Introduction to Machine Learning

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

Xinyue Chen NetID: xc1305

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 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 jth entry, xj is 1 if the jth 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.

Written in “MainCode\_for\_Q1Q2Q3.py”

2. Implement the functions perceptron\_train(data) and perceptron\_error(w, data).

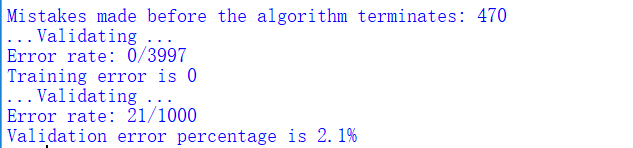
The function perceptron\_train(data) trains a perceptron classifier using the examples provided to the function, and should return w, k, and iter, the final classification vector (as a Python list), the number of updates (mistakes) performed (integer), and the number of passes through the data (integer),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

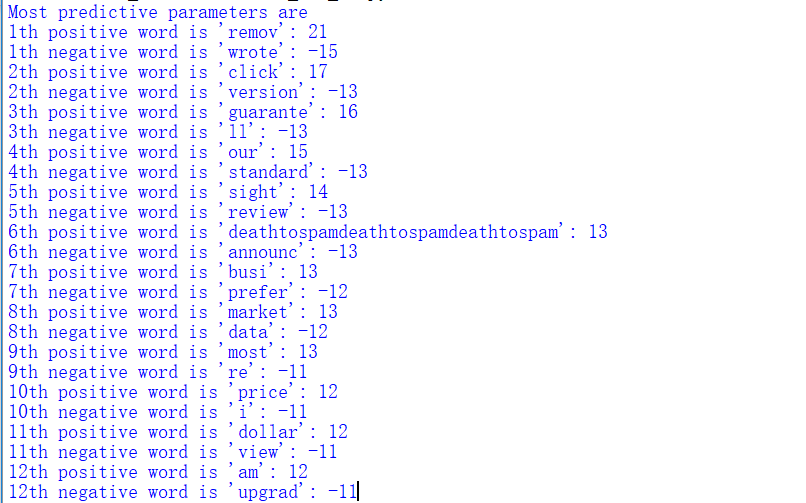
The function perceptron\_error(w, data) should take as input the weight vector w and a set of examples. The function should return the error rate, i.e., the fraction of examples that are misclassified by w.

Written in “MainCode\_for\_Q1Q2Q3.py”

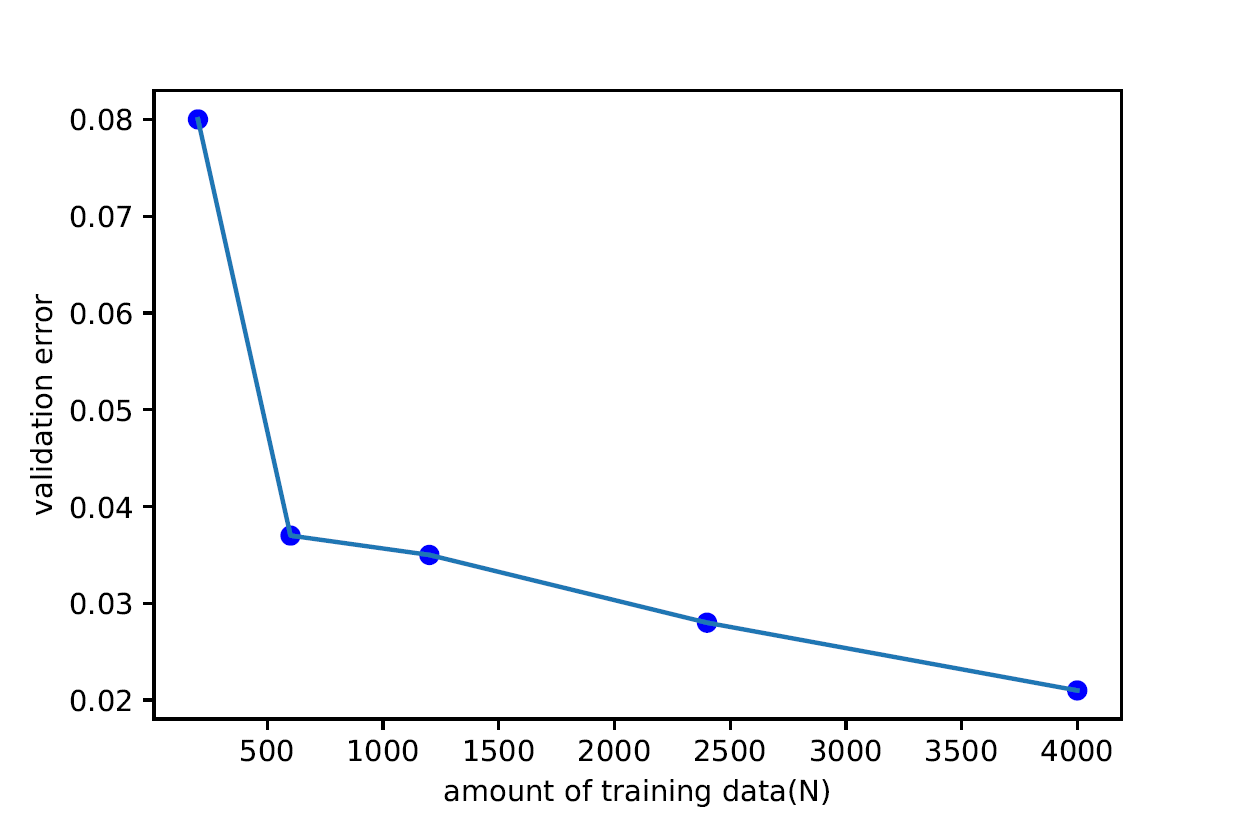
3. Train the linear classifier using your training set. How many mistakes are made before the algorithm terminates? Test your implementation of perceptron\_error by running it with the learned parameters and the training data, making sure that the training error is zero. Next, classify the emails in your validation set. What is the validation error?



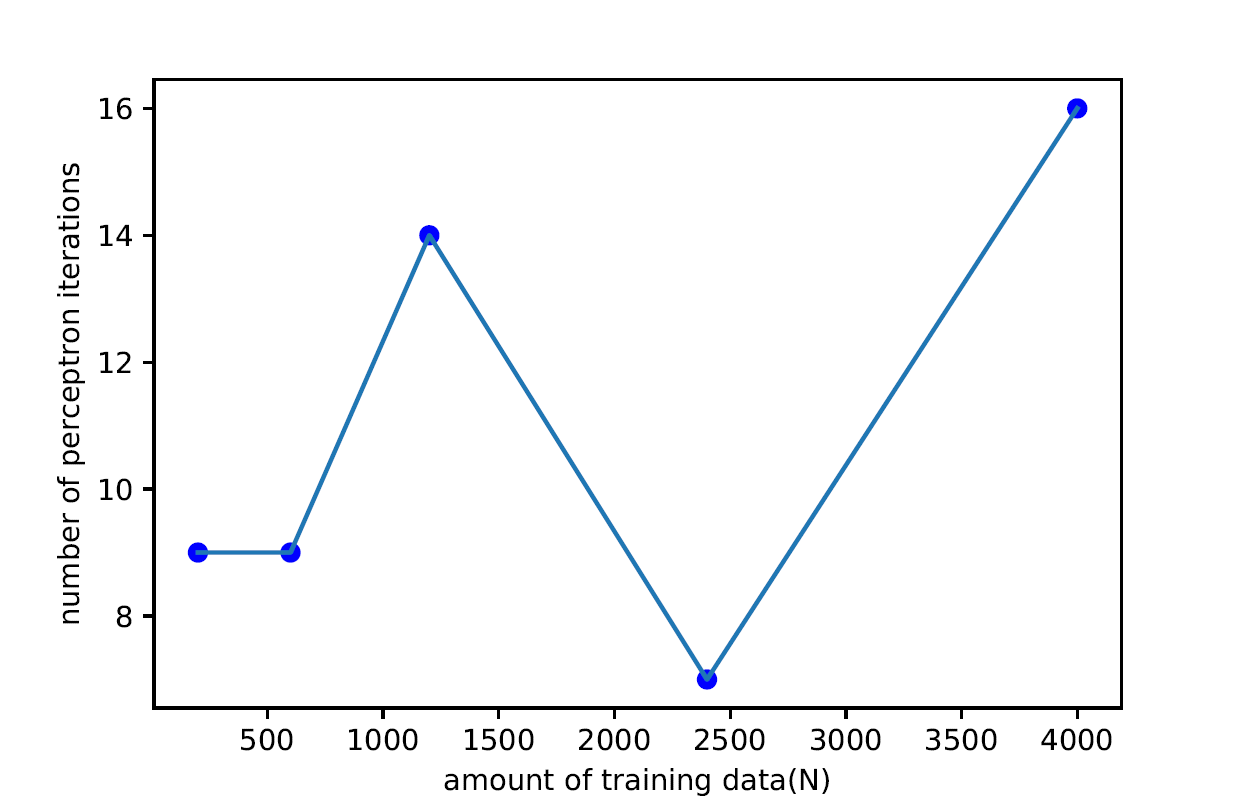
4. To better understand how the spam classifier works, we can inspect the parameters to see which words the classifier thinks are the most predictive of spam. Using the vocabulary list together with the parameters learned in the previous question, output the 12 words with the most positive weights. What are they? Which 12 words have the most negative weights?



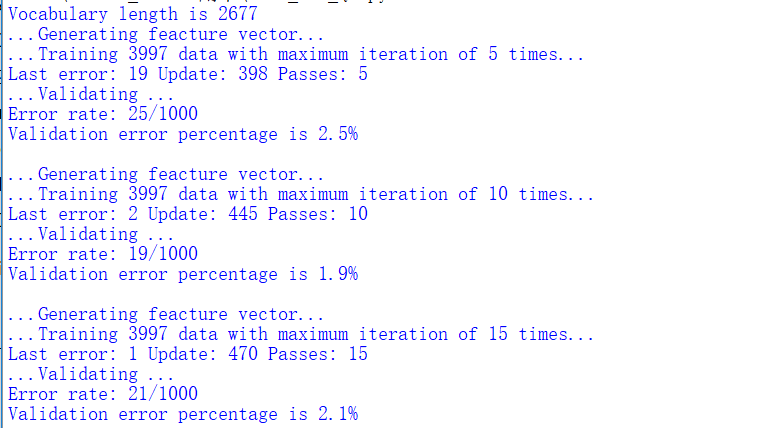
5. One should expect that the test error decreases as the amount of training data increases. Using only the first N rows of your training data, run both the perceptron algorithm on this smaller training set and evaluate the corresponding validation error (using all of the validation data). Do this for N = 200,600, 1200, 2400, ~~4000~~3997, and create a plot of the validation error as a function of N.



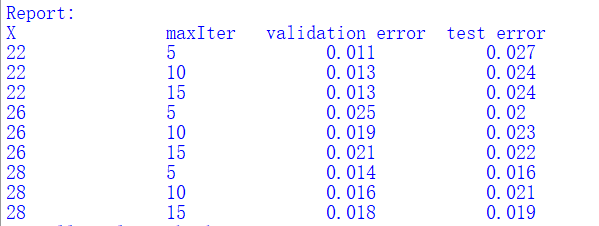
6. Also for N = 200, 600, 1200, 2400, ~~4000~~3997, create a plot of the number of perceptron iterations as a function of N, where by iteration we mean a complete pass through the training data. As the amount of training data increases, the margin of the training set decreases, which generally leads to an increase in the number of iterations perceptron takes to converge (although it need not be monotonic).



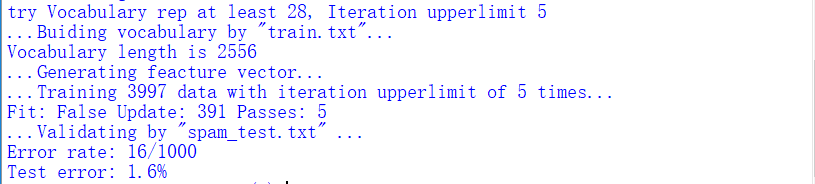
7. One consequence of this is that the later iterations typically perform updates on only a small subset of the data points, which can contribute to over-fitting. A way to solve this is to control the maximum number of iterations of the perceptron algorithm. Add an argument to the perceptron algorithm 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.)



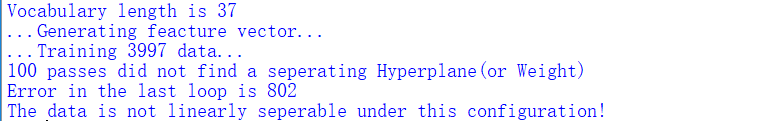
8. Congratulations, you now understand various properties of the perceptron algorithm. Try various configurations of the algorithm on your own using all 4000 training points, and find a good configuration having a low error on your validation set. In particular, try changing the maximum number of iterations 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 linearly separable?



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

**Training set** is where the machine learning algorithm always learn from, until it finds a fit model. (Error should be 0 or very small.)

**Validation set** is for evaluating the performance of a model. Checking error rate on this set, we go back to the training set, change the configuration of algorithm and find a new model. This process continues until performance on validation set is optimized (or validation error is minimized).

**Test set** is for estimating the final error rate of the machine learning model. It should never be peeked at, never optimized on.