Homework 1: September 7, 2016

Machine Learning

Susan Cherry

Question 1

Part A) When I begin with x_1, the algorithm makes two mistake (by mistake, I mean that the algorithm updates w twice from it's initialized value of $\mathbf{0}$). The output in this case is $w = [0.5, 2]$. When I start at x_2, the algorithm makes one mistake (only updates the weights once). My output in this case is $w = [0, 1]$.

See attached photos for the progress of my algorithm at both starting points, as well as diagrams with the decision boundary as the algorithm progresses.

Part B) Next, we change $x_3 = [10, 1]$. Now the algorithm makes 6 mistakes when I start from x_1 (the algorithm must update the weights six times). My output in this case is $w = [5, 6]$. When I start from x_2, the algorithm makes one mistake and the output is again $w = [0, 1]$. Again, see attached photos for the progress of my algorithm and decision boundaries.

Part C) Why are there differences in Part A and Part B? Different points affect the algorithm runs. We always initialize the beginning weights to $w_1 = (0, 0)$, so the algorithm always sees the first point as a mistake and updates the weights to $w_2 = (x_1)$ (or whatever our starting point is). This means that w_2 is highly dependent upon our starting point. Beginning with x_2 is a "better" starting point because the first weight update ($w_2 = (0, 1)$ in this case) linearly separates our data. Beginning with x_1, on the other hand, leads us to $w_2 = (-1, 1)$, which does not linearly separate the data, meaning the algorithm makes mistakes and we must try reweighing again.

Similarly, changing $x_3 = (1.5, 1)$ to $x_3 = (10, 1)$ affects our algorithm. Now when we update to w_3, we get a different weight value. This is because $w_3 = w_2 + y_3 * x_3$. When $x_3 = (1.5, 1)$, the resulting decision boundary correctly classifies all of the points, specifically point x_1 (refer to hand drawn diagrams for further details). When $x_3 = (10, 1)$, however, the decision boundary is rotated by $x_3 = (10, 1)$ (which is a more extreme value than the original x_3) above x_1 (meaning it is incorrectly classified), and the algorithm must continue to run to correct this. Because x_1 is misclassified, we continue rotating the decision

boundary by increments of (-1,1) until x_1 is correctly classified.

Part D) What is a procedure that would maximize the number of mistakes? We could start from the points that are mislabeled and pick the point that has the smallest margin closest to the current decision boundary. As we saw in Part 3, the current decision boundary is reweighed by the misclassified points. If we pick the point with the smallest margin to the current decision boundary, we will get the re-weight the new decision boundary by the smallest amount. Every time we correct the decision boundary, the correction will be small, leading to more mistakes before we finally get to the decision boundary that correctly classifies all of the points.

Question 2

First, I code my own implementation of the perceptron algorithm. See attached R file. After running the algorithm and testing it using the "mnist test" data set, I find that my algorithm gives 99.85816% accuracy.

Next, I try feature reduction (reducing the number of features, but keeping the same accuracy).

Remove all features with weight 0: First, I simply try removing all of the features that were assigned a weight of 0 by my perceptron algorithm. This reduces the number of features from 784 to 471 (a 40% reduction). I rerun the perceptron algorithm using only the remaining features. If find that the accuracy does not change (99.85816% accuracy), so removing the features with 0 weight was a good choice.

Remove features with low variance: Next, I try removing features with low variance. The reason for this is that in the extreme case when a feature has 0 variance (every observation has the same value for that feature), the feature will not be very helpful in learning about different groups of data/splitting the data into different groups. 167 features have 0 standard deviation, so I will definitely not include them. I also discard all features with standard deviation less than 55 (this is a somewhat arbitrary choice, but I just picked a threshold that greatly reduced my number of features). This left me with 191 features (a 75%) decrease. After running my perceptron algorithm on all remaining features and testing with the test data set, I get an accuracy of 99.76359%, which is very high for having discarded so many features.

I further reduce my number of features to just the 100 features with the largest standard deviation. Now my accuracy is 99.66903%, still very high. I also find that all of the 100 remaining features were among the 471 remaining features from my first method of only using features with a non-zero weight. This is a fairly large result: I was able to reduce the features to less than 15% of their

original number and still maintain nearly the same level of accuracy. This is because the features that were deleted contained little information that would be useful in training the algorithm to classify points correctly.