Rebecca Corcillo
Project 1 .February 20, 2014

1. Using the offset slightly increases training error. In the beginning, the algorithm ran at around 68% accuracy, with offset it ran at around 66% accuracy.

2. On the tweet data, the one cycle of the perceptron algorithm got 85% accuracy while one cycle of the passive aggressive algorithm got 87% accuracy. To get 100% accuracy, the perceptron algorithm took 12 cycles through the data, while the passive-aggressive algorithm took 5 cycles. I made it so the algorithms would continue to cycle through the data, updated its classifier, until it would have around 100% accuracy on the training data classification.

3. The data I used was the

```
##feature_matrix= np.array([[3,3],[3,4],[3,2],[3,5],[2,1],[-1,0],
[1,1],[-1,1],[-1,-1],[4,1],[5,1],[4,1],[1,-2],[4,0],[3,0],[3,-1]])
##labels=([1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1])
feature_matrix= np.array([[0,5],[0,4],[0,6],[1,5],[1,7],[-1,5],
[-1,6],[-1,7],[-1,-1],[-4,1],[-5,3],[-4,-3],[-1,-2],[-3,-4],[-2,-8],[0,-5]])
labels=([1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1])
```

   In the plots labeled lots, we see that the averager completely missclassifies one point, with another being right on the border. This happens due to clusters of data on either side of the classifier on the other side of the plot. Averager is thrown off by this data, because it is weighed as more and given higher precedence by the algorithm. Both the perceptron and passive aggressive algorithms properly classify all points, with slight variations in angle. The passive aggressive result has a larger overall average margin between the classifier and the points because it takes this into account by looking at step size when it updates theta. Similar things are seen in the longer plots. The averager algorithm misclassifies two points based on the blue outliers aligned in the vertical direction, and the red cluster aligned in the horizontal direction. The longer data perceptron and passive aggressive algorithms properly classify all data, again with small differences in angle from the passive-aggressive algorithm trying to optimize margin through step-size.

4. My cross-validation algorithm works by partitioning the training data into training and testing subsets. For every K iteration of the algorithm, we take a small fraction of the data out to use for testing, and leave the majority to train our classifier. We output a θ and θ_0 which we then use to generate labels on the testing set. We compare each tweet's test generated label with its actual label, adding 1 to the errors every time the test tweet was misclassified. We repeat this algorithm K times, changing which fraction we use for testing each time. In the end, we return the average errors for the algorithm,

by dividing the total number of errors by the number of testing tweets evaluated. By comparing average errors between cross-validation using different algorithms to obtain a classifier, like the averager, perceptron and passive aggressive, we are able to evaluate which algorithm should work best on new test data, based on which had the lowest error.

The resulting data actually did what was expected! The expectation was that passive-aggressive would be best, followed by perceptron, followed by averager. And for each algorithm, we expected a higher K to result in fewer errors, since the cross-validation algorithm would have more tweets to train on each time.
The average errors for K=5,10 and 15, respectively were: the averager was .4,.4,.3 , the perceptron algorithm was .2,.3,.13333, passive aggressive was .2,.5,.2 . While very accurate, the passive aggressive algorithm took the longest to run.

5. Computer learning is an extremely useful and powerful tool in computer science. It is used everywhere from seach suggestions, handwriting conversion to tumor classification. You can think of it the same way you think of learning: you see something with multiple features, you are told what it is, and you learn over time how to associate the combination of different features you've seen to a label. Being able to distinguish gender from faces is a very good example of this and can be used to explain how a computer would learn as well. When you see faces for the first time, you see features, eye seperation, eyebrow shape, facial hair. Your parents start to teach you which gender is which, and you learn to associate unique combinations of facial features to a gender. You build a sort of rule for doing so and update this rule every time you introduced to a person as a he or she. Maybe in the beginning your rule hasn't seen enough examples, so you overgeneralize and think that to be a boy you MUST have the exact facial features as your father. Your rule is not robust enough to generalize. Over time, you build up this rule to the point where you can always tell a person's gender just by looking at them without being told. This system is equivalent to computer learning. Computers do the same thing. I just did the same things with tweets, feeding the computer many tweets, each with an associated positive or negative label (for positive or negative movie reviews). The computer broke the tweet down and began to make a rule for how it associated combinations of parts to a positive or negative label. After looking at a lot of tweets and labels, it then had a robust rule that it was able to use to look at unseen tweets and determine wether or not they were positive or negative with very little error.

6. I chose to change the way the feature vectors for individual tweets were consturcted. Instead of having them contain only 0's or 1's at a specific index to signify if they do or don't have the corresponding dictionary word, I made it so they would instead contain the number of occurences of that word the tweet contains.

```
  feature_matrix[pos, dict.index(word)] = 1} to
\texttt { feature_matrix[pos, dict.index(word)] += 1}
```

I simply changed a line in extract feature vectors of project1 code from the feature matrix's feature vectors entry to + = 1 if if does contain a word in the dictionary. I figured this would produce a more accurate representation of the tweet and data, allowing the algorithm to more intelligently make a classifier based on training with actual occurences of words, since multiple occurences of a word add to the overall meaning of a tweet.
With only this difference, improvements were seen in the performance of the perceptron algorithm based on cross-validation. The results for k= 5,10 and 15 go from the previous .2,.3,.13333 to .2,.1,.2 , which is a lower overall average error.