN classifier. The outputs $C1$ and $C2$ are the classification vectors for $Test1$ and $Test2$, respectively. As usual, the first class is denoted by 1 and the second class by -1. Therefore $C1(i) = 1$ and $C2(i) = -1$ are correct classifications, and $C1(i) = -1$, $C2(i) = 1$ are incorrect classifications.

(b) Report the test error and confusion matrix for $k = 1, 3, 10$. It's convinient to use the function you wrote in the previous excercise.

(c) Let us investigate the importance of using many features for Ham/Spam classification. Fix $k = 1$. Find 3 consequitive features that give you the lowest test error. That is you should perform kNN classification using only features $\{1, 2, 3\}$, then only features $\{2, 3, 4\}$, ..., lastly only features $\{97, 98, 99\}$. Find and report the best error and the best feature subset. Compare with the test error for $k = 1$ using the full feature set.

3. (15%)

(a) Write a function $[C1, C2] = LinearMSE(Train1, Train2, Test1, Test2)$ which performs linear classification with MSE procedure. The inputs are exactly the same as the first four inputs in Problem 2, and the outputs are exactly the same as in Problem 2. Using Matlab *pinv* function is better than using the backslash operator in this case.

(b) Report the test error and confusion matrix for your Linear MSE-based classifier for the training and test data. To get the training error and confusion matrix, just plug in training data instead of testing, i.e. run $LinearMSE(Train1, Train2, Train1, Train2)$.

(c) Let us investigate the over-fitting problem with higher order polynomials. We will use a quadratic clssifier. Recall that this is equivalent to adding new features that are products of pairs of original features. In principle, we could add all possible pairwise features, but the running time would be too large. Therefore we will add only some pairwise features, namely those pairs that are no more than 20 apart. Specifically, let x be an example, which, in, our case, has 99 features. For any pair of original features x_i and x_j with $|i - j| \leq 20$, add a new feature $x_{i,j}$ whose value is $x_i \cdot x_j$. Therefore the set of all new features is

$$
\begin{array}{l}
x_{1,1}, x_{1,2}, ...x_{1,21}, \\
x_{2,2}, x_{2,3}, ...x_{2,22}, \\
x_{3,3}, x_{3,4}, ...x_{3,23}, \\
..... \\
x_{79,79}, x_{79,80}, ...x_{79,99},
\end{array}
$$

You should get 1758 features in total, of which 99 are old and the rest are new. Report the training and test errors and discuss them in contrast with the training and test errors of the linear classifier in part (a).

4. (20%)

(a) Write a function $aBest = PerceptronBatch(Train1, Train2, k)$ which takes as an input training samples from two classes and runs the Perceptron batch rule for k iterations with a constant learning rate $\alpha = 1$. The output should be the best weight vector a that you are able to find during training. To implement this, use a simple modification to Perceptron algorithm discussed in class. As you iterate, keep track of the best weight a found so far, that is a that gives the smallest number of misclassified examples.

(b) Write function $[C1, C2] = LinearClassifier(a, Test1, Test2)$ that performs linear classification. The inputs are the weights vector a to use for linear classification, and the test samples of class 1 and class 2, in $Test1$ and $Test2$, respectively. Test samples are not augmented and not normalized. The output vectors are as in Problem 2.

(c) Run $PerceptronBatch$ for $k = 100, 1000, 10000, 100000$. Compute and write down the error rates and confusion matrices for these four cases, using functions $LinearClassifier$ and $SummarizeResults$. Discuss the results and compare them with those for the linear classifier in Problem 3(a).

5. (30%) In this problem, you will develop a boosted classifier based on simple single feature classifiers discussed in class. In particular, they have the following form:

$$\begin{cases} (threshold - x_i) > 0 \Rightarrow & class1 \\ (threshold - x_i) \leq 0 \Rightarrow & class2 \end{cases} \quad \text{and} \quad \begin{cases} (threshold - x_i) < 0 \Rightarrow & class1 \\ (threshold - x_i) \geq 0 \Rightarrow & class2 \end{cases}$$

To simplify implementation, instead of using 2 different classifier types as above, we introduce an additional parameter *polarity* which takes only 2 values: 1 and -1. We can now use the following single classifier type, completely equivalent to the two above (with $polarity = 1$ it is the classifier on the left, with $polarity = -1$, it is the classifier on the right):

$$\begin{cases} polarity \cdot (threshold - x_i) > 0 \Rightarrow & class1 \\ polarity \cdot (threshold - x_i) \leq 0 \Rightarrow & class2 \end{cases}$$

How many classifiers of this type are there? The classifier above depends on polarity, feature i and threshold. For polarity, we have 2 choices, for feature, we have d choices, where d is the dimension of our examples. What about *threshold*? At the first glance it seems like we have to consider a continuous range of values for the *threshold*. However, only a finite number of *threshold* values actually lead to different classification results. How many *threshold* values are there that make a difference in classification depend on the chosen feature i. Suppose we have only four samples and let their values for feature i be

$$\{0.5, 1.5, 1.5, 9.4\}.$$

That is there are only three unique values for feature i. Consider thresholds of 2 and 3. These two different thresholds result in exactlythe same classification. Namely, for $polarity = 1$, the first three samples get assigned to class 1 and the last sample to class 2. In fact, any *threshold* between 1.5 and 9.4 gives exactly the same classifier. Therefore the number of unique classifiers for feature i is only 4: one t*hreshold* less than 0.5, another in between 0.5 and 1.5, another in between 1.5 and 9.4, and the last one larger than 9.4. You can use Matlab function *unique* to find all the unique values for a feature.

(a) Write a function

$$[features, thresholds, polarities, alphas] = boost(Train1, Train2, t)$$

that performs boosting based on single feature classifiers discussed above. The input are the training data for the two classes, and t, the number of iterations for boosting. The output are the features, thresholds, polarities, and alphas stored as column vectors of length t. These output vectors specify the boosted classifier

$$H(x) = \sum_{i=1}^{t} alphas(i) \cdot sign(polarities(i) \cdot (thresholds(i) - x_{features(i)}),)$$

where x_k is the kth feature of example x, $feature(i)$ is the feature, $alpha(i)$ is the weak classifier weight, $polarity(i)$ is the polarity chosen at round i of boosting. For example, suppose we boost for 3 rounds and at each round:

- Round1: $feature$ 3, $threshold = 3.3$, $polarity = 1$, $alpha = 0.4$
- Round1: $feature$ 1, $threshold = 5.3$, $polarity = -1$, $alpha = 0.5$
- Round1: $feature$ 10, $threshold = 7.1$, $polarity = 1$, $alpha = 0.2$

The corresponding discriminant function is:

$$H(x) = 0.4 \cdot sign\left(1 \cdot (3.3 - x_3)\right) + 0.5 \cdot sign\left(-1 \cdot (5.3 - x_1)\right) + 0.2 \cdot sign\left(1 \cdot (7.1 - x_{10})\right),$$

and the output of your function *boost* should be:

$$features = \begin{bmatrix} 3 \\ 1 \\ 10 \end{bmatrix}, thresholds = \begin{bmatrix} 3.3 \\ 5.3 \\ 7.1 \end{bmatrix}, polarities = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}, alphas = \begin{bmatrix} 0.4 \\ 0.5 \\ 0.2 \end{bmatrix},$$

In order to resample the data according to distribution d, use my function $[CNew1, CNew2] = resampleData(C1, C2)$. The inputs are the samples from the two classes and the desired distribution d, as a column vector. Notice that d holds the weights for both classes, first those for class 1 and then those for class 2. Thus the length of d is equal to the number of samples in $C1$ and $C2$. $CNew1$ and $CNew2$ are the resampled classes 1 and 2. Notice that due to the nature of sampling, their sizes may be slightly different from the sizes of $C1$ and $C2$, but it does not effect the performance of boosting.

(b) Perform boosting for 100 iterations. Then apply the boosted classsifier using *ApplyBoost* function that I provide. Write down the testing error and the confusion matrix.

(c) What are the feature numbers for the 5 most useful features? These are the features selected at the first five rounds. If you want, you can figure out what they are by looking at my function $ExtractFeatures$.

6. (10%)

(a) Competition time! Write function $[C1, C2] = BestClassifier(Test1, Test2)$ which takes test samples for classes 1 and 2 and outputs classification vectors. You can use any classifier you have developed in the previous problems. Choose the classifier that you think will perform the best on new data. If you need any data inside your classifier, store it in a file, say "myData.mat" and put in command "load myData" as the first line in your function. For the classifier of your choice, it makes sense to retrain it on all data I gave you (both training and test) for the best performance. I will run your $BestClassifier$ on my own test data. Whoever wins this competition, gets a recognition in class and a useless prize.

(b) The spam/ham data I collected is mostly old. Let us check how your best classifier developed in part (a) works on more recent email. I will post new data on the web site, and you report and discuss performance (error rate and the confusion matrix) on the new data.

7. (Extra Credit, 10%) Try to improve your classifier performance. You can change classifiers themselves or improve/add features. I put original emails and my feature extraction algorithms on line if you wish to change the features. You can submit this classifier for the competition.