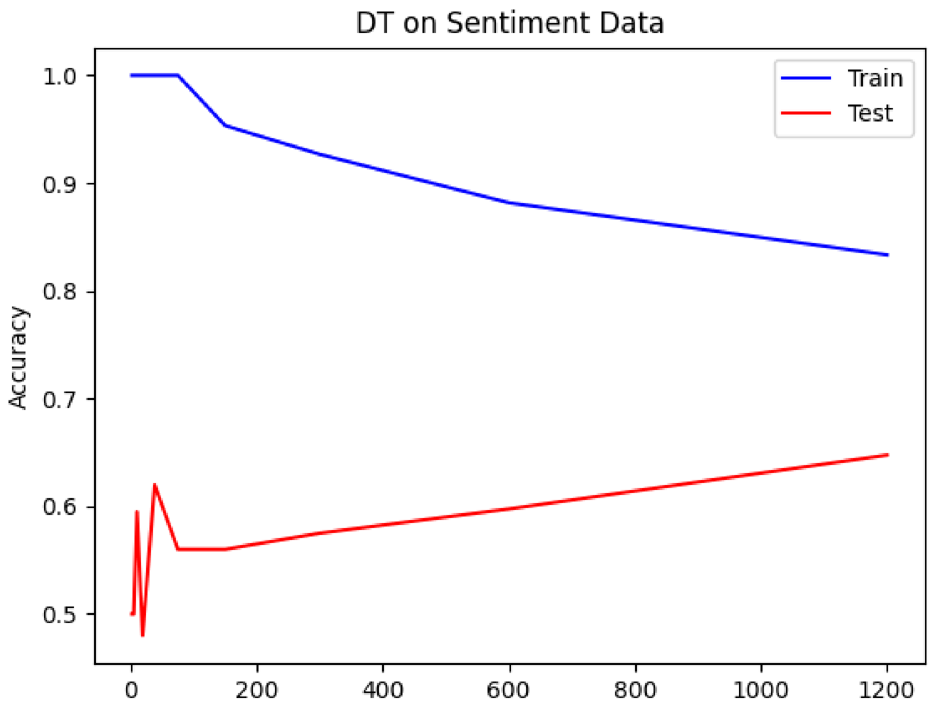
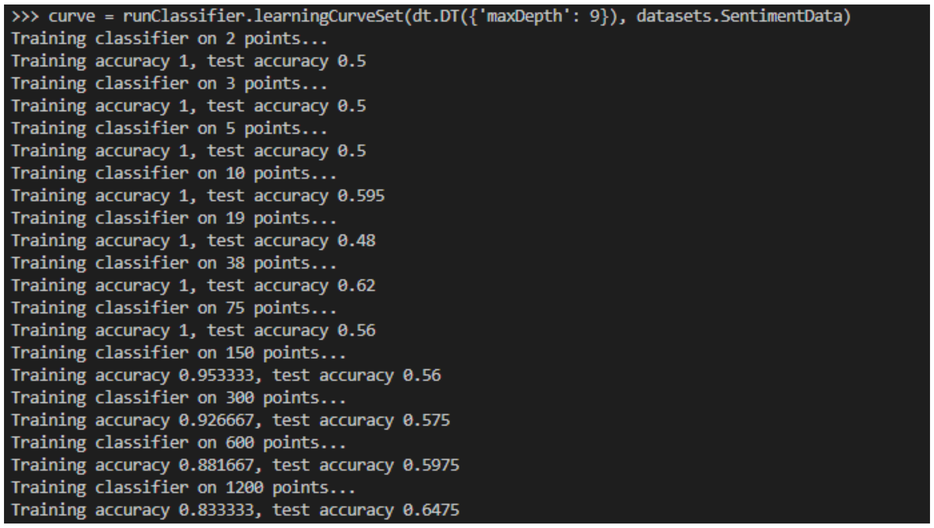
WU1:

This is the same as directly computing classification accuracy because the prediction of the classifier used here always returns one. This means the accuracy of this classifier is simply the proportion of 1’s in the Y vector of the dataset.

WU2:

We see a decrease in training accuracy and increase in testing accuracy as expected. The training accuracy tends to go down because it usually starts with very high (close to 100%) accuracy due to its nature of binary decisions. As more data points are included, the situation becomes more complex and more variations start to appear, which makes it harder for binary decisions to be as accurate as before. The test accuracy does not increase smoothly in a linear fashion because when the training dataset is not large enough, the predictions are based on narrow views of possible data, causing the classifier's decisions to be fluctuating more sharply. As the training dataset increases in size, the fluctuation in decisions will not be as severe, hence the testing accuracy will increase at a more steady and smooth rate.

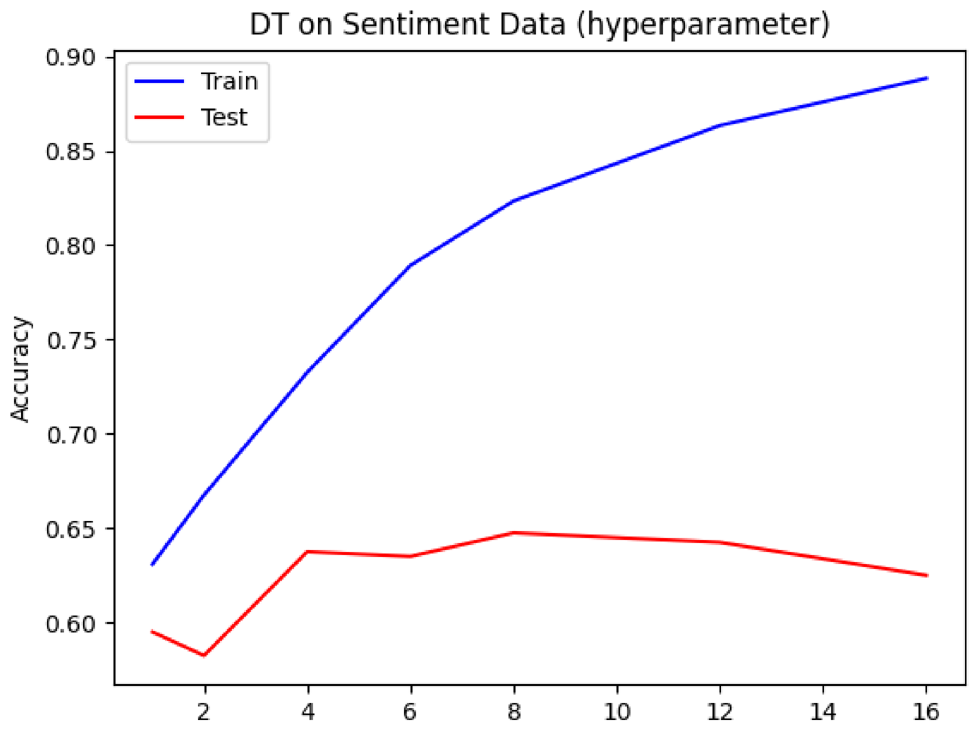


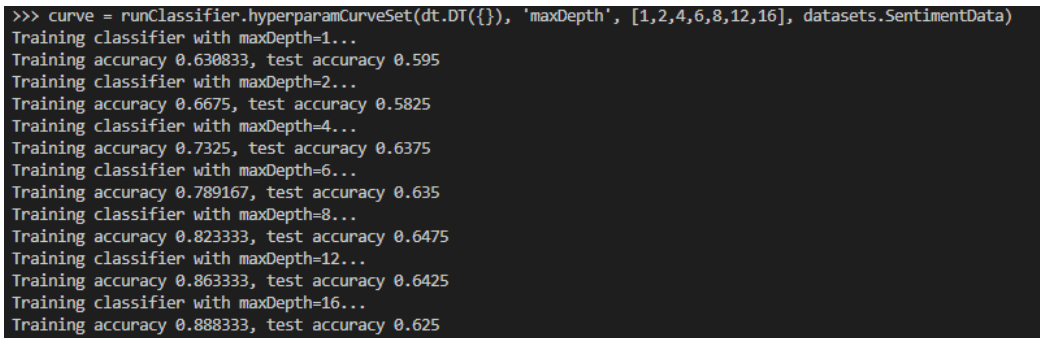


```
>>> curve = runClassifier.learningCurveSet(dt.DT({'maxDepth': 9}), datasets.SentimentData)
Training classifier on 2 points...
Training accuracy 1, test accuracy 0.5
Training classifier on 3 points...
Training accuracy 1, test accuracy 0.5
Training classifier on 5 points...
Training accuracy 1, test accuracy 0.5
Training classifier on 10 points...
Training accuracy 1, test accuracy 0.595
Training classifier on 19 points...
Training accuracy 1, test accuracy 0.48
Training classifier on 38 points...
Training accuracy 1, test accuracy 0.62
Training classifier on 75 points...
Training accuracy 1, test accuracy 0.56
Training classifier on 150 points...
Training accuracy 0.953333, test accuracy 0.56
Training classifier on 300 points...
Training accuracy 0.926667, test accuracy 0.575
Training classifier on 600 points...
Training accuracy 0.881667, test accuracy 0.5975
Training classifier on 1200 points...
Training accuracy 0.833333, test accuracy 0.6475
```

WU3:

The monotonic increase of training accuracy is guaranteed to happen as we increase the amount of features we use to determine the result (which is also the maximum depth x shown on the graph). On the other hand, we kind of expect the testing accuracy to increase as we increase the amount of features, but there is no guarantee for that to happen. This discrepancy is due to the fact that we only adjust the decision tree based on the training dataset, and we hope that the training dataset covers enough variety and distribution that can simulate the conditions of the testing dataset and real world data. Thus, we might see the test accuracy making something like a hill because the classifier might have overfitted to the training dataset.

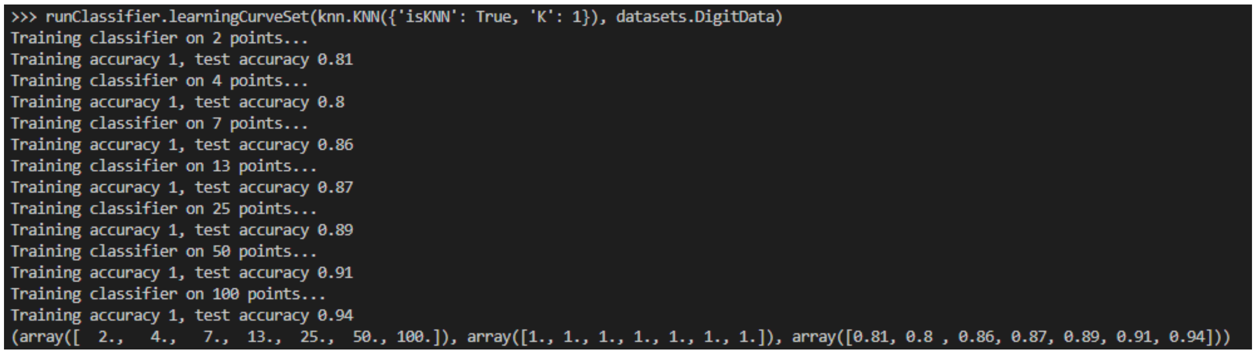




```
>>> curve = runClassifier.hyperparamCurveSet(dt.DT({}), 'maxDepth', [1,2,4,6,8,12,16], datasets.SentimentData)
Training classifier with maxDepth=1...
Training accuracy 0.630833, test accuracy 0.595
Training classifier with maxDepth=2...
Training accuracy 0.6675, test accuracy 0.5825
Training classifier with maxDepth=4...
Training accuracy 0.7325, test accuracy 0.6375
Training classifier with maxDepth=6...
Training accuracy 0.789167, test accuracy 0.635
Training classifier with maxDepth=8...
Training accuracy 0.823333, test accuracy 0.6475
Training classifier with maxDepth=12...
Training accuracy 0.863333, test accuracy 0.6425
Training classifier with maxDepth=16...
Training accuracy 0.888333, test accuracy 0.625
```

WU4:

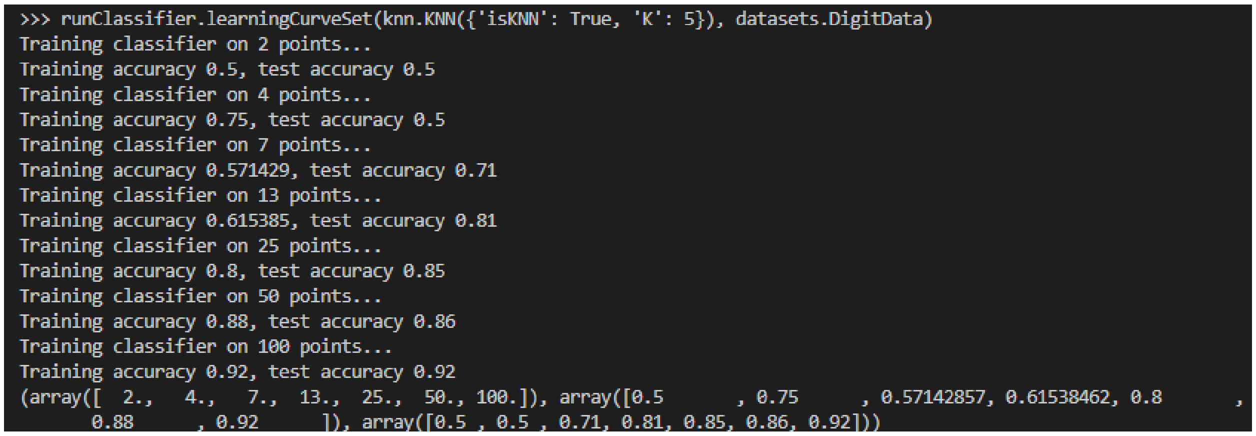
When k = 1, the learning curve seems obviously overfitting:



```
>>> runClassifier.learningCurveSet(knn.KNN({'isKNN': True, 'K': 1}), datasets.DigitData)
Training classifier on 2 points...
Training accuracy 1, test accuracy 0.81
Training classifier on 4 points...
Training accuracy 1, test accuracy 0.8
Training classifier on 7 points...
Training accuracy 1, test accuracy 0.86
Training classifier on 13 points...
Training accuracy 1, test accuracy 0.87
Training classifier on 25 points...
Training accuracy 1, test accuracy 0.89
Training classifier on 50 points...
Training accuracy 1, test accuracy 0.91
Training classifier on 100 points...
Training accuracy 1, test accuracy 0.94
(array([  2.,   4.,   7.,  13.,  25.,  50., 100.]), array([1., 1., 1., 1., 1., 1., 1.]), array([0.81, 0.8 , 0.86, 0.87, 0.89, 0.91, 0.94]))
```

You can see that all the training accuracies are 1's, yet the test accuracies range from 0.8 to 0.94.

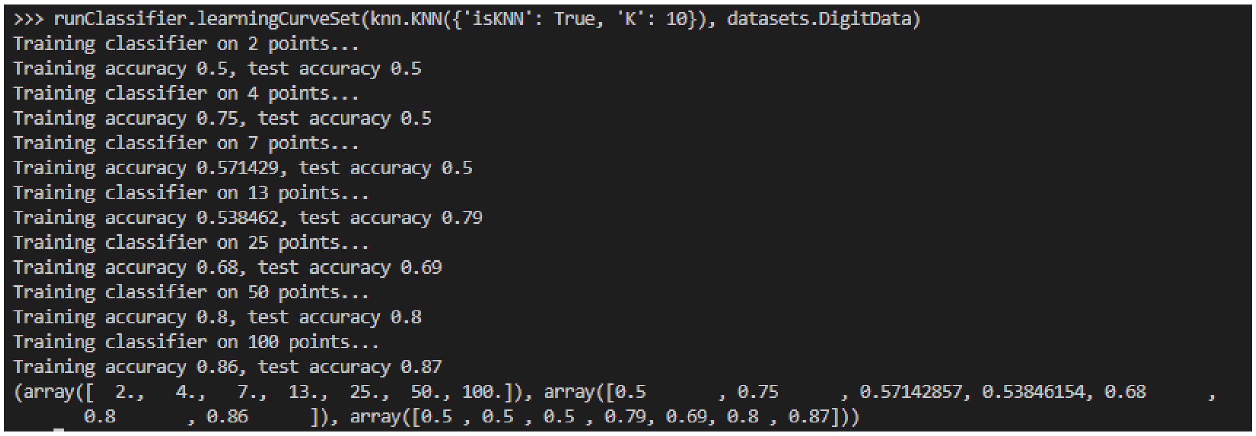
When k = 5, the learning curve shows great improvements as we include more data:



```
>>> runClassifier.learningCurveSet(knn.KNN({'isKNN': True, 'K': 5}), datasets.DigitData)
Training classifier on 2 points...
Training accuracy 0.5, test accuracy 0.5
Training classifier on 4 points...
Training accuracy 0.75, test accuracy 0.5
Training classifier on 7 points...
Training accuracy 0.571429, test accuracy 0.71
Training classifier on 13 points...
Training accuracy 0.615385, test accuracy 0.81
Training classifier on 25 points...
Training accuracy 0.8, test accuracy 0.85
Training classifier on 50 points...
Training accuracy 0.88, test accuracy 0.86
Training classifier on 100 points...
Training accuracy 0.92, test accuracy 0.92
(array([  2.,   4.,   7.,  13.,  25.,  50., 100.]), array([0.5       , 0.75      , 0.57142857, 0.61538462, 0.8       ,
       0.88      , 0.92      ]), array([0.5 , 0.5 , 0.71, 0.81, 0.85, 0.86, 0.92]))
```

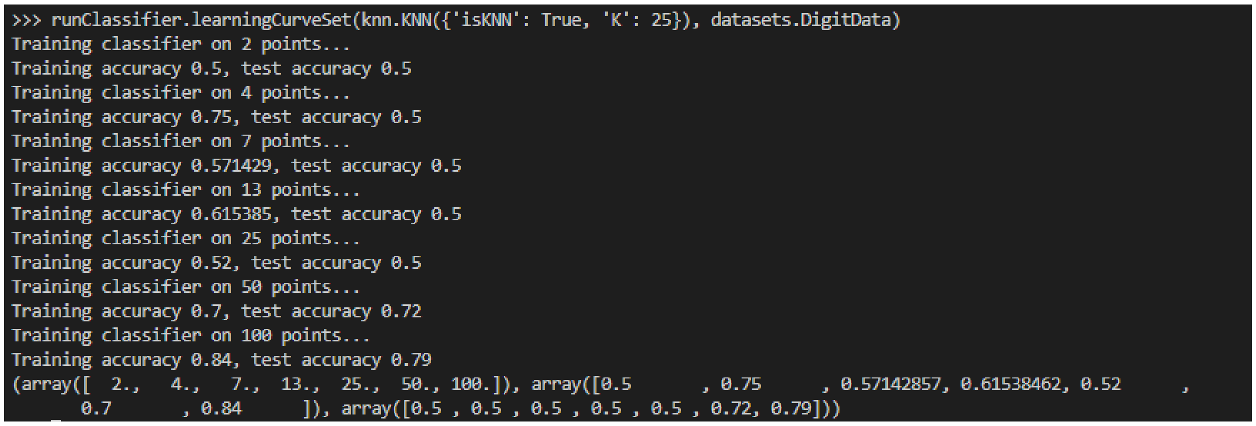
The first train/test accuracy is only 0.5 because we only used 2 data points and it's smaller than k, so there is no "nearest neighbor" as all data points are within range.

When k = 10, we start to see the train/test accuracies drop:



```
>>> runClassifier.learningCurveSet(knn.KNN({'isKNN': True, 'K': 10}), datasets.DigitData)
Training classifier on 2 points...
Training accuracy 0.5, test accuracy 0.5
Training classifier on 4 points...
Training accuracy 0.75, test accuracy 0.5
Training classifier on 7 points...
Training accuracy 0.571429, test accuracy 0.5
Training classifier on 13 points...
Training accuracy 0.538462, test accuracy 0.79
Training classifier on 25 points...
Training accuracy 0.68, test accuracy 0.69
Training classifier on 50 points...
Training accuracy 0.8, test accuracy 0.8
Training classifier on 100 points...
Training accuracy 0.86, test accuracy 0.87
(array([  2.,   4.,   7.,  13.,  25.,  50., 100.]), array([0.5       , 0.75      , 0.57142857, 0.53846154, 0.68      ,
       0.8       , 0.86      ]), array([0.5 , 0.5 , 0.5 , 0.79, 0.69, 0.8 , 0.87]))
```

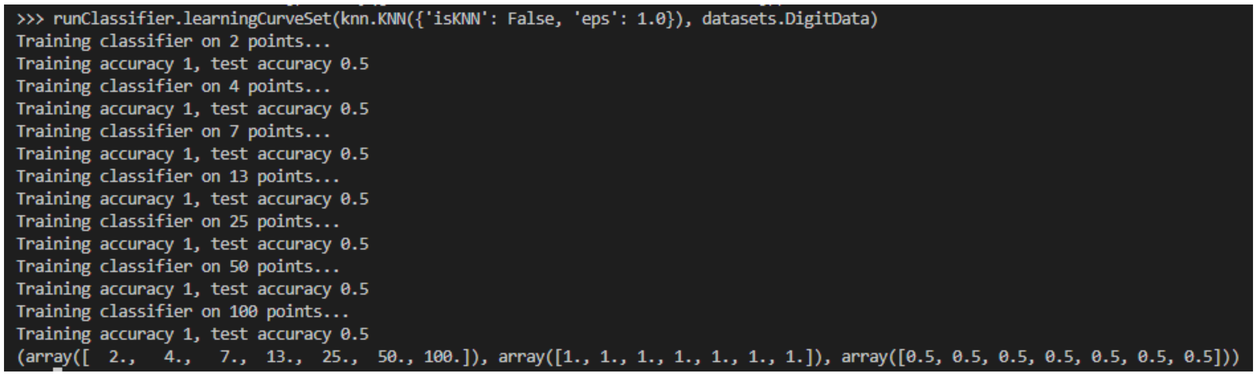
When k = 25, you can see clear signs of underfitting in the early accuracies due to not enough data points being used:



```
>>> runClassifier.learningCurveSet(knn.KNN({'isKNN': True, 'K': 25}), datasets.DigitData)
Training classifier on 2 points...
Training accuracy 0.5, test accuracy 0.5
Training classifier on 4 points...
Training accuracy 0.75, test accuracy 0.5
Training classifier on 7 points...
Training accuracy 0.571429, test accuracy 0.5
Training classifier on 13 points...
Training accuracy 0.615385, test accuracy 0.5
Training classifier on 25 points...
Training accuracy 0.52, test accuracy 0.5
Training classifier on 50 points...
Training accuracy 0.7, test accuracy 0.72
Training classifier on 100 points...
Training accuracy 0.84, test accuracy 0.79
(array([  2.,   4.,   7.,  13.,  25.,  50., 100.]), array([0.5       , 0.75      , 0.57142857, 0.61538462, 0.52      ,
       0.7       , 0.84      ]), array([0.5 , 0.5 , 0.5 , 0.5 , 0.5 , 0.72, 0.79]))
```

As 25 >= 2, 4, 7, 13, and 25, the first 5 test accuracies are all 0.5.

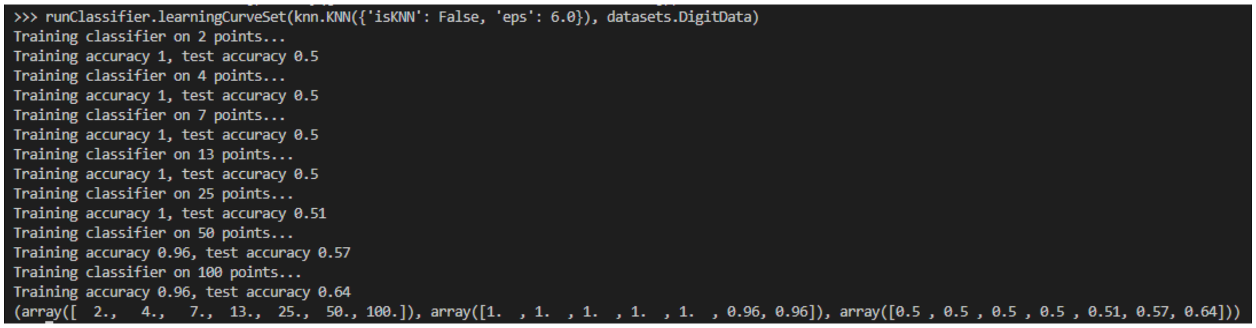
When we are using the epsilon version of knn with eps = 1.0, we see clear underfitting of data:



```
>>> runClassifier.learningCurveSet(knn.KNN({'isKNN': False, 'eps': 1.0}), datasets.DigitData)
Training classifier on 2 points...
Training accuracy 1, test accuracy 0.5
Training classifier on 4 points...
Training accuracy 1, test accuracy 0.5
Training classifier on 7 points...
Training accuracy 1, test accuracy 0.5
Training classifier on 13 points...
Training accuracy 1, test accuracy 0.5
Training classifier on 25 points...
Training accuracy 1, test accuracy 0.5
Training classifier on 50 points...
Training accuracy 1, test accuracy 0.5
Training classifier on 100 points...
Training accuracy 1, test accuracy 0.5
(array([  2.,   4.,   7.,  13.,  25.,  50., 100.]), array([1., 1., 1., 1., 1., 1., 1.]), array([0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]))
```

Since the epsilon is too small, we basically predict on close to no information, hence all the test accuracies resulted in 0.5, signifying pure guesses.

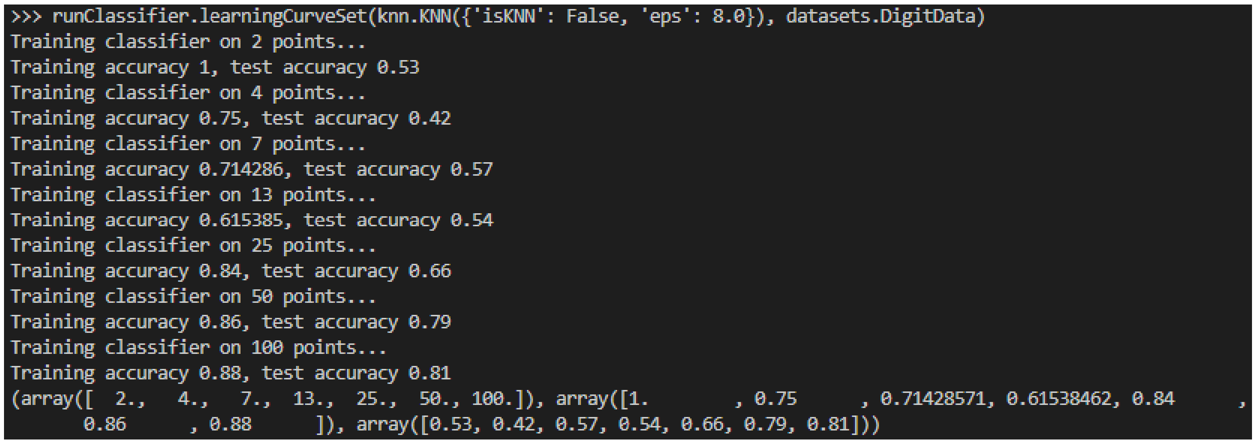
When we have eps = 6.0, we see some improvements:



```
>>> runClassifier.learningCurveSet(knn.KNN({'isKNN': False, 'eps': 6.0}), datasets.DigitData)
Training classifier on 2 points...
Training accuracy 1, test accuracy 0.5
Training classifier on 4 points...
Training accuracy 1, test accuracy 0.5
Training classifier on 7 points...
Training accuracy 1, test accuracy 0.5
Training classifier on 13 points...
Training accuracy 1, test accuracy 0.5
Training classifier on 25 points...
Training accuracy 1, test accuracy 0.51
Training classifier on 50 points...
Training accuracy 0.96, test accuracy 0.57
Training classifier on 100 points...
Training accuracy 0.96, test accuracy 0.64
(array([  2.,   4.,   7.,  13.,  25.,  50., 100.]), array([1.  , 1.  , 1.  , 1.  , 1.  , 0.96, 0.96]), array([0.5 , 0.5 , 0.5 , 0.5 , 0.51, 0.57, 0.64]))
```

You can see that early tests with little data points still have accuracies of 0.5 because the density of data points weren't enough to support low epsilon.

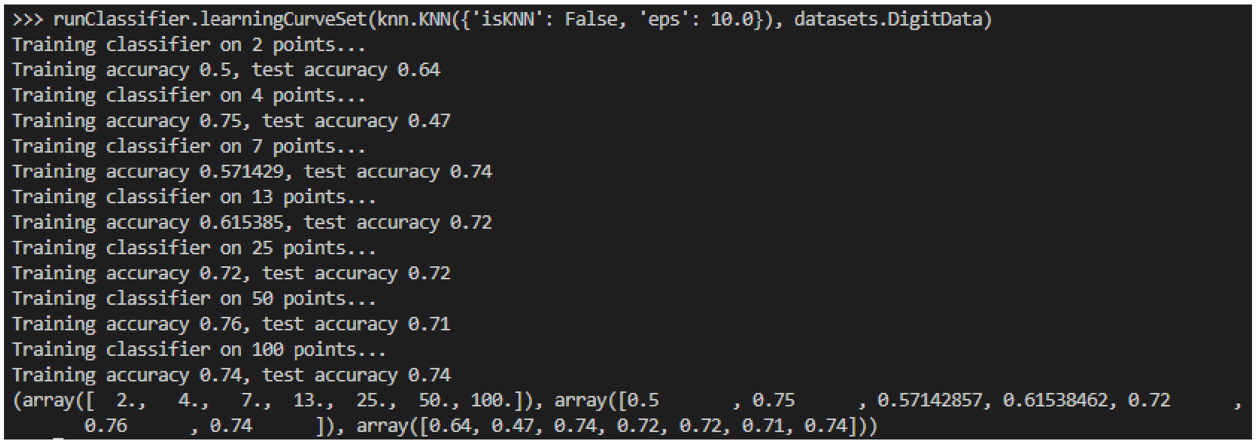
When we have eps = 8.0, we see the best score:



```
>>> runClassifier.learningCurveSet(knn.KNN({'isKNN': False, 'eps': 8.0}), datasets.DigitData)
Training classifier on 2 points...
Training accuracy 1, test accuracy 0.53
Training classifier on 4 points...
Training accuracy 0.75, test accuracy 0.42
Training classifier on 7 points...
Training accuracy 0.714286, test accuracy 0.57
Training classifier on 13 points...
Training accuracy 0.615385, test accuracy 0.54
Training classifier on 25 points...
Training accuracy 0.84, test accuracy 0.66
Training classifier on 50 points...
Training accuracy 0.86, test accuracy 0.79
Training classifier on 100 points...
Training accuracy 0.88, test accuracy 0.81
(array([  2.,   4.,   7.,  13.,  25.,  50., 100.]), array([1.        , 0.75      , 0.71428571, 0.61538462, 0.84      ,
       0.86      , 0.88      ]), array([0.53, 0.42, 0.57, 0.54, 0.66, 0.79, 0.81]))
```

What if we have higher eps? Does that improve the test accuracies even more?

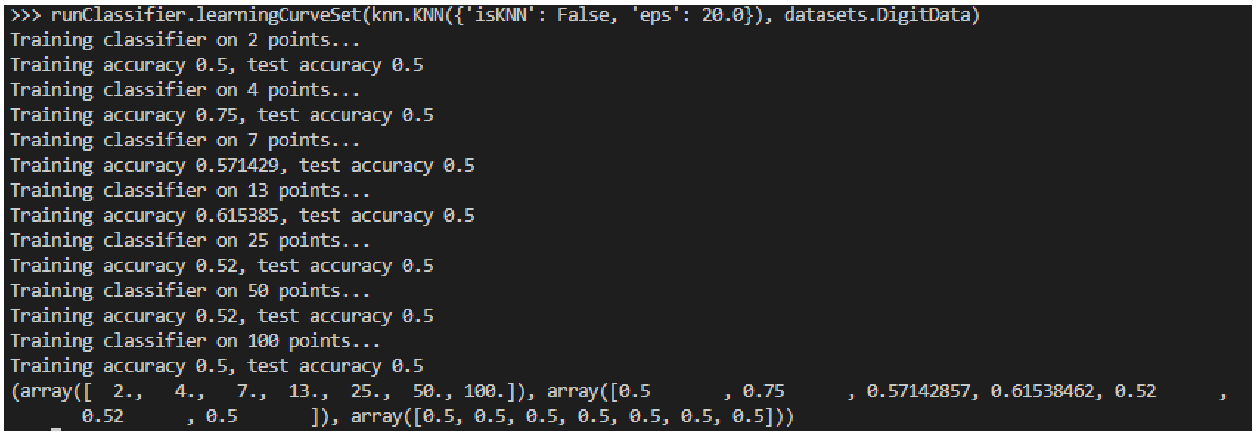
When we have eps = 10.0, we see a drop in accuracy:



```
>>> runClassifier.learningCurveSet(knn.KNN({'isKNN': False, 'eps': 10.0}), datasets.DigitData)
Training classifier on 2 points...
Training accuracy 0.5, test accuracy 0.64
Training classifier on 4 points...
Training accuracy 0.75, test accuracy 0.47
Training classifier on 7 points...
Training accuracy 0.571429, test accuracy 0.74
Training classifier on 13 points...
Training accuracy 0.615385, test accuracy 0.72
Training classifier on 25 points...
Training accuracy 0.72, test accuracy 0.72
Training classifier on 50 points...
Training accuracy 0.76, test accuracy 0.71
Training classifier on 100 points...
Training accuracy 0.74, test accuracy 0.74
(array([  2.,   4.,   7.,  13.,  25.,  50., 100.]), array([0.5       , 0.75      , 0.57142857, 0.61538462, 0.72      ,
       0.76      , 0.74      ]), array([0.64, 0.47, 0.74, 0.72, 0.72, 0.71, 0.74]))
```

Turns out, the best is eps = 8.5, with accuracy of 0.86.

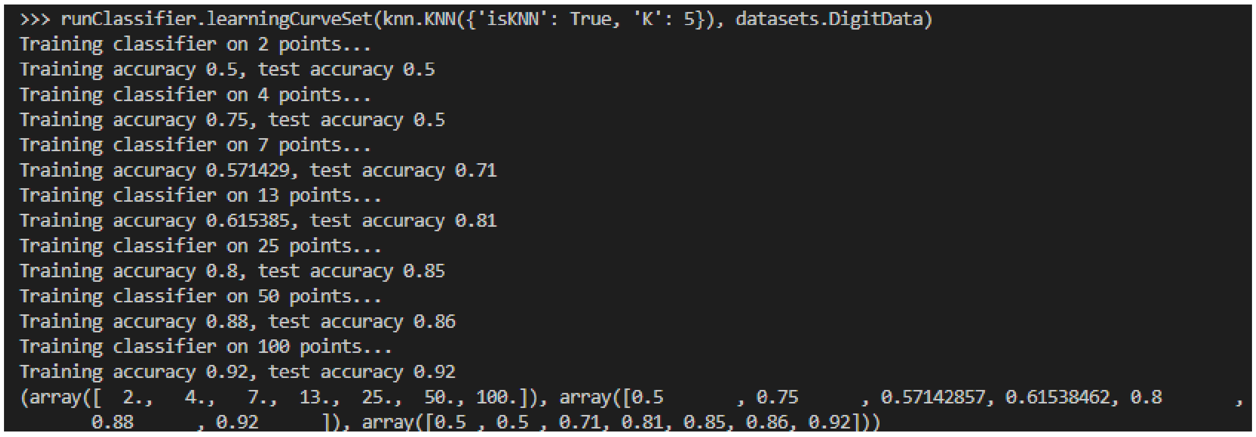
When we have eps = 20.0, we see that the classifier is underfitting:



```
>>> runClassifier.learningCurveSet(knn.KNN({'isKNN': False, 'eps': 20.0}), datasets.DigitData)
Training classifier on 2 points...
Training accuracy 0.5, test accuracy 0.5
Training classifier on 4 points...
Training accuracy 0.75, test accuracy 0.5
Training classifier on 7 points...
Training accuracy 0.571429, test accuracy 0.5
Training classifier on 13 points...
Training accuracy 0.615385, test accuracy 0.5
Training classifier on 25 points...
Training accuracy 0.52, test accuracy 0.5
Training classifier on 50 points...
Training accuracy 0.52, test accuracy 0.5
Training classifier on 100 points...
Training accuracy 0.5, test accuracy 0.5
(array([  2.,   4.,   7.,  13.,  25.,  50., 100.]), array([0.5       , 0.75      , 0.57142857, 0.61538462, 0.52      ,
       0.52      , 0.5       ]), array([0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]))
```

With large eps like this, the classifier probably includes too many data points for prediction.

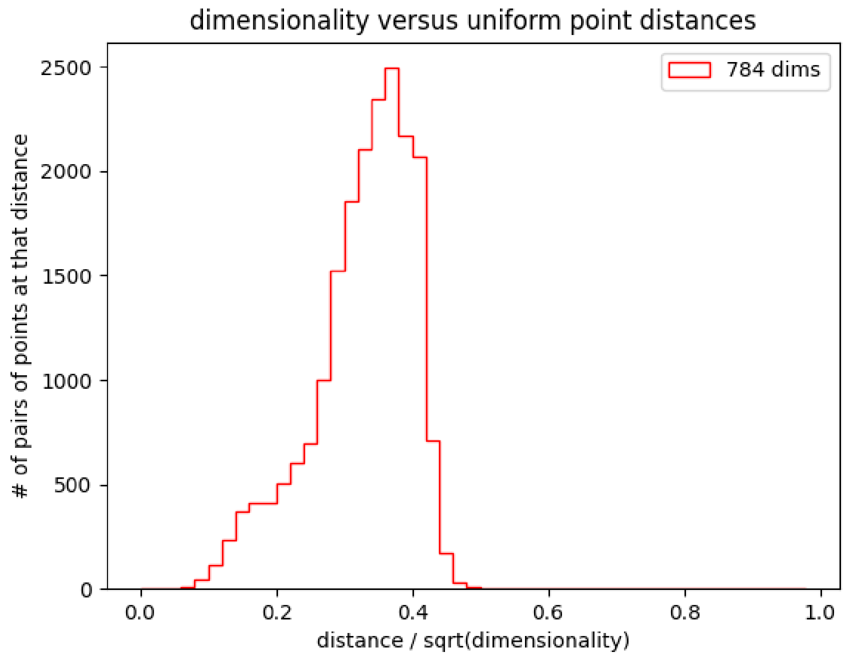
Next, we use k = 5, and the learning curve with digit data is shown below:



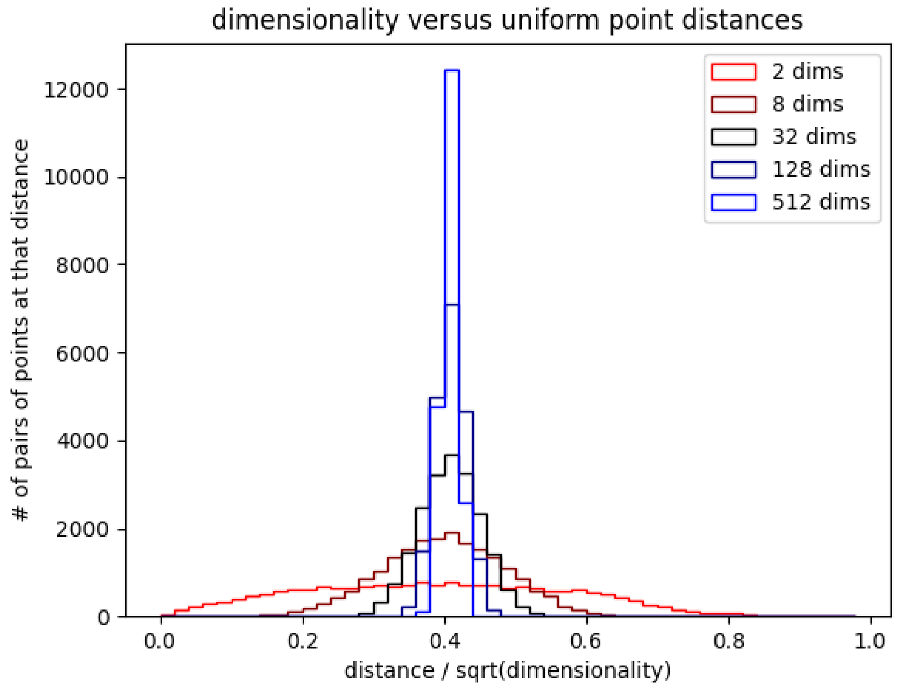
```
>>> runClassifier.learningCurveSet(knn.KNN({'isKNN': True, 'K': 5}), datasets.DigitData)
Training classifier on 2 points...
Training accuracy 0.5, test accuracy 0.5
Training classifier on 4 points...
Training accuracy 0.75, test accuracy 0.5
Training classifier on 7 points...
Training accuracy 0.571429, test accuracy 0.71
Training classifier on 13 points...
Training accuracy 0.615385, test accuracy 0.81
Training classifier on 25 points...
Training accuracy 0.8, test accuracy 0.85
Training classifier on 50 points...
Training accuracy 0.88, test accuracy 0.86
Training classifier on 100 points...
Training accuracy 0.92, test accuracy 0.92
(array([  2.,   4.,   7.,  13.,  25.,  50., 100.]), array([0.5       , 0.75      , 0.57142857, 0.61538462, 0.8       ,
       0.88      , 0.92      ]), array([0.5 , 0.5 , 0.71, 0.81, 0.85, 0.86, 0.92]))
```

WU5:

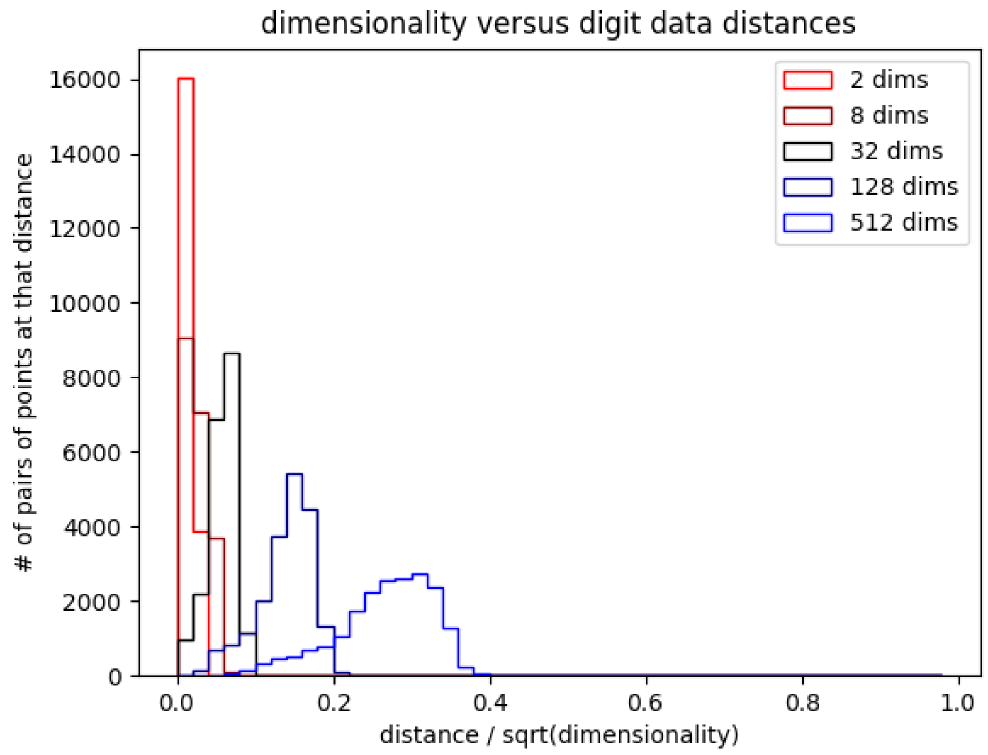
The histogram of raw digits data in 784 dimensions is shown below:



The plot to HighD with d in [2, 8, 32, 128, 512] with random data is shown below:



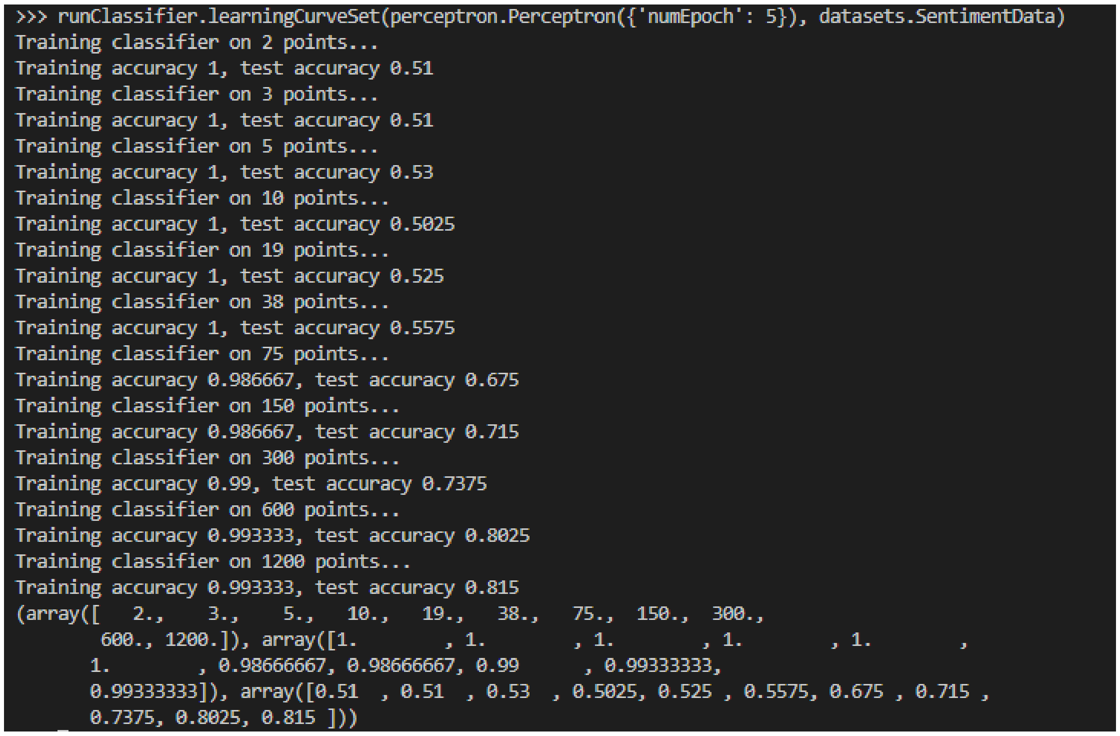
The plot to HighD with same d in [2, 8, 32, 128, 512] with fixed dimensionality on the digit data is shown below:



The uniform points data was completely random, so the plot represented a bell shaped curve to some degree. The digit data is clearly not random, as we can see that the distances peak at lower values than the ones in the uniform points plot. This indicates that the distribution of data points in the digit data dataset is a lot closer than randomly placed uniform points.

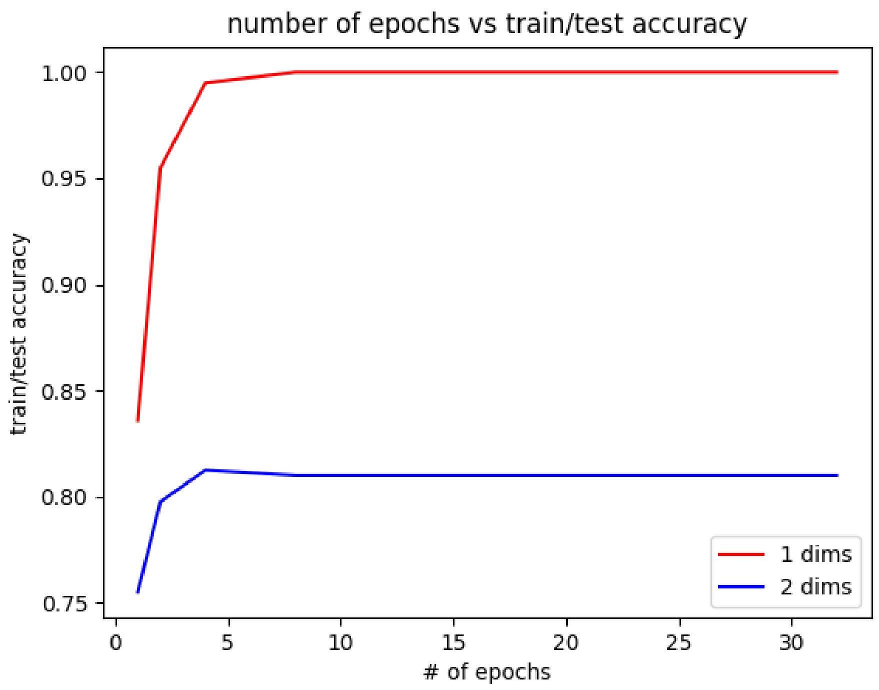
WU6:

The learning curve for the perceptron with 5 epochs on the sentiment data is shown below:



```
>>> runClassifier.learningCurveSet(perceptron.Perceptron({'numEpoch': 5}), datasets.SentimentData)
Training classifier on 2 points...
Training accuracy 1, test accuracy 0.51
Training classifier on 3 points...
Training accuracy 1, test accuracy 0.51
Training classifier on 5 points...
Training accuracy 1, test accuracy 0.53
Training classifier on 10 points...
Training accuracy 1, test accuracy 0.5025
Training classifier on 19 points...
Training accuracy 1, test accuracy 0.525
Training classifier on 38 points...
Training accuracy 1, test accuracy 0.5575
Training classifier on 75 points...
Training accuracy 0.986667, test accuracy 0.675
Training classifier on 150 points...
Training accuracy 0.986667, test accuracy 0.715
Training classifier on 300 points...
Training accuracy 0.99, test accuracy 0.7375
Training classifier on 600 points...
Training accuracy 0.993333, test accuracy 0.8025
Training classifier on 1200 points...
Training accuracy 0.993333, test accuracy 0.815
(array([   2.,    3.,    5.,   10.,   19.,   38.,   75.,  150.,  300.,
        600., 1200.]), array([1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 0.98666667, 0.98666667, 0.99      , 0.99333333,
       0.99333333]), array([0.51  , 0.51  , 0.53  , 0.5025, 0.525 , 0.5575, 0.675 , 0.715 ,
       0.7375, 0.8025, 0.815 ]))
```

A plot of number of epochs versus train/test accuracy on the entire dataset is shown below:



The perceptron quickly converged to around accuracy = 0.81 for testing as the training accuracy reached 1.