Introduction to Machine Learning

Homework 1: Supervised ML and the Perceptron algorithm

Important: *See problem set policy on the course web site.*

In this problem set you will implement the Perceptron algorithm and apply it to the problem of e-mail spam classification.

Instructions. You are to use the Python programming language. For this assignment, you are **not** permitted to use or reference any machine learning code or libraries not written by yourself. You are also **not** permitted to use numpy. In addition to your answers to the questions below, submit all code that you write for this assignment.

Data files. We have provided you with two files: `spam_train.txt`, `spam_test.txt`. Each row of the data files corresponds to a single email. The first column gives the label (1=spam, 0=not spam).

Pre-processing. The dataset included for this exercise is based on a subset of the SpamAssassin Public Corpus. Figure 1 shows a sample email that contains a URL, an email address (at the end), numbers, and dollar amounts. While many emails would contain similar types of entities (e.g., numbers, other URLs, or other email addresses), the specific entities (e.g., the specific URL or specific dollar amount) will be different in almost every email. Therefore, one method often employed in processing emails is to "normalize" these values, so that all URLs are treated the same, all numbers are treated the same, etc. For example, we could replace each URL in the email with the unique string "httpaddr" to indicate that a URL was present. This has the effect of letting the spam classifier make a classification decision based on whether any URL was present, rather than whether a specific URL waspresent. This typically improves the performance of a spam classifier, since spammers often randomize the URLs, and thus the odds of seeing any particular URL again in a new piece of spam is very small.

We have already implemented the following email preprocessing steps: lower-casing; removal of HTML tags; normalization of URLs, e-mail addresses, and numbers. In addition, words are reduced to their stemmed form. For example, "discount", "discounts", "discounted" and "discounting" are all replaced with "discount". Finally, we removed all non-words and punctuation. The result of these preprocessing steps is shown in Figure 2.

1. In this homework assignment, you will implement a few variants of the Perceptron algorithm. Before you can build these models and measure their performance, split your

```
> Anyone knows how much it costs to host a web portal?
> Well, it depends on how many visitors youre expecting. This can be anywhere from
less than 10 bucks a month to a couple of $100. You should checkout
http://www.rackspace.com/ or perhaps Amazon EC2 if youre running something big..

To unsubscribe yourself from this mailing list, send an email to: groupname-
unsubscribe@egroups.com
```

Figure 1: Sample e-mail in SpamAssassin corpus before pre-processing.

```
anyon know how much it cost to host a web portal well it depend on how mani visitor
your expect thi can be anywher from less than number buck a month to a coupl of
dollarnumb you should checkout httpaddr or perhap amazon ecnumb if your run
someth big to unsubscrib yourself from thi mail list send an email to emailaddr
```

Figure 2: Pre-processed version of the sample e-mail from Figure 1.

training data (i.e. `spam_train.txt`) into a training and validate set, putting the last 1000 emails into the validation set. Thus, you will have a new training set (call it `train.txt`) with 4000 emails and a validation set (call it `validation.txt`) with 1000 emails. You will not use `spam_test.txt` until the end of the assignment.

Your first task will be to create the vocabulary from train.text. To this end, write a function create_vocabulary(filename, thresh) that takes as input a file (in the format of train.txt) and also a threshold value (integer) and returns the vocabulary as a list. The job of this function is to put in a list all the words in filename that appear in more than thresh emails. Ignoring words that appear fewer than thresh emails is both a means to limit overfitting (to be discussed in class) and speed up computation. Typically values of thresh range from 10 to 200. The default value of thresh is 40 unless otherwise stated.

2) Now that you have the vocabulary, the next step is to create feature vectors. Write a function create_feature_vectors(filename, vocabulary) that takes as input a file (in the format of train.txt) and a vocabulary (list) and produces x and y, where x is 2-dimensional list and y is a one dimensional list. The length of y is number of emails in filename. Each entry in y should by +1 or -1 depending on whether the corresponding email is spam or not. There is one row in x for every email; each row consists of a list of length len(vocabulary), where each element is 1 or 0 depending on whether the corresponding word in the vocabulary appears in the email. In your main function, let x_train, y_train = create_feature_vectors(train.txt, vocabulary), x_validate, y_validate = create_feature_vectors(validate.txt, vocabulary), and x_test, y_test = create_feature_vectors(test.txt, vocabulary).

3) Write a function perceptron_train(x_train, y_train) that returns **w**, updates, epochs, where **w** the final classification vector (as a Python list), updates is the number of updates (mistakes) performed (integer), and epochs is the number of passes through the data (integer). You may assume that the input data provided to your function is linearly separable (so the stopping criterion should be that all points are correctly classified). It is recommended that you first write a function to perform the inner product of two vectors, and have perceptron_train() call this function.

For this exercise, you do not need to add a bias feature to the feature vector (it turns out not to improve classification accuracy, possibly because a frequently occurring word already serves this purpose). Your implementation should cycle through the data points in the order as given in the data files (rather than randomizing), so that results are consistent for grading purposes.

4) Now write the function error(w, x, y), which returns the error rate, i.e., the fraction of examples that are misclassified by **w** for feature vectors x and corresponding labels y. Test your functions by determining the error rate with x = x_train, y = y_train. (It should be equal to zero.) Also determine

the error rate with x = x_validate and y = y_validate. Report the error rate. Also report the number of updates and epochs that occurred during training.

5) To better understand how the spam classifier works, we can inspect the parameters to see which words the classifier thinks are the most predictive of spam. Using the vocabulary list together with the parameters w learned in the previous question, write a function find_important_words(vocabulary, w) that returns the 12 words with the most positive weights and the 12 words with the most negative weights.

6) One should expect that the test error decreases as the amount of training data increases. Modify perceptron_train() to also take as input M, where M corresponds to using only the first *M* rows of your training data. Run the perceptron algorithm on this smaller training set and evaluate the corresponding validation error (using all of the validation data). Do this for *M* = 200, 600, 1200, 2400, 4000, and create a plot of the validation error as a function of *M*. Also plot the number of epochs (number of passes through the training data) for the different values of *M*. As the amount of training data increases, the margin of the training set decreases, which generally leads to an increase in the number of iterations perceptron takes to converge (although it need not be monotonic).

7) The later iterations typically perform updates on only a small subset of the data points, which can contribute to over-fitting. A way to solve this is to control the maximum number of iterations of the perceptron algorithm. Add an argument max_epochs to the perceptron algorithm that controls the maximum number of passes over the data. (Note that by limiting the number of passes, the training data may no longer be linearly separated by the resulting hyperplane **w**.) (Also, the algorithm should terminate before max_epochs is reached if all the training data becomes correctly classified.) Using all 4000 training emails, report the error rates for max_epochs =10, 15, 20, and Z, where Z is the number of epochs during training in 4).

8) Your algorithm now has two hyper-parameters: thresh and max_epochs. Write a program that does a search over these two hyper-parameters (step sizes of 10 for thresh and step sizes of 5 for max_epochs), all with M = 4000. The goal is to find the hyper-parameter combination that has the lowest validation error. What is the best hyper-parameter setting and its corresponding validation errors? (Note that each hyper-parameter setting will have its own **w**.)

9) Is now time to determine the test error using the data in spam_test.txt. To do this, you will use the parameters **w** and the hyper-parameters thresh, and max_epochs obtained in part (8). Report the test error for the best hyper-parameter setting. This is your final test error, and is what you report to your boss.

10) Describe in words why we need a training set, validation set, and test set (three disjoint sets of emails).

Acknowledgement: This problem set is based on a problem set developed by David Sontag, who in turn was inspired by a question developed by Andrew Ng.