Machine Learning and Computational Statistics, Spring 2014

Problem Set 1: Perceptron Algorithm

Elizabeth Lamm

February 6, 2014

0.1

Python code to split data into a training and validate set, putting the last 1000 emails into the validation set:

```
import numpy as np
import matplotlib
from matplotlib import pyplot as plt

#   0. Read in data file
trainTotal = list()
trainingFile = open('spam_train.txt')
for line in trainingFile:
    trainTotal.append(line.replace('\n', '').split(' '))

#   1. Split overall training set into training set (subTrain) and validate set
splitPoint = 4000
numTrainTotal = len(trainTotal)
subTrainSet = trainTotal[0:splitPoint]
validateSet = trainTotal[splitPoint:numTrainTotal]
```

If I didn't create a validation set from the training data, I would not have any labeled data to evaluate the classifier performance that was not used in the development of the classifier. I would have problems setting the parameters, and would probably overfit the training data. The validation set lets me assess the generalization performance of my classifier.

0.2

Python functions to transform the data into feature vectors and build the vocabulary lists:

```
def createTotalVocab(datalist):
# This function takes "datalist" of emails and returns a list of all words in the emails
    datalen = len(datalist)
    vocabset = set()
    for i in range(0, datalen):
        thisEmailLen = len(datalist[i])
        for j in range(1, thisEmailLen):
            vocabset.add(datalist[i][j])
    vocablist = list(vocabset)
    return vocablist

def createNarrowedVocab(datalist, allVocab, minEmails):
# This function takes "datalist" of emails, "allVocab" list of all words used in
# datalist emails, and returns a list of words that are in at least minEmails
    datalen = len(datalist)
    vocablen = len(allVocab)
    vocabBOWarray = np.zeros((datalen, vocablen))
    thisEmailVocab = set()
    for i in range(0, datalen):
        thisEmailLen = len(datalist[i])
        for j in range(1, thisEmailLen):
            thisEmailVocab.add(datalist[i][j])
        for k in range(0, vocablen):
            if allVocab[k] in thisEmailVocab:
                vocabBOWarray[i][k] = 1
            else:
                next
        thisEmailVocab.clear()

    allVocabCount = vocabBOWarray.sum(axis = 0, keepdims=True)
    narrowedVocab = list()
    for k in range(0, vocablen):
        if allVocabCount[0][k] >= minEmails:
            narrowedVocab.append(allVocab[k])
        else:
            next

    return narrowedVocab

def featurize(datalist, narrowedVocab):
# This function takes "datalist" of emails, "narrowedVocab" list of words used in
# a minimum number of emails, and returns a featurized data array
    datalen = len(datalist)
    vocablen = len(narrowedVocab)
    vocabBOWarray = np.zeros((datalen, vocablen))
    thisEmailVocab = set()
    for i in range(0, datalen):
        thisEmailLen = len(datalist[i])
        for j in range(1, thisEmailLen):
            thisEmailVocab.add(datalist[i][j])
        for k in range(0, vocablen):
            if narrowedVocab[k] in thisEmailVocab:
                vocabBOWarray[i][k] = 1
            else:
                next
        thisEmailVocab.clear()

    return vocabBOWarray

def labelize(datalist):
#   This function takes "datalist" of emails and returns an array of the labels,
#   with 0 labels converted to -1

    datalen = len(datalist)
    labels = np.zeros((datalen,), dtype=int)

    for i in range(0, datalen):
        labels[i] = 2*int(datalist[i][0])-1

    return labels
```

This python code executes the functions above:

```
#   2. Transform all the data into feature vectors
#   Find all the words that occur in the training set
allVocab = createTotalVocab(subTrainSet)
minEmails = 30
narrowedVocab = createNarrowedVocab(subTrainSet, allVocab, minEmails)
STfeaturized = featurize(subTrainSet, narrowedVocab)
VSfeaturized = featurize(validateSet, narrowedVocab)
STlabels = labelize(subTrainSet)
VSlabels = labelize(validateSet)
```

0.3

Python code to implement the functions $perceptron\_train\,(data)$ and $perceptron\_test\,(w, data)$ - note that my "data" is passed to the functions as two arguments, 1. data and 2. labels:

```
#   3. Implement the functions perceptron_train(data) and perceptron_test(data)
def perceptron_train(data, labels, bias=0, stop=20):
#   This function takes feature vectors in array data and the data labels in vector labels
#   and trains the data using the perceptron algorithm.  It returns the trained weight vector,
#   number of mistakes, and number of passes through the data.
    numDataPoints = data.shape[0]
    numFeatures = data.shape[1]
    dataMatrix = data
    if bias == 1:
#   add bias to features
        dataMatrix = np.insert(data, 0, 1, axis = 1)

    w_vector = np.zeros((numFeatures + bias, ), dtype = float)
    predicted_class_vector = np.zeros((numDataPoints, ), dtype = int)
    k_mistakes = int(x=0)
    iter = int(x=0)

    for i in range(0, stop):
        iter += 1
        for j in range(0, numDataPoints):
            if np.vdot(dataMatrix[j,:], w_vector) >= 0:
                predicted_class_vector[j] = 1
            else:
                predicted_class_vector[j] = -1
            if predicted_class_vector[j]*labels[j] == -1:
                k_mistakes += 1
                w_vector = w_vector + labels[j]*dataMatrix[j]
        if np.amin(np.multiply(predicted_class_vector, labels)) < 0:
            next
        else:
            break

    return w_vector, k_mistakes, iter

def perceptron_test(w_vector, data, labels):
    error_count = int(x=0)
    numDataPoints = data.shape[0]
    predicted_class_vector = np.zeros((numDataPoints, ), dtype = int)
    check_errors= np.zeros((numDataPoints, ), dtype = int)

    for j in range(0, numDataPoints):
        if np.vdot(data[j,:], w_vector) >= 0:
            predicted_class_vector[j] = 1
        else:
            predicted_class_vector[j] = -1

    check_errors = np.multiply(predicted_class_vector, labels)
    for j in range(0, numDataPoints):
        if check_errors[j] < 0:
            error_count += 1
        else:
            next

    return float(error_count)/float(numDataPoints)
```

0.4

I discussed this problem with Cafer Yildirim.

Train the linear classifier using your training set:

```
#   4. Train the linear classifier using your training set/Test your implementation of
#   perceptron_test by running it with the learned parameters and the training data

w_vector, k_mistakes, iter = perceptron_train(STfeaturized, STlabels)
check_training_error = perceptron_test(w_vector, STfeaturized, STlabels)
test_error = perceptron_test(w_vector, VSfeaturized, VSlabels)
```

```
print k_mistakes
print iter
print check_training_error
print test_error
```

The python output was:
447
11
0.0
0.02
This means the classifier made 447 mistakes before the algorithm terminated. The algorithm made 11 passes through the data. The training error was confirmed to be 0.0, and the validation error was 0.02.

0.5

I discussed this problem with Cafer Yildirim.

I wrote the function $find\_k\_largest\_weights\,(w_vector, k, narrowedVocab)$ to find the words in the vocabulary list with the k most positive and the k most negative weights:

```
#   5. Find the 15 words with the most positive weights and the 15 words with the most
#   negative weights
def find_k_largest_weights(w_vector, k, narrowedVocab):
    num_weights = w_vector.shape[0]
    words = list()

    sort_array = np.sort(np.copy(w_vector))
    largest_weights = np.zeros((2*k, ), dtype = float)
    largest_weights[0:k] = sort_array[0:k]
    largest_weights[k:2*k] = sort_array[num_weights-k:num_weights]

    for i in range(0, num_weights):
        if w_vector[i] in largest_weights:
            words.append([w_vector[i], narrowedVocab[i]])
        else:
            next

    return sorted(words, key = lambda words:words[0])
```

This code calls the function with k = 15:

```
lgwg_words = find_k_largest_weights(w_vector, 15, narrowedVocab)
for i in range(0, len(lgwg_words)):
    print lgwg_words[i]
```

The python output was:

```
[-15.0, 'wrote']
[-15.0, 'but']
[-15.0, 'prefer']
[-14.0, 'and']
[-13.0, 'reserv']
[-13.0, 'i']
[-12.0, 'still']
[-12.0, 'technolog']
[-12.0, 'on']
[-11.0, 'copyright']
[-11.0, 'url']
[-11.0, 'recipi']
[-11.0, 'instead']
[-11.0, 'upgrad']
[-11.0, 'sinc']
[13.0, 'am']
[13.0, 'major']
[14.0, 'deathtospamdeathtospamdeathtospam']
[14.0, 'present']
[14.0, 'your']
[14.0, 'ever']
[15.0, 'click']
[15.0, 'nbsp']
[15.0, 'these']
[15.0, 'pleas']
[15.0, 'market']
[15.0, 'guarante']
[16.0, 'remov']
[16.0, 'yourself']
[17.0, 'our']
[19.0, 'sight']
```

I observe that I actually got 16 words with the 15 most positive weights, because there are 2 words, 'am' and 'major', with equal size weight ranked 15.

0.6

Python code to implement the function *avg_perceptron_train* (*data*):

```
#   6. Implement the averaged perceptron algorithm
def avg_perceptron_train(data, labels, bias=0, stop=20):
#   This function takes feature vectors in array data and the data labels in vector labels
#   and trains the data using the AVERAGED perceptron algorithm.  It returns the trained weight vector,
#   number of mistakes, and number of passes through the data.
    numDataPoints = data.shape[0]
    numFeatures = data.shape[1]
    dataMatrix = data
    if bias == 1:
#   add bias to features
        dataMatrix = np.insert(data, 0, 1, axis = 1)

    w_vector = np.zeros((numFeatures + bias, ), dtype = float)
    sum_w_vectors = np.zeros((numFeatures + bias, ), dtype = float)
    num_w_vectors = int(x=0)
    predicted_class_vector = np.zeros((numDataPoints, ), dtype = int)

    k_mistakes = int(x=0)
    iter = int(x=0)

    for i in range(0, stop):
        iter += 1
        for j in range(0, numDataPoints):
            sum_w_vectors = sum_w_vectors + w_vector
            num_w_vectors += 1
            if np.vdot(dataMatrix[j,:], w_vector) >= 0:
                predicted_class_vector[j] = 1
            else:
                predicted_class_vector[j] = -1
            if predicted_class_vector[j]*labels[j] == -1:
                k_mistakes += 1
                w_vector = w_vector + labels[j]*dataMatrix[j]
        if np.amin(np.multiply(predicted_class_vector, labels)) < 0:
            next
        else:
            break

    return sum_w_vectors/float(num_w_vectors), k_mistakes, iter

avg_w_vector, avg_k_mistakes, avg_iter = avg_perceptron_train(STfeaturized, STlabels)
check_aptrain_error = perceptron_test(avg_w_vector, STfeaturized, STlabels)
aptest_error = perceptron_test(avg_w_vector, VSfeaturized, VSlabels)
print check_aptrain_error
print aptest_error
```

The python output was:
0.00075
0.018
I notice that my validation error has decreased slightly from the regular perceptron (from 0.02 to 0.018), however I now have 4000*0.00075 = 3 data

points in my subtraining set that the average weight vector would misclassify. This is okay - it means the averaged perceptron algorithm does not overfit the subtraining set as much as the perceptron algorithm for this data set.

0.7

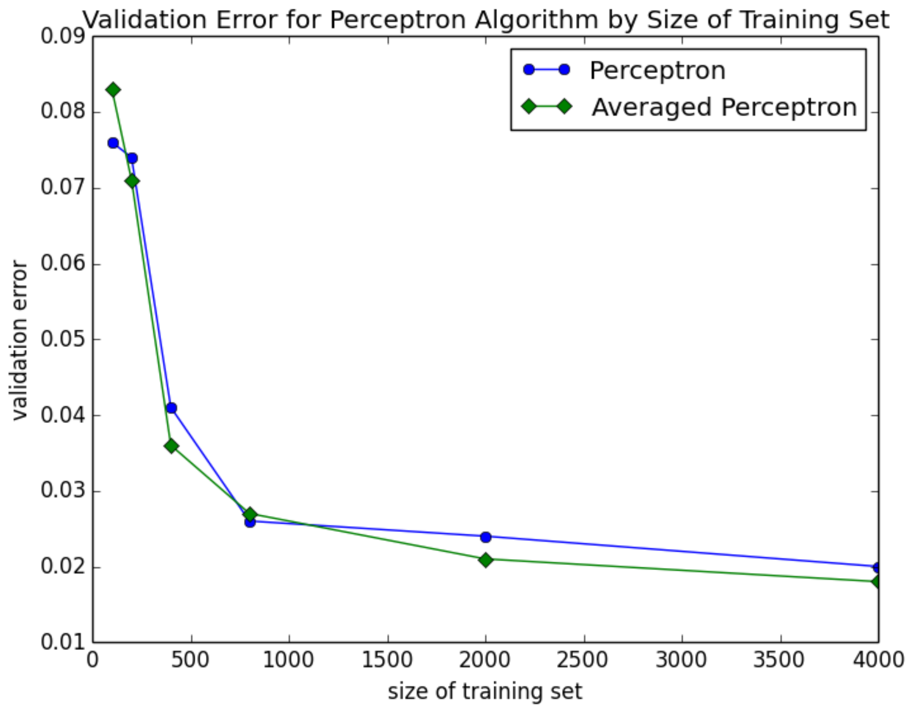
I discussed this problem with Cafer Yildirim.

```
#   7. Run the perceptron algorithm and the averaged perceptron algorithm on smaller training sets
#   Evaluate the corresponding validation error and create a plot of validation errors as a function of N
headers = ['training size', 'validation err', 'valid err - avg', 'training err', 'train err - avg', 'iters', 'iters - a
training_sizes = [100, 200, 400, 800, 2000, 4000]
outputs = np.zeros((len(headers), len(training_sizes)))
column = int(x=0)
for N in training_sizes:
    subTrainSet = trainTotal[0:N]
    STfeaturized = featurize(subTrainSet, narrowedVocab)
    STlabels = labelize(subTrainSet)

    w_vector, k_mistakes, iter = perceptron_train(STfeaturized, STlabels)

    avg_w_vector, avg_k_mistakes, avg_iter = avg_perceptron_train(STfeaturized, STlabels)

    outputs[0][column] = training_sizes[column]
    outputs[1][column] = perceptron_test(w_vector, VSfeaturized, VSlabels)
    outputs[2][column] = perceptron_test(avg_w_vector, VSfeaturized, VSlabels)
    outputs[3][column] = perceptron_test(w_vector, STfeaturized, STlabels)
    outputs[4][column] = perceptron_test(avg_w_vector, STfeaturized, STlabels)
    outputs[5][column] = iter
    outputs[6][column] = avg_iter
    outputs[7][column] = k_mistakes
    outputs[8][column] = avg_k_mistakes
    column += 1

for i in range(0, len(headers)):
    print headers[i], outputs[i, :]

fig, ax = plt.subplots()
ax.plot(training_sizes, outputs[1], 'bo-', label = 'Perceptron')
ax.plot(training_sizes, outputs[2], 'gD-', label = 'Averaged Perceptron')
ax.legend(loc='upper right')
plt.xlabel('size of training set')
plt.ylabel('validation error')
plt.title('Validation Error for Perceptron Algorithm by Size of Training Set')
plt.show()
```

Python output:

```
training size  [  100.   200.   400.   800.  2000.  4000.]
validation err  [ 0.076  0.074  0.041  0.026  0.024  0.02 ]
valid err - avg  [ 0.083  0.071  0.036  0.027  0.021  0.018]
```

```
training err  [ 0.  0.  0.  0.  0.  0.]
train err - avg  [ 0.   0.01   0.   0.00125  0.0005   0.00075]
iters  [  6.   3.   6.   5.  11.  11.]
iters - avg  [  6.   3.   6.   5.  11.  11.]
mistakes  [  32.   39.   84.  111.  259.  447.]
mistakes - avg  [  32.   39.   84.  111.  259.  447.]
```

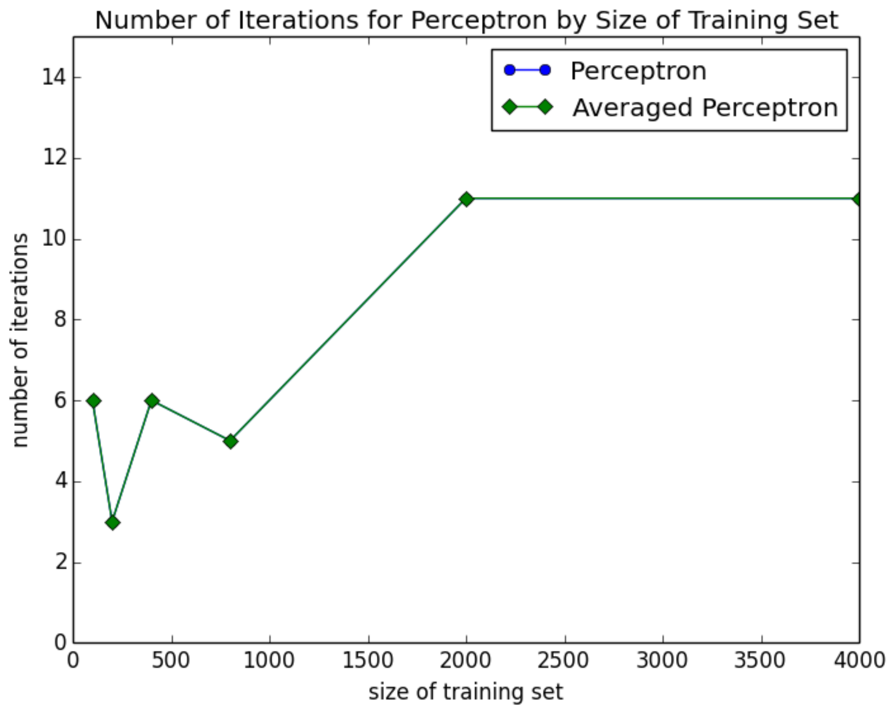


0.8

I discussed this problem with Cafer Yildirim.

```
#   8. Also create a plot of the number of iterations as a function of N
fig, ax = plt.subplots()
ax.set_xlim(0, 4000)
ax.set_ylim(0, 15)
ax.plot(training_sizes, outputs[5], 'bo-', label = 'Perceptron')
ax.plot(training_sizes, outputs[6], 'gD-', label = 'Averaged Perceptron')
ax.legend(loc='upper right')
plt.xlabel('size of training set')
plt.ylabel('number of iterations')
plt.title('Number of Iterations for Perceptron by Size of Training Set')
plt.show()
```

Both the perceptron algorithm and the averaged perceptron algorithm have the same number of iterations, so in this plot you see only one because they are on top of eachother.



0.9

Please see the last parameter in part 3 *perceptron_train* (*data*) and *perceptron_test* (*w*, *data*), the parameter "stop" defaults to 20 unless otherwise specified. It is used in the same manner for the function *avg_perceptron_train* (*data*) in part 6.

```
#   9. Functions modified above to add 4th parameter "stop"
```

0.10

I discussed this problem with Cafer Yildirim.

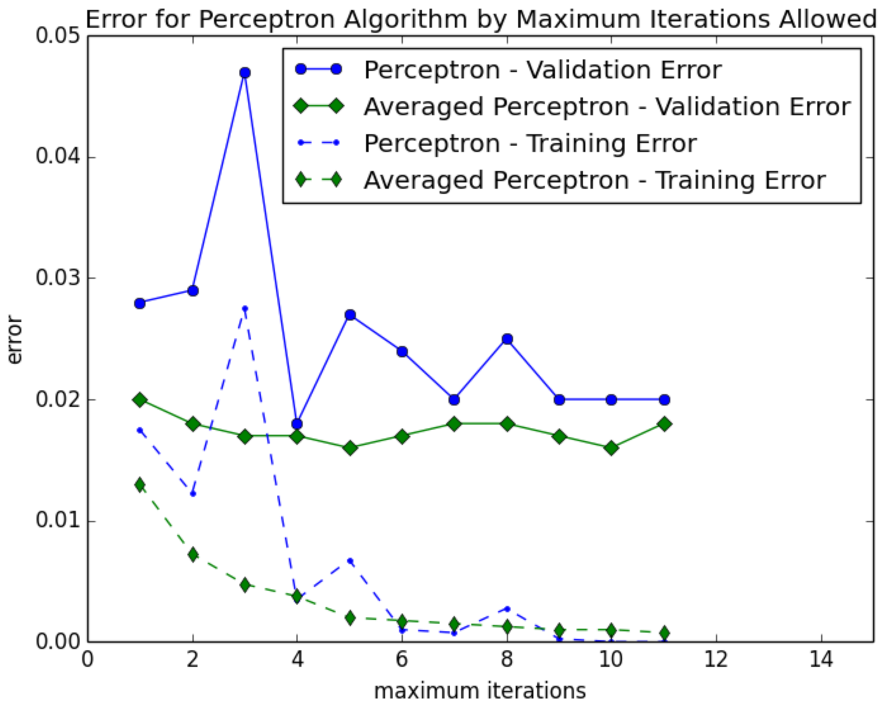
I tried the following configurations:

```
#   10. Try various configurations (reg/avg algorithm, maximum number of iterations) on your own
#   Train on the full training set
headers = ['max iters', 'validation err', 'valid err - avg', 'training err', 'train err - avg', 'iters', 'iters - avg'
outputs = np.zeros((len(headers), 11))
column = int(x=0)
for numiters in range(1, 12):
    subTrainSet = trainTotal[0:4000]
    STfeaturized = featurize(subTrainSet, narrowedVocab)
    STlabels = labelize(subTrainSet)
    w_vector, k_mistakes, iter = perceptron_train(STfeaturized, STlabels, stop = numiters)
    avg_w_vector, avg_k_mistakes, avg_iter = avg_perceptron_train(STfeaturized, STlabels, stop = numiters)

    outputs[0][column] = numiters
    outputs[1][column] = perceptron_test(w_vector, VSfeaturized, VSlabels)
    outputs[2][column] = perceptron_test(avg_w_vector, VSfeaturized, VSlabels)
    outputs[3][column] = perceptron_test(w_vector, STfeaturized, STlabels)
    outputs[4][column] = perceptron_test(avg_w_vector, STfeaturized, STlabels)
    outputs[5][column] = iter
    outputs[6][column] = avg_iter
    outputs[7][column] = k_mistakes
    outputs[8][column] = avg_k_mistakes
    column += 1

for i in range(0, len(headers)):
    print headers[i], outputs[i, :]

fig, ax = plt.subplots()
ax.set_xlim(0, 15)
ax.set_ylim(0, .05)
ax.plot(outputs[0], outputs[1], 'bo-', label = 'Perceptron - Validation Error')
ax.plot(outputs[0], outputs[2], 'gD-', label = 'Averaged Perceptron - Validation Error')
ax.plot(outputs[0], outputs[3], 'b.--', label = 'Perceptron - Training Error')
ax.plot(outputs[0], outputs[4], 'gd--', label = 'Averaged Perceptron - Training Error')
ax.legend(loc='upper right')
plt.xlabel('maximum iterations')
plt.ylabel('error')
plt.title('Error for Perceptron Algorithm by Maximum Iterations Allowed')
plt.show()
```

Python Output:

```
max iters  [  1.   2.   3.   4.   5.   6.   7.   8.   9.  10.  11.]
validation err  [ 0.028  0.029  0.047  0.018  0.027  0.024  0.02   0.025  0.02   0.02
valid err - avg  [ 0.02   0.018  0.017  0.017  0.016  0.017  0.018  0.018  0.017  0.016
  0.018]
training err  [ 0.0175   0.01225  0.0275   0.0035   0.00675  0.001    0.00075  0.00275
```

```
  0.00025  0.       0.     ]
train err - avg  [ 0.013    0.00725  0.00475  0.00375  0.002    0.00175  0.0015   0.0012
  0.001    0.001    0.00075]
iters  [  1.   2.   3.   4.   5.   6.   7.   8.   9.  10.  11.]
iters - avg  [  1.   2.   3.   4.   5.   6.   7.   8.   9.  10.  11.]
mistakes  [ 237.  319.  357.  393.  408.  413.  418.  431.  444.  447.  447.]
mistakes - avg  [ 237.  319.  357.  393.  408.  413.  418.  431.  444.  447.  447.]
```



In reviewing my output, I see that the validation error for the averaged perceptron algorithm is always lower than the validation error for the non-averaged perceptron algorithm. Therefore, the best configuration will use the averaged perceptron. The smallest validation errors for the averaged perceptron is 0.016. This validation error occured with maximum iterations of 5 and 10. Therefore, I choose to limit the maximum iterations to 5 in the best configuration (additional iterations did not achieve smaller error on the validation set).

Now I will train this best configuration (Averaged perceptron, max iterations = 5) on the full training set, and find the error on the test set, with the following python code:

```
trainSet = trainTotal
```

```
TSfeaturized = featurize(trainSet, narrowedVocab)
TSlabels = labelize(trainSet)
w_vector, k_mistakes, iter = avg_perceptron_train(TSfeaturized, TSlabels, stop = 5)
trainError = perceptron_test(w_vector, TSfeaturized, TSlabels)

testTotal = list()
testFile = open('spam_test.txt')
for line in testFile:
    testTotal.append(line.replace('\n', '').split(' '))
testFeaturized = featurize(testTotal, narrowedVocab)
testLabels = labelize(testTotal)
testError = perceptron_test(w_vector, testFeaturized, testLabels)

print trainError
print testError
```

Python output the following:
0.0026
0.019

My error on the test set was 0.019.