Project 1

WU1: Why is this computation equivalent to computing classification accuracy?):

- Classification accuracy is the percentage of correct predictions and to find it. Finding which values are the same across the two tables, assigning 1 for correctness, 0 for a false classification, we can fin the mean of the 0s and 1s for correctness to easily compute the average number for the correct guesses, or in other words, a percentage of correct predictions.

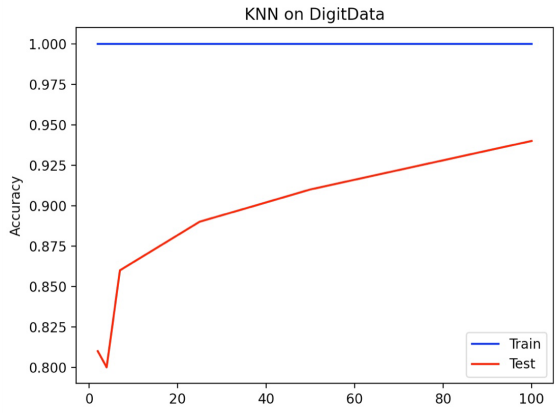
WU2: We should see training accuracy (roughly) going down and test accuracy (roughly) going up. Why does training accuracy tend to go *down?* Why is test accuracy not monotonically increasing? You should also see jaggedness in the test curve toward the left. Why?

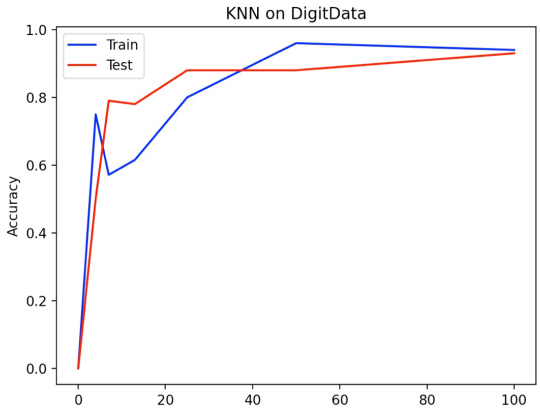
- As you start out with less data points, a classifier is able to memorize every sample and give a high training accuracy, but doesn't classify as properly in a test since it hasn't been trained on a larger variety of samples. Test accuracy doesn't monotonically increase because learning something based on a greater number of data points won't have a linear effect on testing accuracy due to the fact that learning from added data points may skew a prediction in one way or another as compared to when it had learned less or more. The jaggedness to the left is an explanation of that, where the small changes in classification from small data groups are drastically displayed with large, quick accuracy changes until about 100 data points.

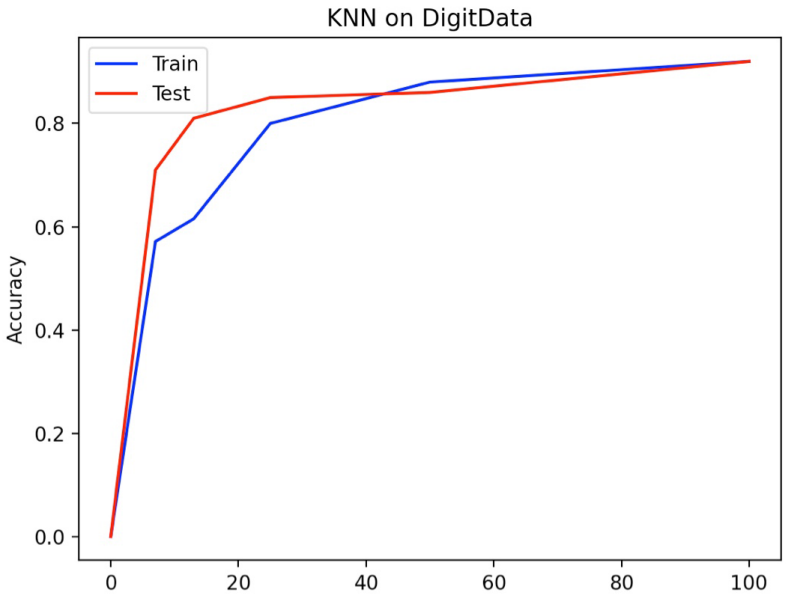
WU3: You should see training accuracy monotonically increasing and test accuracy making something like a hill. Which of these is *guaranteed* to happen and which is just something we might expect to happen? Why?

- The training accuracy increasing is something that is guaranteed where as the latter is something that can be expected. This is because as tree depth increases, it becomes more precise to fit the training set exactly, hence driving the training accuracy, but this doesn't always translate to a higher testing accuracy because as you start to memorize the training set, other values can't be as properly predicted, just matched with a training example, hence a hill as it gets better at learning it increases, and as it becomes too accurate to the training data, the test accuracy falls off, this is overfitting at that point.

WU4: For the digits data, generate train/test curves for varying values of K and epsilon (you figure out what are good ranges, this time). Include those curves: do you see evidence of overfitting and underfitting? Next, using K=5, generate learning curves for this data.





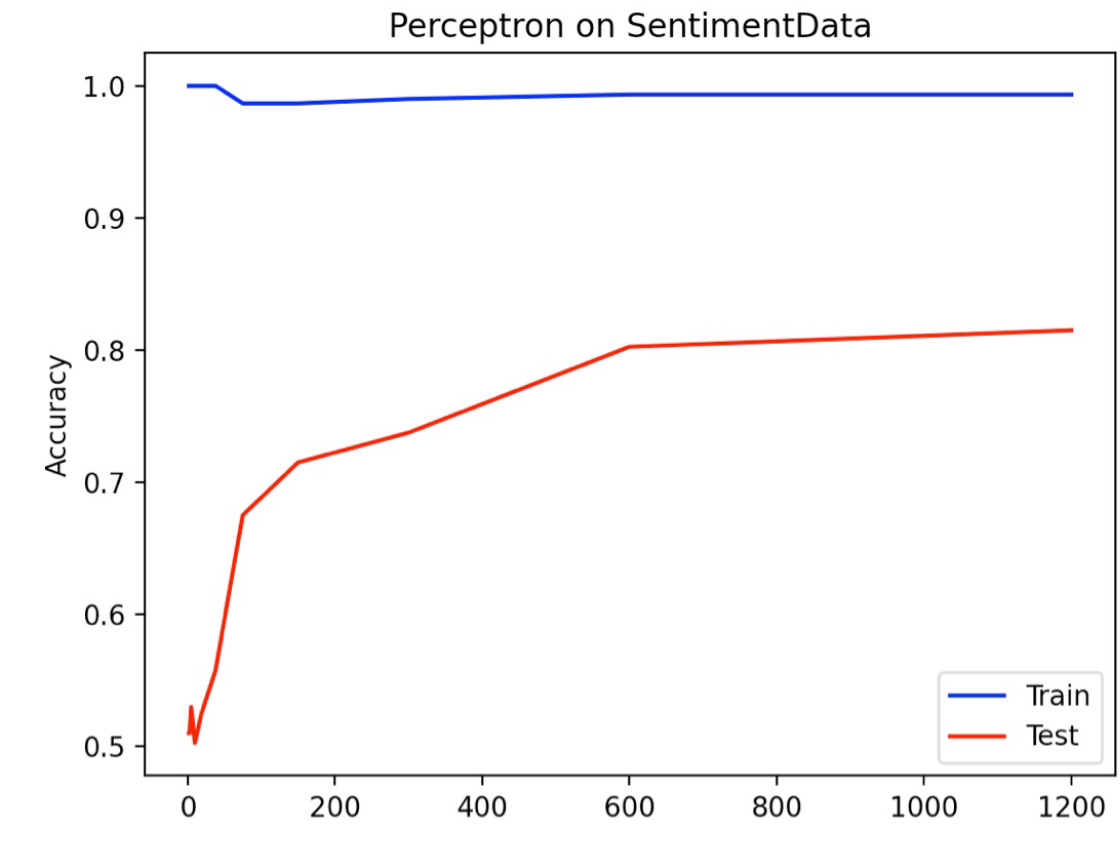


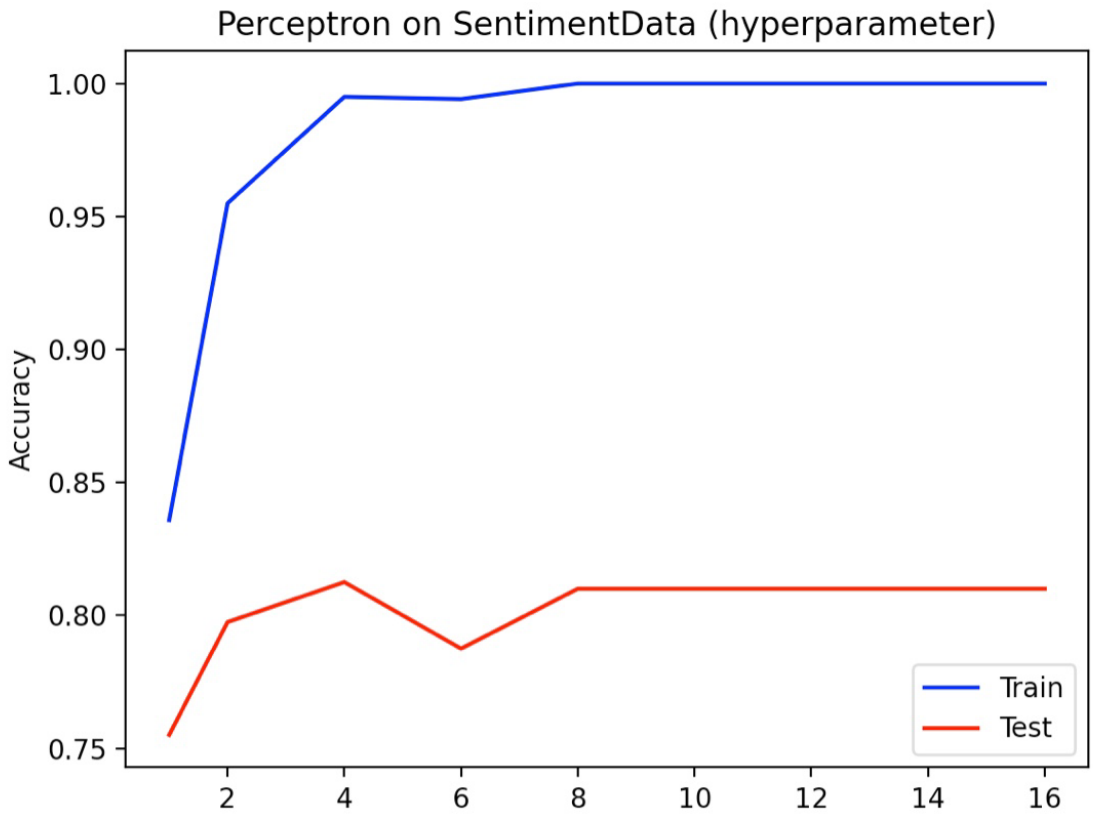
- I do not see evidence of over/underfitting

WU5: A. First, get a histogram of the raw digits data in 784 dimensions. You'll probably want to use the `computeDistances` function together with the plotting in HighD. **B.** Rewrite `computeDistances` so that it can subsample features down to some fixed dimensionality. For example, you might write `computeDistancesSubdims(data, d)`, where `d` is the target dimensionality. In this function, you should pick `d` dimensions at random (I would suggest generating a permutation of the number [1..784] and then taking the first d of them), and then compute the distance but *only* looking at those dimensions. **C.** Generate an equivalent plot to HighD with d in [2, 8, 32, 128, 512] but for the digits data rather than the random data. Include a copy of both plots and describe the differences.

-

WU6: Using the tools provided, generate (a) a learning curve (x-axis=number of training examples) for the perceptron (5 epochs) on the sentiment data and (b) a plot of number of epochs versus train/test accuracy on the entire dataset.




Perceptron on SentimentData (hyperparameter)
Accuracy
1.00
0.95
0.90
0.85
0.80
0.75
2
4
6
8
10
12
14
16
Train
Test