WU Questions

WU1:

Why is this computation equivalent to computing classification accuracy?

The statement within the mean function returns a 1 if the prediction is correct and a 0 if the prediction is incorrect. The mean function then takes the average of all the 1s and 0s, which is the same as the classification accuracy.

As reference, accuracy is

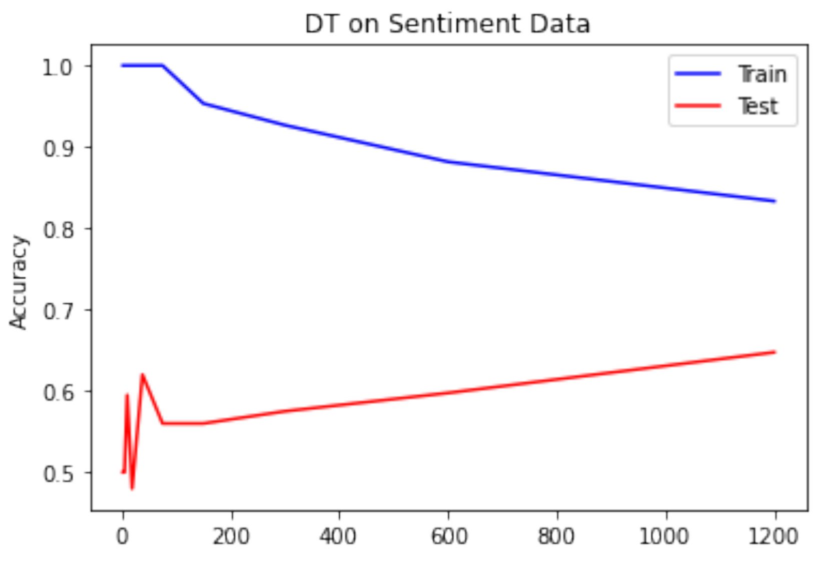
$$\frac{TP+TN}{TP+TN+FP+FN}$$

, which is the same as the probability that the classification is done correctly, which corresponds to the mean of that function.

WU2:

```
In [ ]: import dt
        from imports import *

        curve = runClassifier.learningCurveSet(dt.DT({'maxDepth': 9}), datasets.S
        runClassifier.plotCurve('DT on Sentiment Data', curve)
```

```
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
```



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 go down as we have more data points because when we have more data points, the error rate on the training data will increase as a large training set is more difficult to fit into.

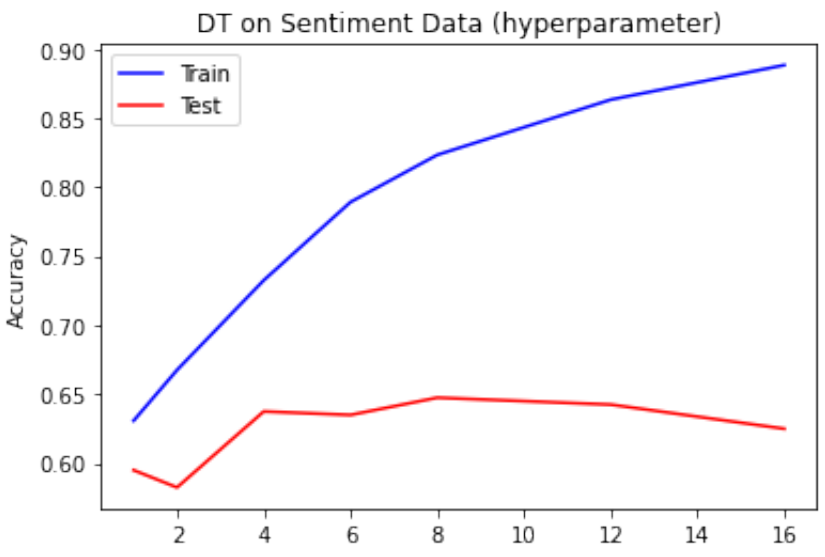
Test accuracy is not monotonically increasing because the first few data points are data that the model has never seen before, and are naturally expected to be inaccurate. But once the model has seen enough data points, it will be able to make more accurate predictions.

The jaggedness in the test curve toward the left is also because the model is unable to form accurate predictions for the first few never-seen-before data in the test set.

WU3:

```
In [ ]: curve = runClassifier.hyperparamCurveSet(dt.DT({}), 'maxDepth', [1,2,4,6,
        runClassifier.plotCurve('DT on Sentiment Data (hyperparameter)', curve)
```

```
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
```



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?

Training accuracy is guaranteed to monotonically increase because it is training the model and learning and becoming better. This is what we are training for and therefore is what we are guaranteed to see.

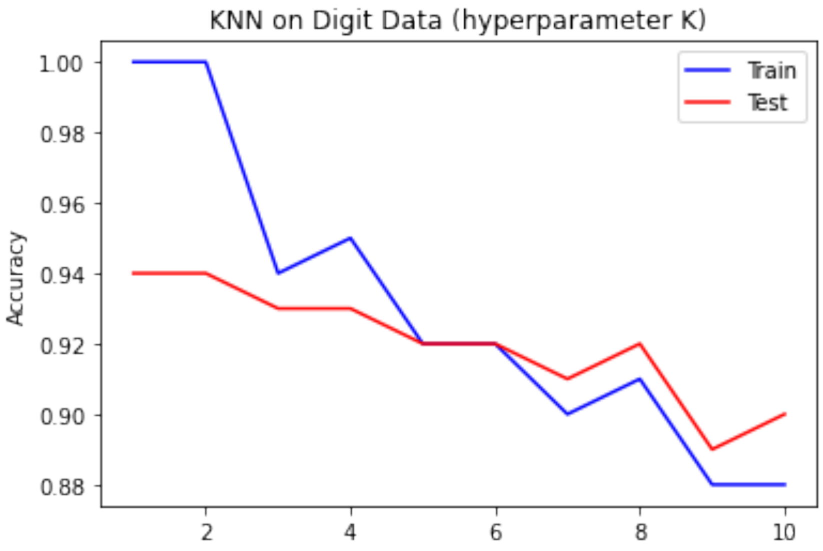
Test accuracy making something like a hill is expected because we are now testing our model on unseen data and the more training data there is, the more the model tends to overfit the training data, and the less the model will be able to fit the test data.

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.

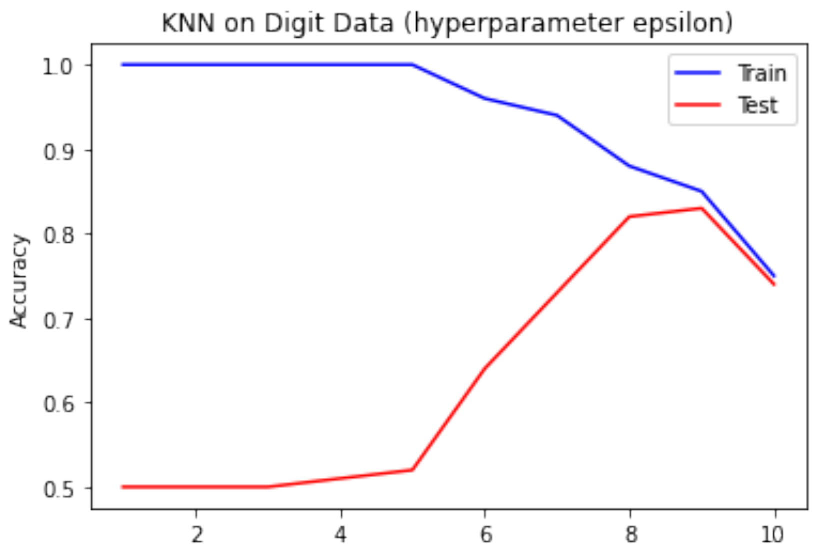
```
In [ ]: import knn
        from imports import *

        curve = runClassifier.hyperparamCurveSet(knn.KNN({'isKNN':True}), 'K', [1
        runClassifier.plotCurve('KNN on Digit Data (hyperparameter K)', curve)
```

```
Training classifier with K=1...
Training accuracy 1, test accuracy 0.94
Training classifier with K=2...
Training accuracy 1, test accuracy 0.94
Training classifier with K=3...
Training accuracy 0.94, test accuracy 0.93
Training classifier with K=4...
Training accuracy 0.95, test accuracy 0.93
Training classifier with K=5...
Training accuracy 0.92, test accuracy 0.92
Training classifier with K=6...
Training accuracy 0.92, test accuracy 0.92
Training classifier with K=7...
Training accuracy 0.9, test accuracy 0.91
Training classifier with K=8...
Training accuracy 0.91, test accuracy 0.92
Training classifier with K=9...
Training accuracy 0.88, test accuracy 0.89
Training classifier with K=10...
Training accuracy 0.88, test accuracy 0.9
```



From this plot, it is shown when K is low. There is an overfit where the training accuracy is very high which is 1

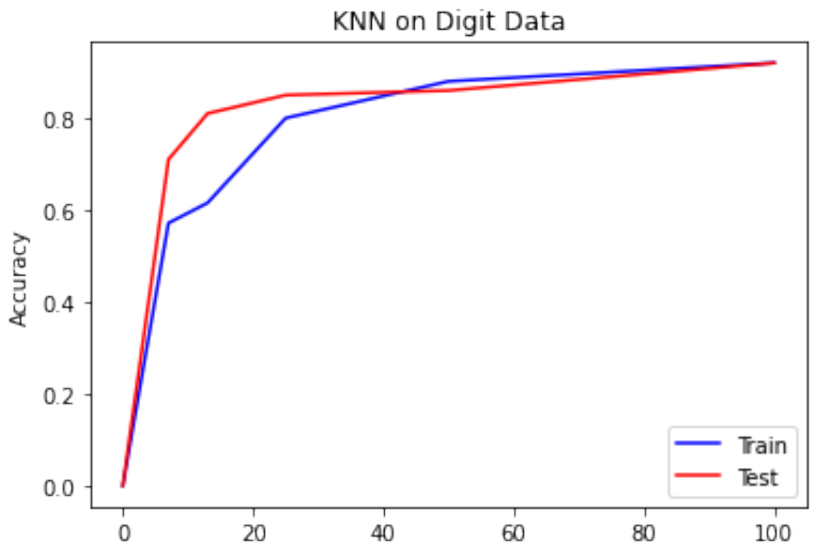
```
In [ ]: curve = runClassifier.hyperparamCurveSet(knn.KNN({'isKNN':False}), 'eps',
        runClassifier.plotCurve('KNN on Digit Data (hyperparameter epsilon)', cur
        Training classifier with eps=1...
        Training accuracy 1, test accuracy 0.5
        Training classifier with eps=2...
        Training accuracy 1, test accuracy 0.5
        Training classifier with eps=3...
        Training accuracy 1, test accuracy 0.5
        Training classifier with eps=4...
        Training accuracy 1, test accuracy 0.51
        Training classifier with eps=5...
        Training accuracy 1, test accuracy 0.52
        Training classifier with eps=6...
        Training accuracy 0.96, test accuracy 0.64
        Training classifier with eps=7...
        Training accuracy 0.94, test accuracy 0.73
        Training classifier with eps=8...
        Training accuracy 0.88, test accuracy 0.82
        Training classifier with eps=9...
        Training accuracy 0.85, test accuracy 0.83
        Training classifier with eps=10...
        Training accuracy 0.75, test accuracy 0.74
```



From this plot, it is shown when epsilon is low. There is a overfit where training accuracy is very high and test accuracy is low

In [ ]:
```
curve = runClassifier.learningCurveSet_knn(knn.KNN({'isKNN': True, 'K': 5
runClassifier.plotCurve('KNN on Digit Data', curve)
```

```
Skip 2 points...
Skip 4 points...
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
```

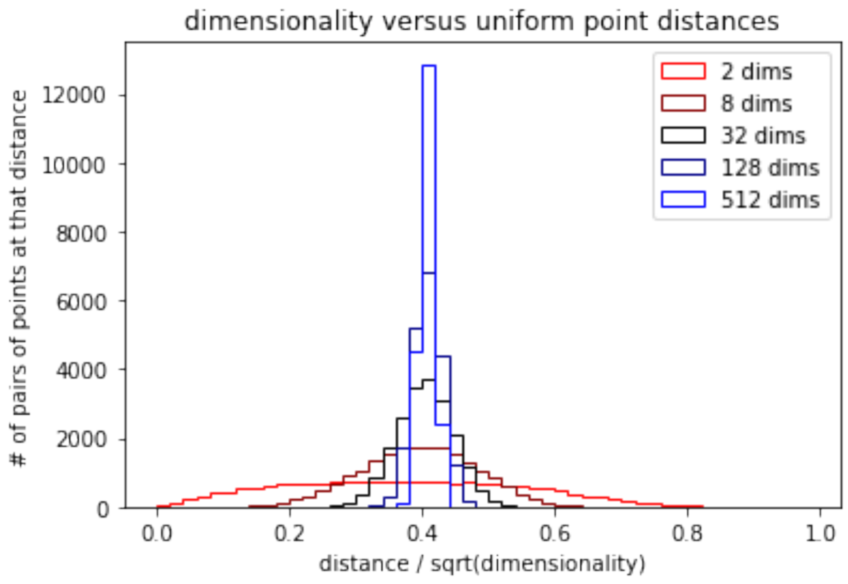


WU5:

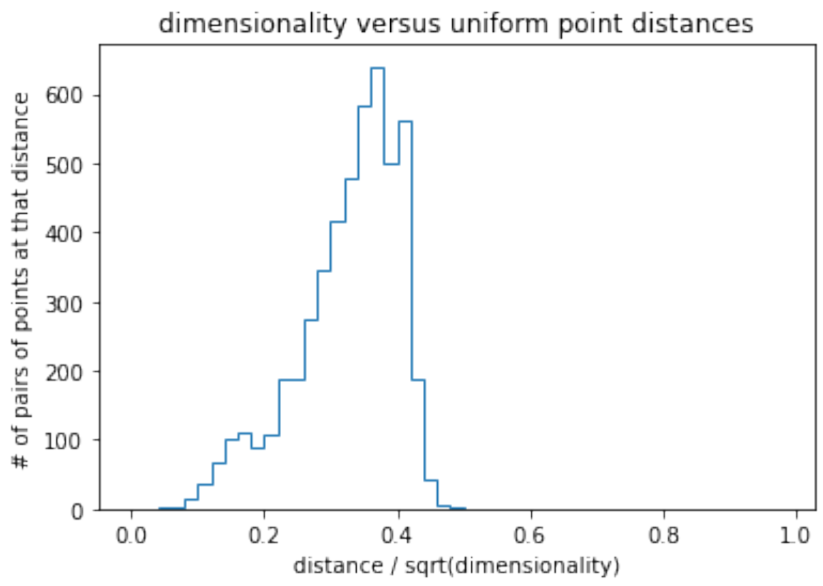
WU5: A. First, get a histogram of the raw digits data in 784 dimensions. You'll probably want to use the computeDistances function together with the plotting in HighD. B. Rewrite computeDistances so that it can subsample features down to some fixed dimensionality. For example, you might write computeDistancesSubdims(data, 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.

```
In [ ]: import HighD
        from numpy import *
        import matplotlib.pyplot as plt
        from math import *

        waitForEnter=False
        N    = 200                     # number of examples
        Bins = arange(0, 1, 0.02)

        plt.xlabel('distance / sqrt(dimensionality)')
        plt.ylabel('# of pairs of points at that distance')
        plt.title('dimensionality versus uniform point distances')

        distances = HighD.computeDistances(datasets.DigitData.X)
        print ("D=%d, average distance=%g" % (784, mean(distances) * sqrt(784)))
        plt.hist(distances,
                 Bins,
                 histtype='step')
        if waitForEnter:
            plt.legend(['%d dims' % d for d in Dims])
            plt.show()
            x = raw_input('Press enter to continue...')
```

```
D=2, average distance=0.527586
D=8, average distance=1.11844
D=32, average distance=2.28364
D=128, average distance=4.61282
D=512, average distance=9.22782
```

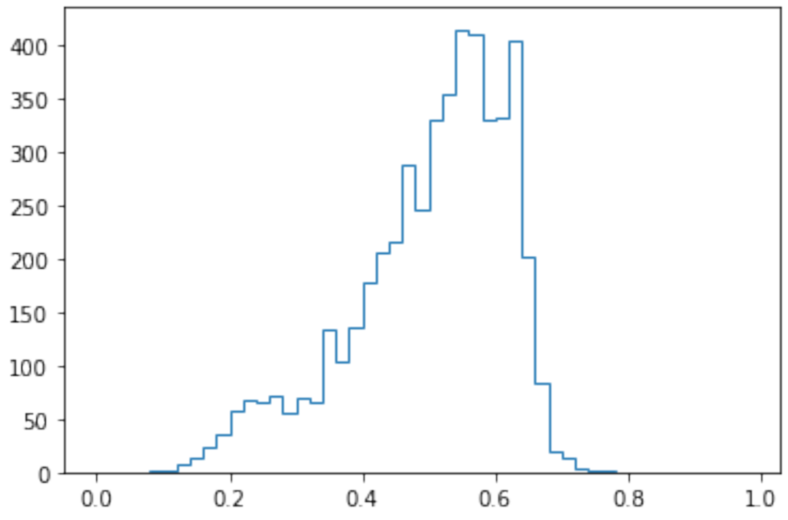


```
D=784, average distance=9.10764
```



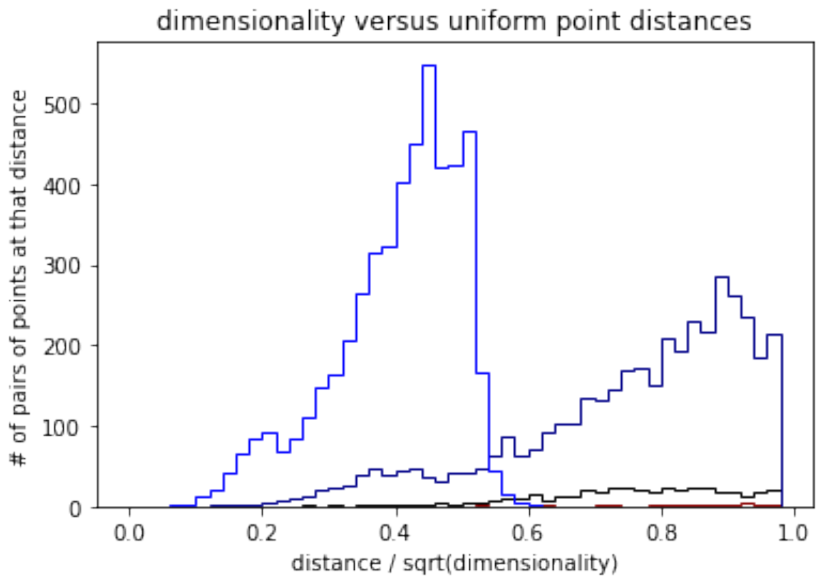
In [ ]:
```
def computeDistances(data, d):
    N = len(data)
    D = d
    dist = []
    for n in range(N):
        for m in range(n):
            dist.append(HighD.computeExampleDistance(data[n],data[m])  /
    return dist

d= random.randint(1, 784)
distances = computeDistances(datasets.DigitData.X, d)

print ("D=%d, average distance=%g" % (d, mean(distances) * sqrt(d)))
plt.hist(distances,
         Bins,
         histtype='step')
if waitForEnter:
    plt.legend(['%d dims' % d for d in Dims])
    plt.show()
    x = raw_input('Press enter to continue...')
```

```
D=331, average distance=9.10764
```



```
In [ ]: import HighD
        from numpy import *
        import matplotlib.pyplot as plt
        from math import *

        waitForEnter=False
        N    = 200                          # number of examples
        Dims = [2, 8, 32, 128, 512]    # dimensionalities to try
        Cols = ['#FF0000', '#880000', '#000000', '#000088', '#0000FF']
        Bins = arange(0, 1, 0.02)

        plt.xlabel('distance / sqrt(dimensionality)')
        plt.ylabel('# of pairs of points at that distance')
        plt.title('dimensionality versus uniform point distances')

        for i,d in enumerate(Dims):
            distances = computeDistances(datasets.DigitData.X, d)
            print ("D=%d, average distance=%g" % (d, mean(distances) * sqrt(d)))
            plt.hist(distances,
                     Bins,
                     histtype='step',
                     color=Cols[i])
            if waitForEnter:
                plt.legend(['%d dims' % d for d in Dims])
                plt.show()
                x = raw_input('Press enter to continue...')
```

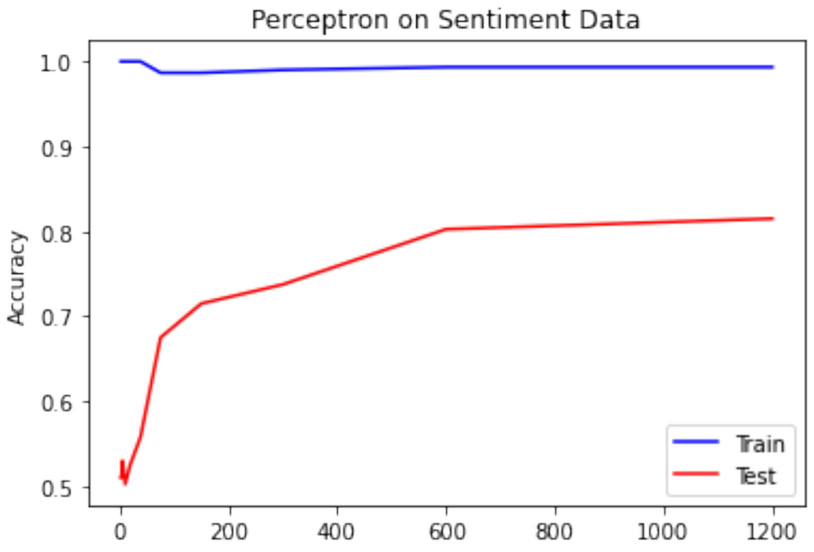
```
D=2, average distance=9.10764
D=8, average distance=9.10764
D=32, average distance=9.10764
D=128, average distance=9.10764
D=512, average distance=9.10764
```



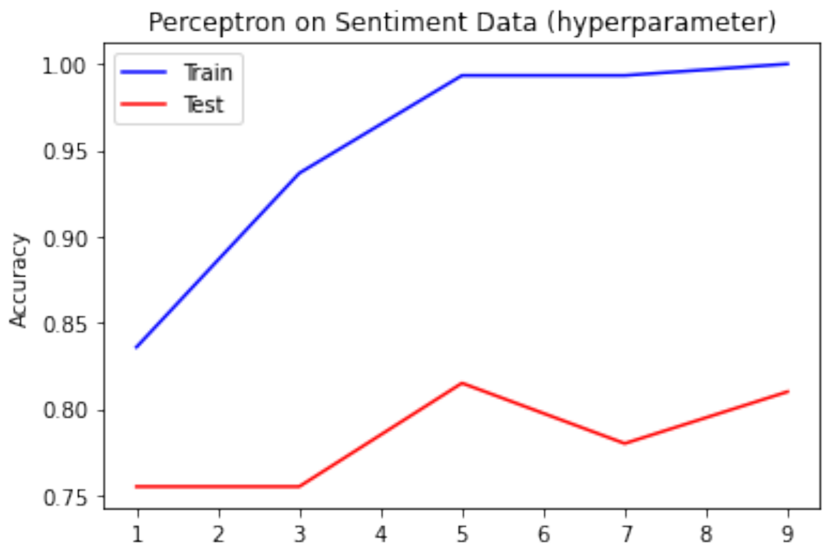
WU6:

In [ ]:
```
import perceptron

curve = runClassifier.learningCurveSet(perceptron.Perceptron({'numEpoch':
runClassifier.plotCurve('Perceptron on Sentiment Data', curve)
```

```
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
```



In [ ]:
```
curve = runClassifier.hyperparamCurveSet(perceptron.Perceptron({}), 'numE
runClassifier.plotCurve('Perceptron on Sentiment Data (hyperparameter)',
Training classifier with numEpoch=1...
Training accuracy 0.835833, test accuracy 0.755
Training classifier with numEpoch=3...
Training accuracy 0.936667, test accuracy 0.755
Training classifier with numEpoch=5...
Training accuracy 0.993333, test accuracy 0.815
Training classifier with numEpoch=7...
Training accuracy 0.993333, test accuracy 0.78
Training classifier with numEpoch=9...
Training accuracy 1, test accuracy 0.81
```



```
In [ ]:
```