Assignment 1: Perceptron Algorithm

Abdullah Zameek (arz268)

Q1. 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.

Q2. 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

These questions have been answered with the code itself. The functions were implemented as per the specification. Other math and helper functions were also introduced in order to reduce the load within the core functions, as well as to make the

program easier to read. Efforts were taken to ensure the code was as user-friendly and readable as possible.

Q3. 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 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?

```
Done, a value of w has been found!
Computing feature vectors..
Features done, labels done
The error rate on the training set is  0.0
The total number of mistakes made is:   470
The total number of passes made through the data is:  16
Computing feature vectors..
Features done, labels done
The error rate on the validation set is  2.3
```

The algorithm makes a total of 470 mistakes in 16 passes.
The error rate for the training set was 0% (as expected), and the validation set reported an error rate of 2.3%

The parameters for this run were as follows:
Training size: 3997
Validation Set Size: 1000
X: 26

Q4. 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?

The algorithm written to find the 12 most positive and negative weight words returned much more than 12 words since multiple words would have had the same weight. Only the first twelve words have been listed below.

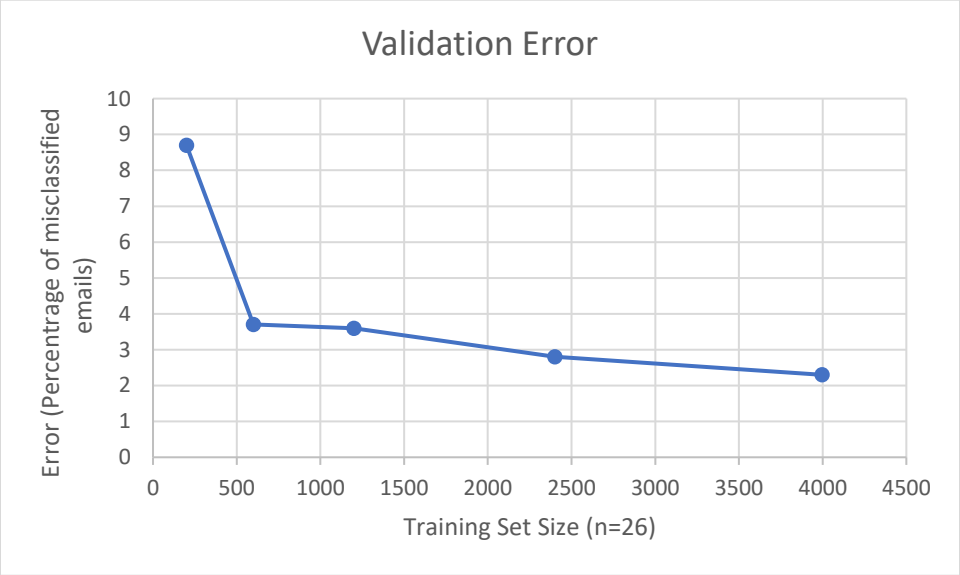
Twelve most **positive** weight words:

['remov', 'click', 'guarante', 'our', 'sight', 'most', 'market', 'busi','deathtospamdeathtospamdeathtospam', 'below', 'am', 'dollar']

Twelve most **negative** weight words:

['wrote', 'version', 'll', 'standard', 'review', 'announc', 'prefer', 'data', 're', 'i', 'view', 'the']

Q5. 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



N	Validation Error
200	8.7
600	3.7
1200	3.6
2400	2.8
3997	2.3

Q6. 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)

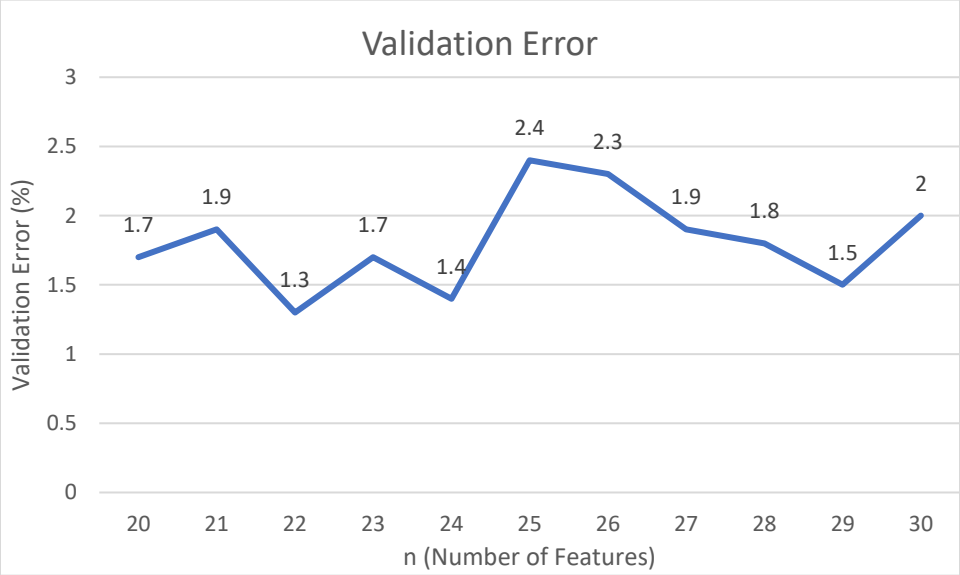
N	# of Loops
200	9
600	9
1200	14
2400	7
3997	16

Q7. 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.)

```
while not separated and iterator < iterLimit
```

Added an additional parameter to the Perceptron class init() and added check to the perceptron_train() as shown above

Q8. 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?



As seen from the above graph, the lowest error for the validation set is **1.3%** and it occurs when **n=22**. Changing the iteration limit at n=22 would often cause the program to exit prematurely and give an error rate > 0 for the training set. For the value that gives it a 0% error rate (n=8), the error rate reported was 2.5%.

n	Error
20	1.7
21	1.9
22	1.3
23	1.7
24	1.4
25	2.4
26	2.3
27	1.9
28	1.8
29	1.5
30	2

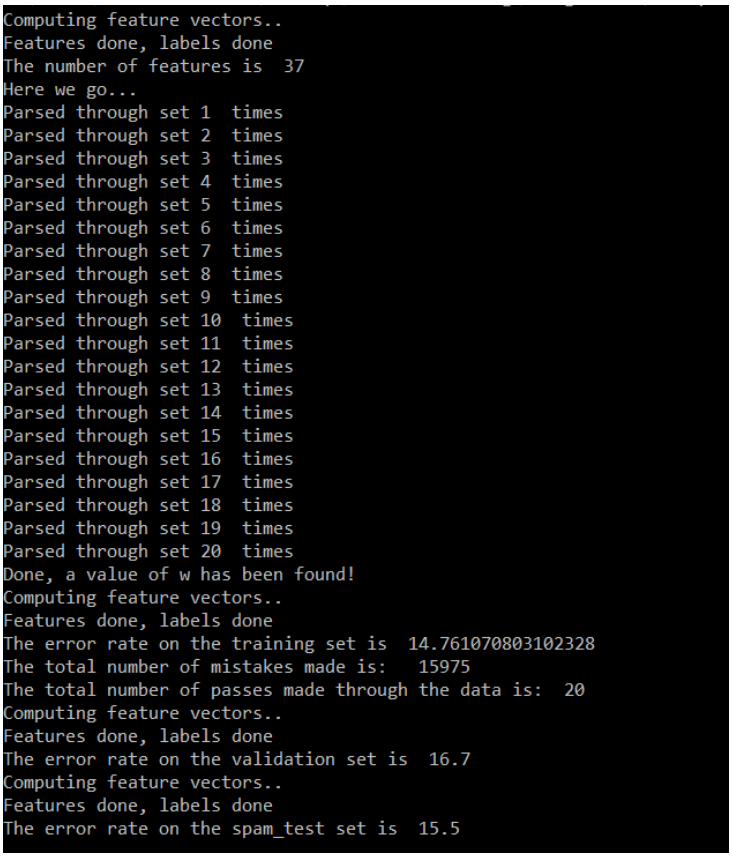
You are ready to train on the full training set, and see if it works on completely new data. Combine the training set and the validation set (i.e. use all of spam train.txt) and learn using the best of the configurations previously found. What is the error on the test set (i.e.,now you finally use spam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

The error rate for the spam test data set is **2.5%** for the given configuration :
Training size: 4977
X: 26

```
Computing feature vectors..
Features done, labels done
The number of features is  3048
Here we go...
Parsed through set 1  times
Parsed through set 2  times
Parsed through set 3  times
Parsed through set 4  times
Parsed through set 5  times
Parsed through set 6  times
Parsed through set 7  times
Parsed through set 8  times
Parsed through set 9  times
Parsed through set 10  times
Parsed through set 11  times
Parsed through set 12  times
Parsed through set 13  times
Parsed through set 14  times
Parsed through set 15  times
Parsed through set 16  times
Done, a value of w has been found!
Computing feature vectors..
Features done, labels done
The error rate on the training set is  0.0
The total number of mistakes made is:   536
The total number of passes made through the data is:  16
Computing feature vectors..
Features done, labels done
The error rate on the spam_test set is  2.5
```

Q9. Suppose we only consider words that appear in at least X-1500 emails. How many features are there? Is the data linearly separable?

This configuration reported a total of 37 features. The algorithm exited after 20 passes (the set iteration limit), returning an error rate of 14.76 on the training data itself. This implies that the algorithm was unable to find a value that linearly separated the data set. The linear inseparability was further confirmed upon increasing the iteration limit to larger values (limit > 100).
This also resulted in error rates > 0 for the training data, therefore the data is now linearly inseparable with a configuration of X = 1500.
The reason for this being so is because, when you are looking for features that appear in at least 1500 emails, very common words that appear in both spam and non-spam emails make its way into the features list. With such a feature set, it is difficult to accurately tell apart a spam email from a legitimate one.



Q10. Describe in words why we have we need a training set, validation set, and text set (three disjoint sets of emails)

The training set is used to calculate the weights of each feature, and hence get the vector w that is a linear separator to the data set. The validation set is used to ensure that the algorithm has not been over-optimized for the training set and hence, it is a

means of preventing overfitting. The test set is used to test for generalization – whether the algorithm can classify new data with a reasonably low error rate.