Introduction to Machine Learning

Problem Set 1: Perceptron algorithm, Validation, Over-fitting

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. In addition to your answers to the below questions, hand in a print out of 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 was present. 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. This problem set will involve your implementing several 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 first 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.

Transform all of the data into feature vectors as follows. Build a vocabulary list using only the 4000 e-mail training set by finding all words that occur across the training set. Note that we assume that the data in the validation and test sets is completely unseen when we train our model, and thus we do not use any information contained in them. Ignore all words that appear in fewer than $X = 26$ e-mails of the 4000 e-mail training set -- this is both a means of preventing over-fitting (to be discussed in class) and of improving scalability. To do this, write a function words(data, X) which takes train.txt as input and returns a Python list containing all the words that occur in at least 26 emails.

For each email (in all three files), transform it into a feature vector **x** where the *j*th entry, x_j is 1 if the *j*th word in the vocabulary occurs in the email, and 0 otherwise. To do this write a function feature_vector(email) that takes as input a single email and returns the corresponding feature vector as a Python list.

2. Implement the functions perceptron_train(data) and perceptron_error(w, data). The function perceptron_train(data) trains a perceptron classifier using the examples provided to the function, and should return **w**, *k*, and *iter*, the final classification vector (as a Python list), the number of updates (mistakes) performed (integer), and the number of passes through the data (integer), respectively. 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).

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.

The function perceptron_error(w, data) should take as input the weight vector **w** and a set of examples. The function should return the error rate, i.e., the fraction of examples that are misclassified by **w**.

3. Train the linear classifier using your training set. How many mistakes are made before the algorithm terminates? Test your implementation of perceptron_error by running it with the

learned parameters and the training data, making sure that the training error is zero. Next, classify the emails in your validation set. What is the validation error?

4. 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 learned in the previous question, output the 12 words with the *most positive* weights. What are they? Which 12 words have the most *negative* weights?

5. One should expect that the test error decreases as the amount of training data increases. Using only the first N rows of your training data, run both the perceptron algorithm on this smaller training set and evaluate the corresponding validation error (using all of the validation data). Do this for N = 200, 600, 1200, 2400, 4000, and create a plot of the validation error as a function of N.

6. Also for N = 200, 600, 1200, 2400, 4000, create a plot of the number of perceptron iterations as a function of N, where by iteration we mean a complete pass through the training data. 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. One consequence of this is that 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 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.)

8. Congratulations, you now understand various properties of the perceptron algorithm. Try various configurations of the algorithm on your own using all 4000 training points, and find a good configuration having a low error on your validation set. In particular, try changing the maximum number of iterations and try changing X =22,28. Report the validation error for several of the configurations that you tried; which configuration works best?

You are ready to train on the full training set, and see if it works on completely new data. Combine the training set and the validation set (i.e. use all of `spam_train.txt`) and learn using the best of the configurations previously found. What is the error on the test set (i.e., now you finally use `spam_test.txt`)?

9. Suppose we only consider words that appear in at least X=1500 emails. How many features are there? Is the data liely separable?

10. Describe in words why we have we need a training set, validation set, and text 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.