Vincent Zhao, Peter Yan

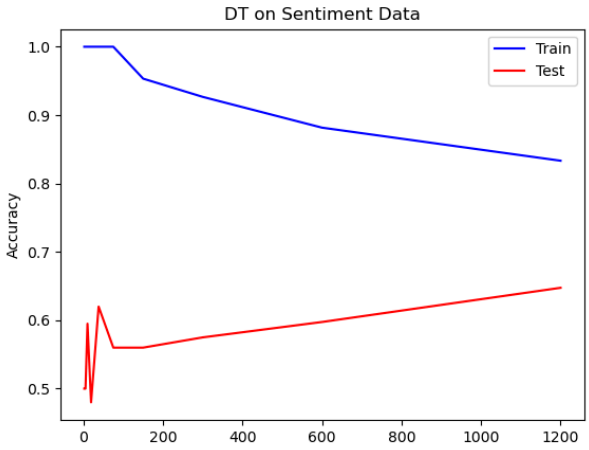
Question 1:
Why is ‘mean((datasets.TennisData.Y > 0) == (h.predictAll(datasets.TennisData.X) > 0))’ equivalent to computing classification accuracy?

The line compares the classifier h’s predictions on TennisData to the true labels. (datasets.TennisData.Y > 0) generates an array where all of the labels are converted to boolean values, with true for observations with a label of 1 (> 0) and false for observations with a label of -1 (is not > 0). This essentially converts the labels into a binary true false format. (h.predictAll(datasets.TennisData.X) > 0)) essentially does the same thing, but for the predicted labels of the dataset using the classifier h.
Combined, this allows us to compare the binary true/false results using the == comparison operator. The resulting array is true for elements where both the label and prediction are the same (either both true or both false). The end result is a boolean array where an element is true if the prediction matches the label, thus providing a way for us to determine classification accuracy.
We calculate the final proportion of correct predictions by taking the mean of this array where true values (correct predictions) correspond to +1 and false values (incorrect predictions) correspond to 0. Calculating the mean thus gives us the classification accuracy.

Question 2:
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?

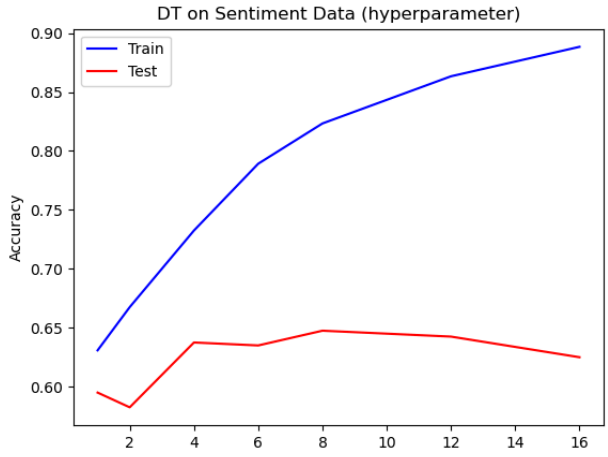
Training accuracy tends to decrease as the size of the training set increases because the model cannot accommodate for variability and noise with a large number of points. With smaller training sets, our decision tree tends to overfit the data, leading to a high training accuracy as the model learns the dataset perfectly. This similarly explains the low testing accuracy since the decision tree perfectly captures the training set and cannot generalize to testing sets. However, with more data points, the model starts to generalize and learn the overall patterns in the data and not just the dataset itself. This decreases training accuracy, but helps to increase testing accuracy thanks to better generalization.

The test accuracy does not increase monotonically because of variance in the data; the classifier may perform better or worse on different subsets of training data. With smaller dataset sizes, each point carries much more weight and any noise or outlier data can dramatically impact accuracy. This is evident in the left side of the graph as the test accuracy starts out extremely jagged. This issue is compounded by overfitting, which is commonplace with small dataset sizes. Overfitting captures noise in the training set which leads to high variability in testing and test accuracy. As a result, some overfitted models may be very accurate with the testing data while others are not. This leads to a non-monotonic increase in accuracy.



Question 3:

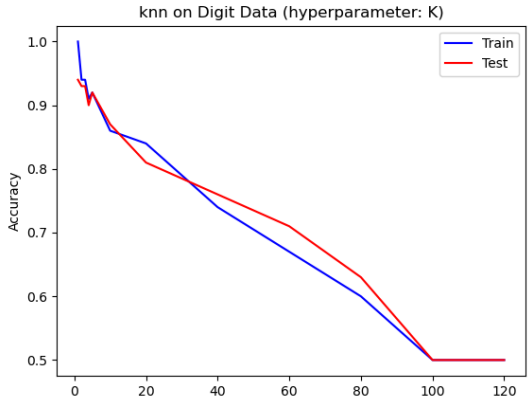
Training accuracy should monotonically increase. This is guaranteed because we are increasing the depth of the tree, increasing the accuracy and specificity with which we can classify our data. As we add more levels and increase the depth, we can better fit our data and better capture its patterns. On the other hand, we cannot guarantee that our testing accuracy will form a hill. This is because of variability in the test set which may or may not match the patterns learned from the training data. If the training data is not representative of the test data/entire dataset, or if the training data results in overfitting, we may not see any increases in testing accuracy reminiscent of a hill.



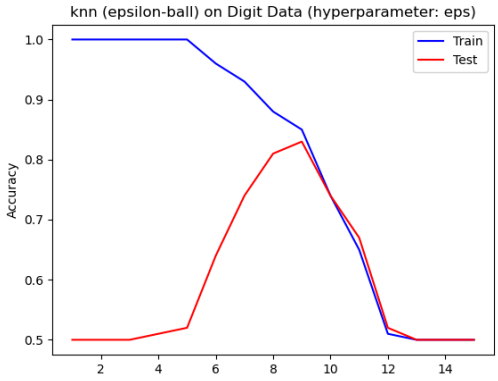
Question 4:

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.

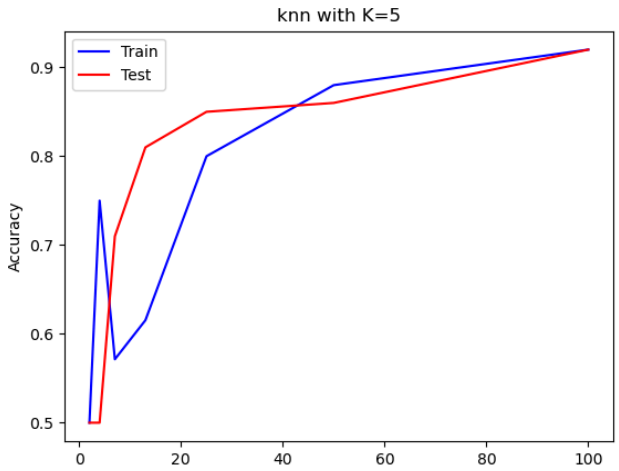
Yes, using a K value range from 1 to 120, I can see that there are some signs of over and underfitting for KNN. With a k-value of 1, the model is exhibiting *very* slight signs of overfitting. This can be seen by the fact that the training accuracy is 1.0 while the testing accuracy is lower. We have perfectly learned the training set by achieving 1.0 accuracy while our test accuracy achieves a lower value. Since the difference in accuracy is not very large and the test accuracy is still very good, this is only very slight overfitting (but overfitting nonetheless according to slide 30 in lecture 2). At the same time, at large values of K approaching 100, we see that our model begins to underfit as both the training and test accuracies converge at a low accuracy of 0.5. The model likely starts to consider all available points in the dataset and cannot capture the intricacies of the data and is no better than guessing.



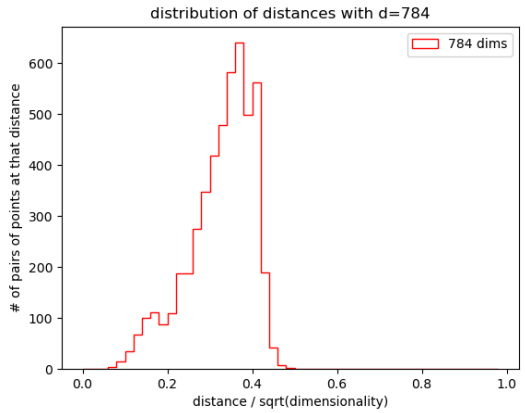
Similarly, using an epsilon ball model, we can clearly see overfitting when epsilon is less than 4 as the training accuracy is 1, indicating that the model is fitting the training perfectly, while the testing accuracy is around 0.5, which is the same as guessing. The model has learned the training data too well and cannot generalize when applied to test data. Around epsilon=8, we reach an optimal value since we have maximized test accuracy. However, at higher values of epsilon, we can see that the model is no better than guessing. This convergence of training and test accuracy at a low value is indicative of underfitting as the model starts to consider all available data points.

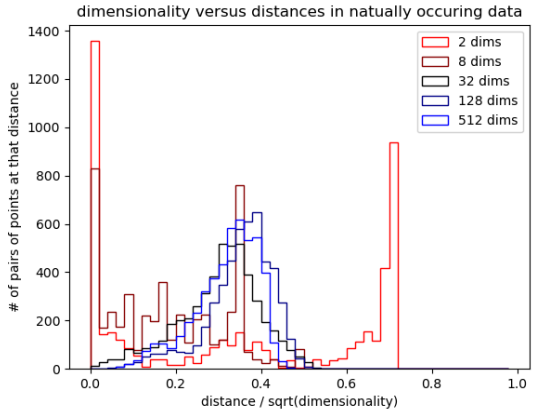


Learning curves with k=5



Question 5:



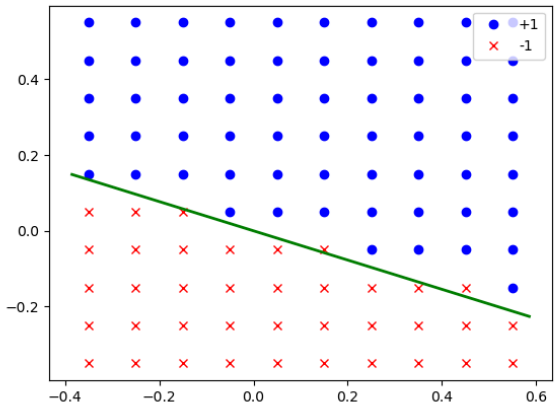


We see that when working with all 784 dimensions, the real dataset is unlike the uniformly randomly generated data points. In the normal implementation of HighD, we get a perfect uniform distribution. Here, we see a slight left skewed distribution with the number of pairs peaking around 0.4 but dropping once we get past 0.4 in our normalized distance measurements. Real life data follows specific patterns and correlations that can lead to this nonuniform distribution. KNN relies on this non-uniform distribution and the clustering of data points for its classification.

When we start to vary the dimensions that we select for measurement and analysis, we see at low dimensions high variability, but as we start to increase our dimensions, we can see a

shape/distribution similar to that of the 784-dimension plot. When d=2 or 8, the variance in distances is higher since we are only considering some dimensions, leading to information loss. We see strange distributions with high concentrations near low and high distances. At high dimensions, distances between points start to become similar and more information is incorporated, leading to a grouping of points thanks to the curse of dimensionality. This leads to a shape more reminiscent of the 784 dimension plot.

Question 6:



Graph generated from part of the directions.

Below are the graphs corresponding to the actual question: