DS-GA 3001.007 Introduction to Machine Learning

Problem Set 1: Perceptron algorithm

Following the guidelines in the syllabus, please upload your solutions to Gradescope by 11:59pm on September 18.

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

Instructions. You should use Python for the programming language. You are not allowed to use or reference any machine learning packages for this assignment (such as sklearn)

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.

> 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.

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 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 with 4000 emails and a validation set with 1000 emails. You will not use `spam_test.txt`

   Explain why measuring the performance of your final classifier would be problematic had you not created this validation set.

2. Transform all of the data into feature vectors. 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 $X = 30$ e-mails of the 4000 e-mail training set – this is both a means of preventing overfitting and of improving scalability. For each email, transform it into a feature vector $\vec{x}$ where the ith entry, x_i, is 1 if the ith word in the vocabulary occurs in the email, and 0 otherwise.

3. Implement the functions `perceptron_train(data)` and `perceptron_test(w, data)`.

   The function `perceptron_train(data)` trains a perceptron classifier using the examples provided to the function, and should return $\vec{w}$, k, and $iter$, the final classification vector, the number of updates (mistakes) performed, and the number of passes through the data, 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 the corner case of $\vec{w} \cdot \vec{x} = 0$, predict the $+1$ (spam) class.

   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_test(w, data)` should take as input the weight vector $\vec{w}$ (the classification vector to be used) and a set of examples. The function should return the test error, i.e. the fraction of examples that are misclassified by $\vec{w}$.

4. Train the linear classifier using your training set. How many mistakes are made before the algorithm terminates? Test your implementation of `perceptron_test` 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?

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 learned in the previous question, output the 15 words with the *most positive* weights. What are they? Which 15 words have the most *negative* weights?

6. Implement the *averaged* perceptron algorithm, which is the same as your current implementation but which, rather than returning the final weight vector, returns the average of all weight vectors considered during the algorithm (including examples where no mistake was made). Averaging reduces the variance between the different vectors.

7. Add an argument to both the perceptron and the averaged perceptron that controls the maximum number of passes over the data. This is an important hyperparameter because for large training sets, the perceptron algorithm can take many iterations just changing a small subset of the point -- leading to overfitting.

8. Experiment with various maximum iterations on the two algorithms checking performance on the validation set. Optionally you can try to change X from question 2. Report the best validation error for the two algorithms.

9. Combine the training set and the validation set (i.e. us all of spam_train.txt) and learn using the best of the configurations previously found. You do not need to rebuild the vocabulary when re-training on the train + validate set. What is the error on the test set (i.e. spam_test.txt).