P1_Testing

September 23, 2023

[1]:
```
import util, datasets, binary, dumbClassifiers, runClassifier, dt, knn,␣
↪perceptron
from numpy import *
from pylab import *
```

1 Warming up to Classifiers

[2]:
```
h = dumbClassifiers.AlwaysPredictOne({})
```

[3]:
```
h
```

[3]:
```
AlwaysPredictOne
```

[4]:
```
h.train(datasets.TennisData.X, datasets.TennisData.Y)
```

[5]:
```
h.predictAll(datasets.TennisData.X)
```

[5]:
```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

1.1 WU1: why is this computation equivalent to computing classification accuracy?

[6]:
```
mean((datasets.TennisData.Y > 0) == (h.predictAll(datasets.TennisData.X) > 0))
```

[6]:
```
0.6428571428571429
```

The above calculates the classification accuracy because (datasets.TennisData.Y > 0) finds where all the true labels are 1 and (h.predictAll(datasets.TennisData.X) > 0) finds where all the predicted labels are 1 (does this by putting true or false in the corresponding index). Then == finds where both labels are 1 (true) or both are 0 (false), which is the same as finding the observations where the classifier made a correct prediction. Then the mean function just sums all the times where the classifier made a correct prediction divided by the total number of predictions it made (the size of the array). Hence, the result is correct_predictions/total_predictions = classification accuracy.

[7]:
```
mean((datasets.TennisData.Yte > 0) == (h.predictAll(datasets.TennisData.Xte) >␣
↪0))
```

[7]:
```
0.5
```

[8]:
```
runClassifier.trainTestSet(h, datasets.TennisData)
```

```
Training accuracy 0.642857, test accuracy 0.5
```

[9]:
```
h = dumbClassifiers.AlwaysPredictMostFrequent({})
```

[10]:
```
runClassifier.trainTestSet(h, datasets.TennisData)
```

```
Training accuracy 0.642857, test accuracy 0.5
```

[11]:
```
runClassifier.trainTestSet(dumbClassifiers.AlwaysPredictOne({}), datasets.
↪SentimentData)
```

```
Training accuracy 0.504167, test accuracy 0.5025
```

[12]:
```
runClassifier.trainTestSet(dumbClassifiers.AlwaysPredictMostFrequent({}),␣
↪datasets.SentimentData)
```

```
Training accuracy 0.504167, test accuracy 0.5025
```

[13]:
```
runClassifier.trainTestSet(dumbClassifiers.FirstFeatureClassifier({}), datasets.
↪TennisData)
runClassifier.trainTestSet(dumbClassifiers.FirstFeatureClassifier({}), datasets.
↪SentimentData)
```

```
Training accuracy 0.714286, test accuracy 0.666667
Training accuracy 0.504167, test accuracy 0.5025
```

2 Decision Trees

[14]:
```
h = dt.DT({'maxDepth': 1})
```

[15]:
```
h
```

[15]:
```
Leaf 1
```

[16]:
```
h.train(datasets.TennisData.X, datasets.TennisData.Y)
```

[17]:
```
h
```

[17]:
```
Branch 6
  Leaf 1.0
  Leaf -1.0
```

[18]:
```
h = dt.DT({'maxDepth': 2})
```

[19]:
```
h.train(datasets.TennisData.X, datasets.TennisData.Y)
```

[20]:
```
h
```

[20]:
```
Branch 6
  Branch 7
    Leaf 1.0
    Leaf 1.0
  Branch 1
    Leaf -1.0
    Leaf 1.0
```

[21]:
```
h = dt.DT({'maxDepth': 5})
```

[22]:
```
h.train(datasets.TennisData.X, datasets.TennisData.Y)
```

[23]:
```
h
```

[23]:
```
Branch 6
  Branch 7
    Leaf 1.0
    Branch 2
      Leaf 1.0
      Leaf -1.0
  Branch 1
    Branch 5
      Branch 4
        Leaf -1.0
        Branch 3
          Leaf -1.0
          Leaf 1.0
      Leaf 1.0
    Leaf 1.0
```

[24]:
```
h = dt.DT({'maxDepth': 2})
```

[25]:
```
h.train(datasets.SentimentData.X, datasets.SentimentData.Y)
```

[26]:
```
h
```

[26]:
```
Branch 626
  Branch 683
    Leaf 1.0
    Leaf -1.0
  Branch 1139
    Leaf -1.0
    Leaf 1.0
```

[27]:
```
datasets.SentimentData.words[h.feature]
```

[27]:
```
'bad'
```

[28]:
```
datasets.SentimentData.words[h.left.feature]
```

[28]:
```
'worst'
```

[29]:
```
datasets.SentimentData.words[h.right.feature]
```

[29]:
```
'sequence'
```

[30]:
```
runClassifier.trainTestSet(dt.DT({'maxDepth': 1}), datasets.SentimentData)
```

```
Training accuracy 0.630833, test accuracy 0.595
```

[31]:
```
runClassifier.trainTestSet(dt.DT({'maxDepth': 3}), datasets.SentimentData)
```

```
Training accuracy 0.701667, test accuracy 0.6175
```

[32]:
```
runClassifier.trainTestSet(dt.DT({'maxDepth': 5}), datasets.SentimentData)
```

```
Training accuracy 0.765833, test accuracy 0.625
```

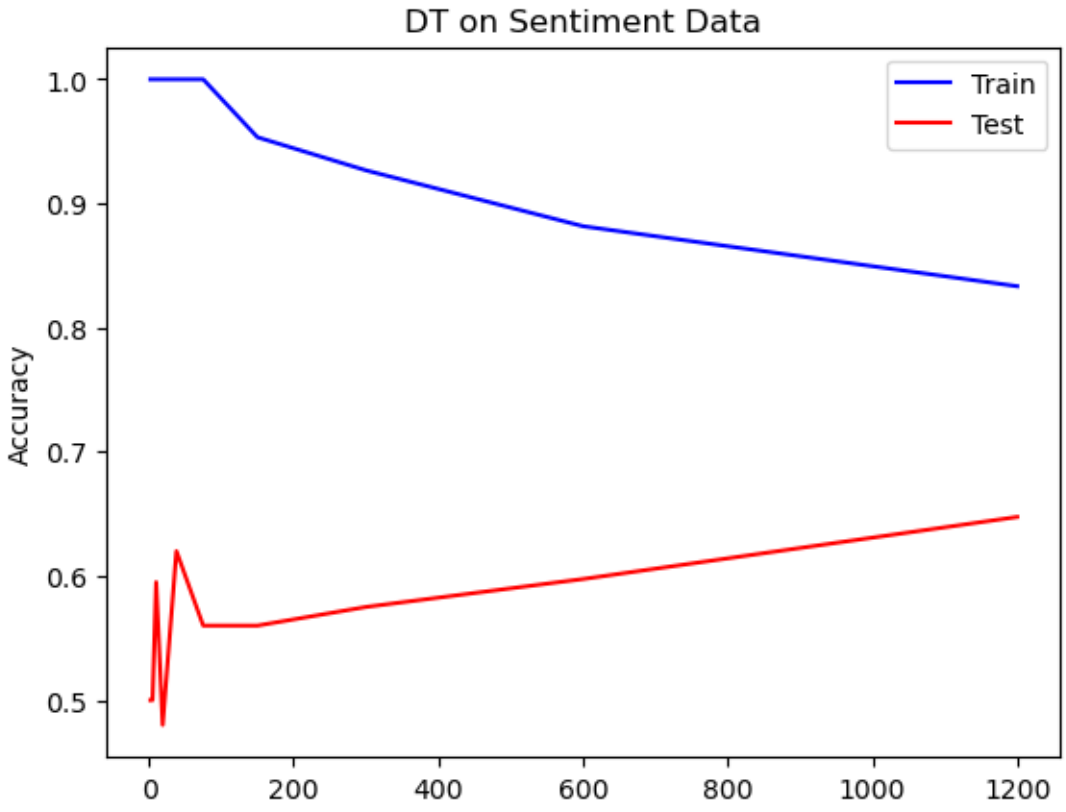
2.1 WU2: We should see training accuracy (roughly) going down and test accuracy (roughly) going up. Why does training accuracy tend to go down? Why is test accuracy not monotonically increasing? You should also see jaggedness in the test curve toward the left. Why?

[33]:
```
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
```

[34]:
```
runClassifier.plotCurve('DT on Sentiment Data', curve)
```



Training accuracy goes down as we use more points to train the tree because it becomes harder for the tree to find perfect boundaries when there are more points (and potentially more outliers). This is because when there are more points, it becomes more likely to find overlapping class "regions" or to have interference in a class region due to outliers from another class. For example, when there are only 2 points to train the data on, the tree could probably find a perfect feature to split on to get 100% accuracy since it only has to worry about the two points which is at max 2 regions. However, when there are 1000 points, there might not be a perfect boundary/combination of boundaries that can separate each point into the correct group (especially when limiting the depth of the tree). As a result, the model makes more misclassifications on the training data, and so the training accuracy decreases.

Test accuracy is not monotonically increasing because as you increase the amount of points the model can be trained on, the model's complexity is allowed to increase (has more points => more likely able to split on more features => higher complexity => may get overfit). Note that as complexity increases, variance increases hence test error may increase. Also, at some point testing accuracy cannot keep increasing because there is irreducible error, which means that the model makes the best prediction it can but the test error still fluctuates/varies due to the random

irreducible error.

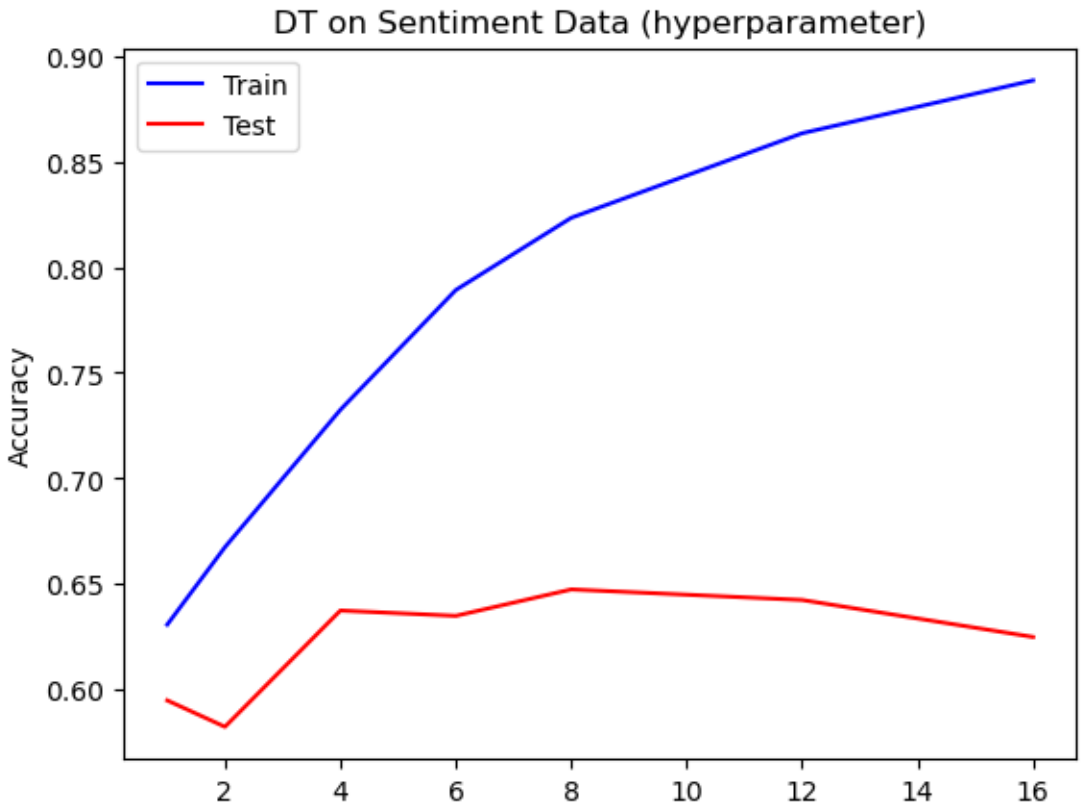
In particular, there's another reason why the test accuracy is not monotonically increasing when you look toward the left. On the left, the model is trained on 2 vs 3 vs 5 vs … vs 38 vs 75 points which is not enough points to get a model that can generalize well (i.e. the models are being fit on only a select few points which causes the trees to have high variance/be highly sensitive to their training sets). This explains the chaotic jumps in the test error of the early models (the models are unstable/highly variable and can't generalize on the test data it hasn't seen); the test errors jumping around were most likely just due to chance as the model couldn't have learned enough from that few points to actually make an "educated" prediction.

2.2 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?

[35]:
```
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
```

[36]:
```
runClassifier.plotCurve('DT on Sentiment Data (hyperparameter)', curve)
```



Training accuracy is guaranteed to decrease as we increase the model's complexity. This is because our train function is written such that training accuracy is maximized and it's greedy. So allowing the tree to be more complex allows the train function to find more splits that maximize training accuracy better and since it's greedy it builds upon the previous tree, so training accuracy never has a chance to decrease.

On the other hand, we expect that the test accuracy should slightly increase as we increase the model's complexity because the model should go from underfit to actually learning some patterns from the data (we expect that the model's complexity starts to approach the true complexity of the relationship, hence increasing test accuracy by decreasing bias). However, at some point, we expect that the model becomes too complex and learns noise/irreducible error that isn't directly produced by the true relationship, which increases variance. Hence, its like the model starts to overthink and makes more errors when trying to generalize on the test data. We expect this to happen due to the bias-variance tradeoff (i.e. increasing complexity => increases variance and decreases bias which has a parabolic effect on the test error). But this trend is not guaranteed to always look like a "hill" because it depends on lots of factors like the true complexity of the relationship, what model you pick, etc.

3 Nearest Neighbors

[37]:
```
runClassifier.trainTestSet(knn.KNN({'isKNN': False, 'eps': 0.5}), datasets.
  ↪TennisData)
```

```
Training accuracy 1, test accuracy 1
```

[38]:
```
runClassifier.trainTestSet(knn.KNN({'isKNN': False, 'eps': 1.0}), datasets.
  ↪TennisData)
```

```
Training accuracy 0.857143, test accuracy 0.833333
```

[39]:
```
runClassifier.trainTestSet(knn.KNN({'isKNN': False, 'eps': 2.0}), datasets.
  ↪TennisData)
```

```
Training accuracy 0.642857, test accuracy 0.5
```

[40]:
```
runClassifier.trainTestSet(knn.KNN({'isKNN': True, 'K': 1}), datasets.
  ↪TennisData)
```

```
Training accuracy 1, test accuracy 1
```

[41]:
```
runClassifier.trainTestSet(knn.KNN({'isKNN': True, 'K': 3}), datasets.
  ↪TennisData)
```

```
Training accuracy 0.785714, test accuracy 0.833333
```

[42]:
```
runClassifier.trainTestSet(knn.KNN({'isKNN': True, 'K': 5}), datasets.
  ↪TennisData)
```

```
Training accuracy 0.857143, test accuracy 0.833333
```

[43]:
```
runClassifier.trainTestSet(knn.KNN({'isKNN': False, 'eps': 6.0}), datasets.
  ↪DigitData)
```

```
Training accuracy 0.96, test accuracy 0.64
```

[44]:
```
runClassifier.trainTestSet(knn.KNN({'isKNN': False, 'eps': 8.0}), datasets.
  ↪DigitData)
```

```
Training accuracy 0.88, test accuracy 0.81
```

[45]:
```
runClassifier.trainTestSet(knn.KNN({'isKNN': False, 'eps': 10.0}), datasets.
  ↪DigitData)
```

```
Training accuracy 0.74, test accuracy 0.74
```

[46]:
```
runClassifier.trainTestSet(knn.KNN({'isKNN': True, 'K': 1}), datasets.DigitData)
```

```
Training accuracy 1, test accuracy 0.94
```

[47]:
```
runClassifier.trainTestSet(knn.KNN({'isKNN': True, 'K': 3}), datasets.DigitData)
```

```
Training accuracy 0.94, test accuracy 0.93
```

[48]:
```
runClassifier.trainTestSet(knn.KNN({'isKNN': True, 'K': 5}), datasets.DigitData)
```

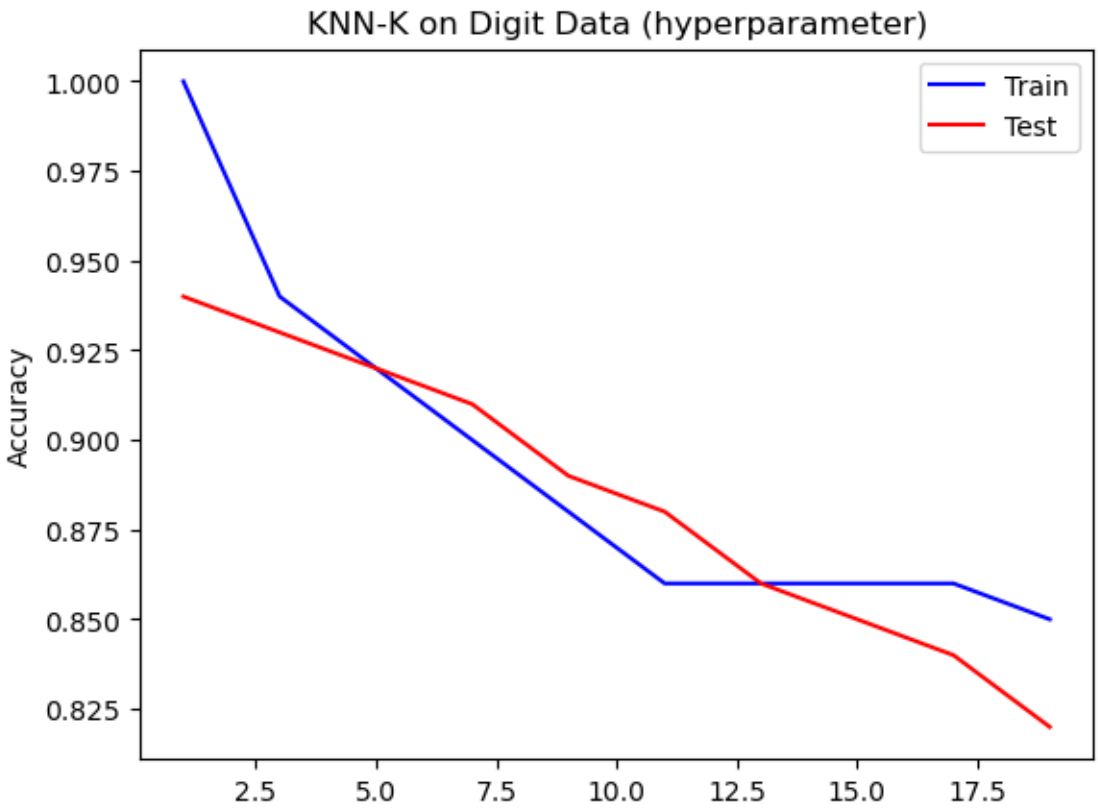
```
Training accuracy 0.92, test accuracy 0.92
```

3.1 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.

[49]:
```
curve = runClassifier.hyperparamCurveSet(knn.KNN({'isKNN':True}),
                                         'K', range(1,20,2), datasets.DigitData)
```

```
Training classifier with K=1…
Training accuracy 1, test accuracy 0.94
Training classifier with K=3…
Training accuracy 0.94, test accuracy 0.93
Training classifier with K=5…
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=9…
Training accuracy 0.88, test accuracy 0.89
Training classifier with K=11…
Training accuracy 0.86, test accuracy 0.88
Training classifier with K=13…
Training accuracy 0.86, test accuracy 0.86
Training classifier with K=15…
Training accuracy 0.86, test accuracy 0.85
Training classifier with K=17…
Training accuracy 0.86, test accuracy 0.84
Training classifier with K=19…
Training accuracy 0.85, test accuracy 0.82
```

[50]:
```
runClassifier.plotCurve('KNN-K on Digit Data (hyperparameter)', curve)
```



There is no extreme cases of underfitting nor overfitting, since both train and test predictions are fairly good in these ranges of K (greater than 80% accuracy). However, one could consider that there is some slight overfitting/underfitting due to the below reasoning.

The model is slightly underfitting as k increases past 9 because both training and test accuracy decrease, i.e. the model doesn't learn enough from the training set so it can't generalize as well on neither the train nor test set. It's even the case that sometimes the test accuracy is better than the training accuracy, which could be a sign that the model is underfit and getting lucky on the test set.

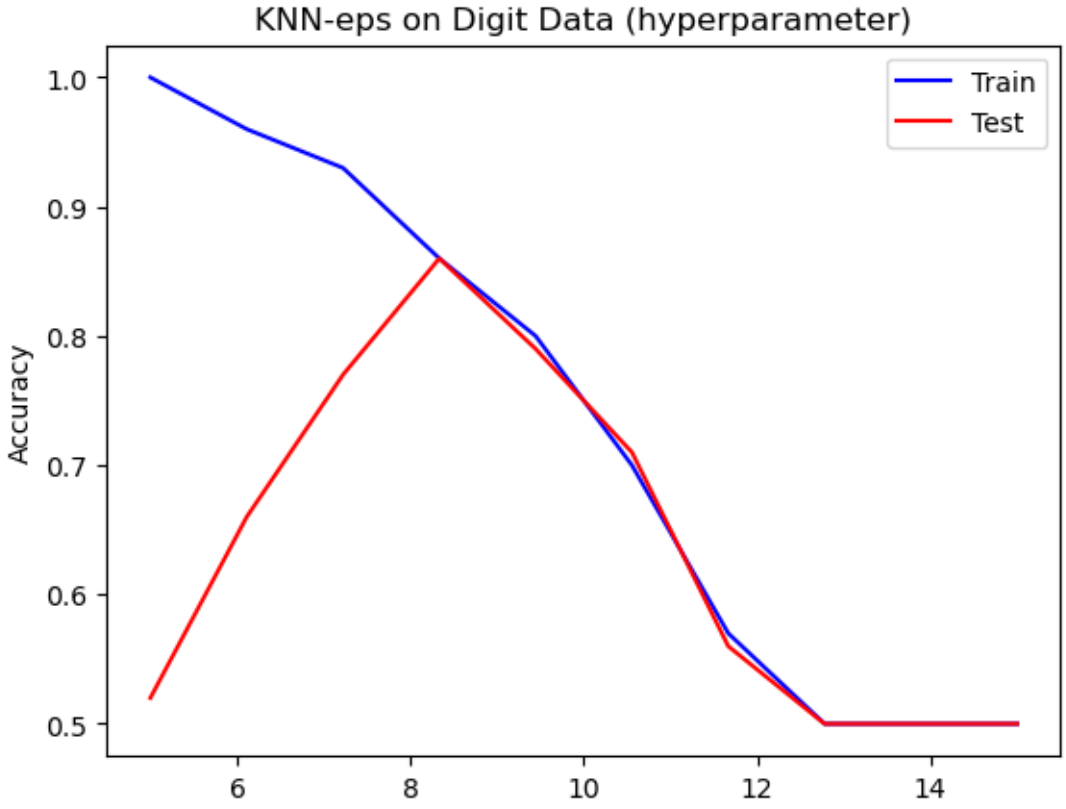
Also, there seems to be evidence of slight overfitting when $K = 1$. When $K = 1$, which is its smallest possible value, the train accuracy is perfect but the test accuracy is much lower (greater than 5% difference in accuracy).

[51]:
```
curve = runClassifier.hyperparamCurveSet(knn.KNN({'isKNN':False}),
                                         'eps', linspace(5,15,10), datasets.
↪DigitData)
```

```
Training classifier with eps=5…
Training accuracy 1, test accuracy 0.52
Training classifier with eps=6.11111…
Training accuracy 0.96, test accuracy 0.66
Training classifier with eps=7.22222…
```

```
Training accuracy 0.93, test accuracy 0.77
Training classifier with eps=8.33333…
Training accuracy 0.86, test accuracy 0.86
Training classifier with eps=9.44444…
Training accuracy 0.8, test accuracy 0.79
Training classifier with eps=10.5556…
Training accuracy 0.7, test accuracy 0.71
Training classifier with eps=11.6667…
Training accuracy 0.57, test accuracy 0.56
Training classifier with eps=12.7778…
Training accuracy 0.5, test accuracy 0.5
Training classifier with eps=13.8889…
Training accuracy 0.5, test accuracy 0.5
Training classifier with eps=15…
Training accuracy 0.5, test accuracy 0.5
```

[52]:
```
runClassifier.plotCurve('KNN-eps on Digit Data (hyperparameter)', curve)
```



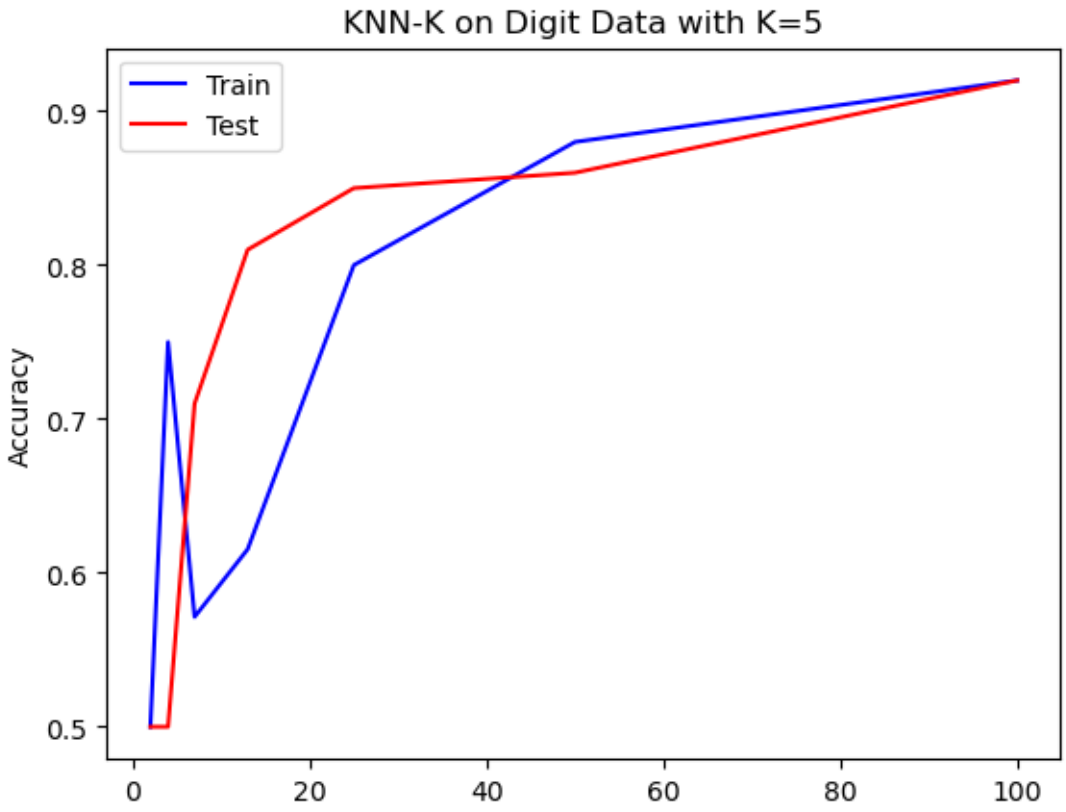
The model is clearly overfitting when eps < 8 because the training accuracy is much higher than the testing accuracy. This means when eps < 8 the models learns too much noise from the training data.

The model is clearly underfitting when eps > 10 because both the training accuracy and testing accuracy get low and decrease quickly. This means at eps > 10, the model doesn't learn enough from the training data, so it does poorly when tested on both sets.

[53]:
```
curve = 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
```

[54]:
```
runClassifier.plotCurve('KNN-K on Digit Data with K=5', curve)
```



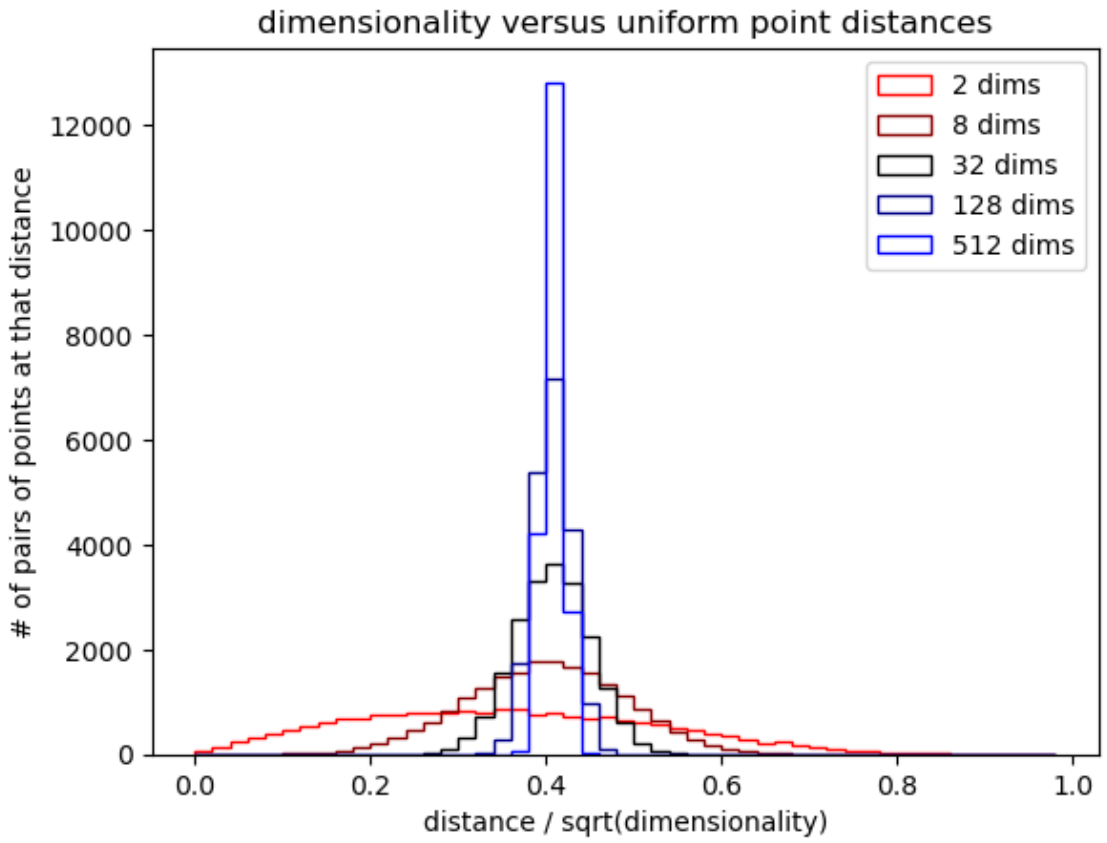
3.2 WU5:

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.

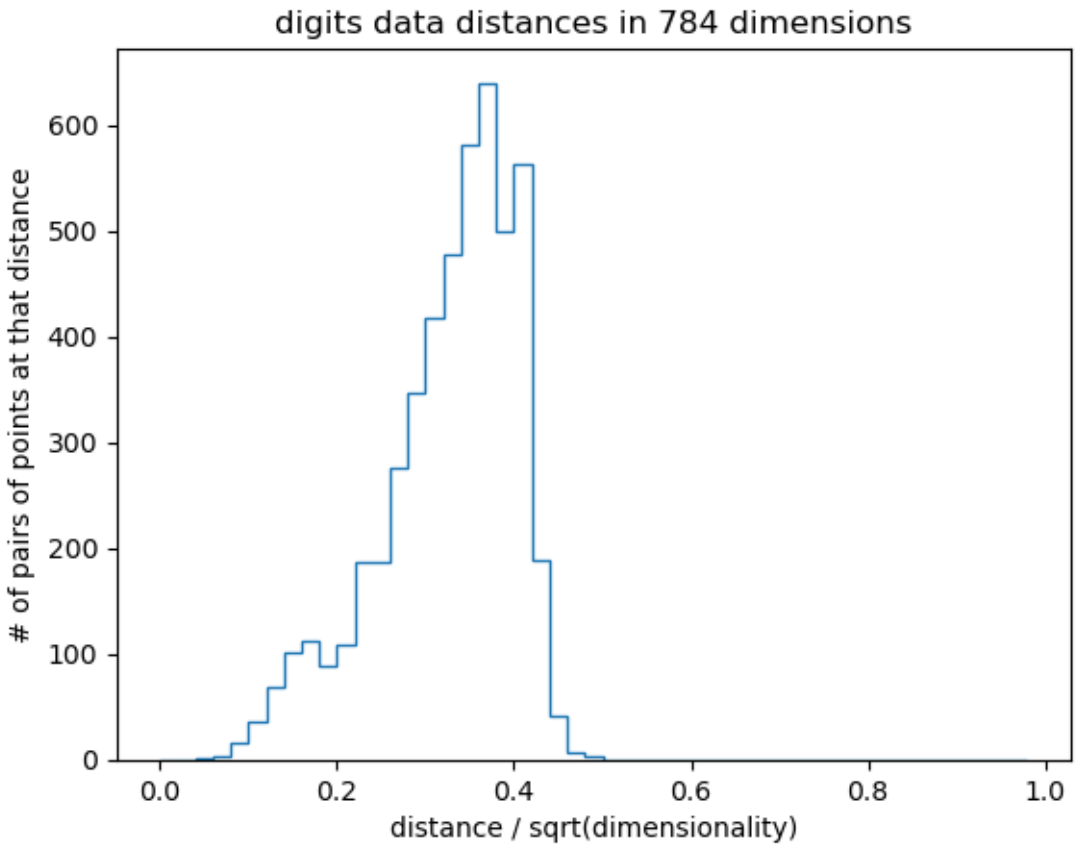
[55]:
```
from math import *
import random
from numpy import *
import matplotlib.pyplot as plt

waitForEnter=False

def generateUniformExample(numDim):
    return [random.random() for d in range(numDim)]

def generateUniformDataset(numDim, numEx):
    return [generateUniformExample(numDim) for n in range(numEx)]

def computeExampleDistance(x1, x2):
    dist = 0.0
    for d in range(len(x1)):
        dist += (x1[d] - x2[d]) * (x1[d] - x2[d])
    return sqrt(dist)

def computeDistances(data):
    N = len(data)
    D = len(data[0])
    dist = []
    for n in range(N):
        for m in range(n):
            dist.append( computeExampleDistance(data[n],data[m])  / sqrt(D))
    return dist

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(generateUniformDataset(d, N))
    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(False)
        x = raw_input('Press enter to continue...')


plt.legend(['%d dims' % d for d in Dims])
plt.savefig('fig.pdf')
plt.show()
```

```
D=2, average distance=0.502464
D=8, average distance=1.13112
D=32, average distance=2.30275
D=128, average distance=4.60219
D=512, average distance=9.24342
```



[56]:
```
plt.xlabel('distance / sqrt(dimensionality)')
plt.ylabel('# of pairs of points at that distance')
plt.title(f'digits data distances in {datasets.DigitData.X.shape[1]}␣
  ↪dimensions')
distances = computeDistances(datasets.DigitData.X)
plt.hist(distances, Bins, histtype='step')
plt.show()
```

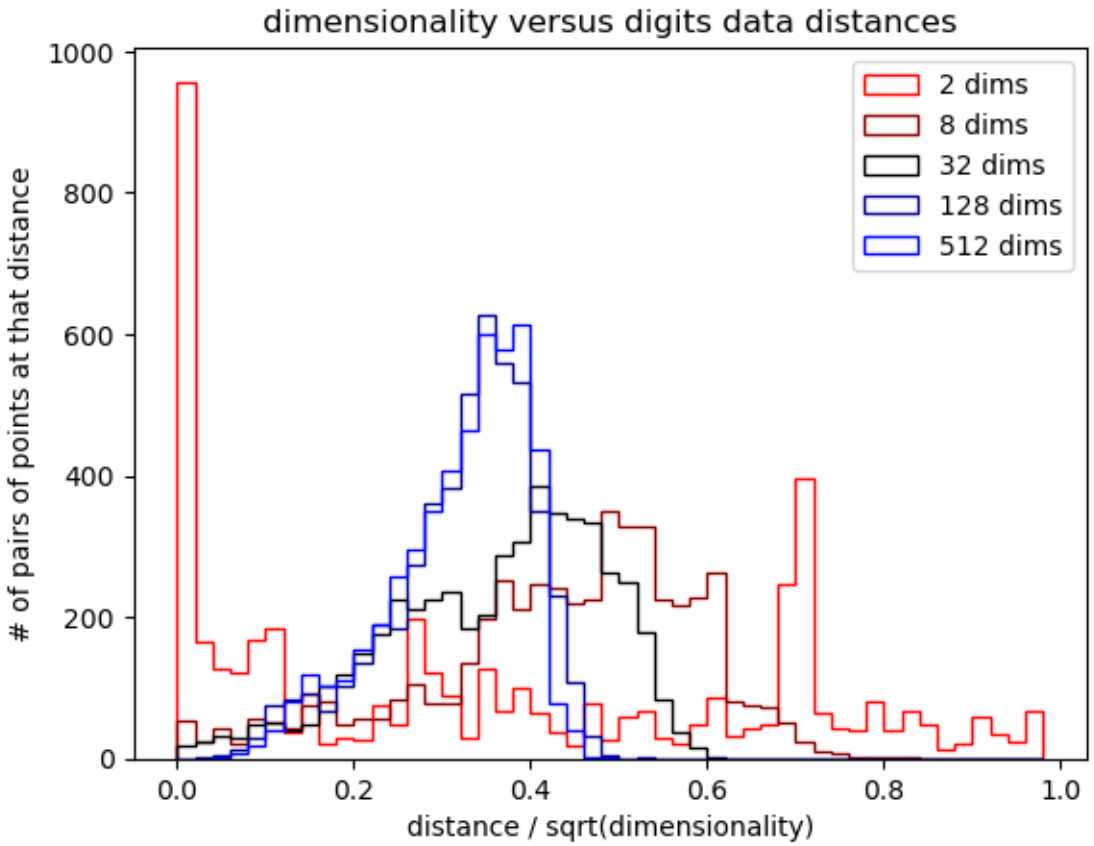


[57]:
```
def computeDistancesSubdims(data, d):
    N,D = data.shape
    indices = list(range(D))
    util.permute(indices)
    indices = indices[:d]
    distances = computeDistances(datasets.DigitData.X[:,indices])
    return distances
```

[58]:
```
plt.xlabel('distance / sqrt(dimensionality)')
plt.ylabel('# of pairs of points at that distance')
plt.title('dimensionality versus digits data distances')

for i,d in enumerate(Dims):
    distances = computeDistancesSubdims(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(False)
        x = raw_input('Press enter to continue...')


plt.legend(['%d dims' % d for d in Dims])
plt.show()
```

```
D=2, average distance=0.551648
D=8, average distance=1.22724
D=32, average distance=2.04882
D=128, average distance=3.64491
D=512, average distance=7.18375
```



The difference is that the HighD data is uniformly distributed, but the digits data isn't (as digit features are mostly 0's/sparse). As you can see, as dimensionality increases in the digits data, variance doesn't decrease substantially (except for maybe 2 to 8 dims) as the distances remain spread out, unlike in the HighD data. This means that the points don't get too similar in higher dimensions of the digits data which is good because it means KNN is a valid model.

Another difference is that in 2 dimensions the digits data has a strange bimodal distribution with

peaks at the extreme ends, unlike the HighD data which is always unimodal regardless of dimension. This may be due to the fact that the two features randomly picked in the digits data are either really similar across observations, or really different (no in between). Or perhaps more features are required to get meaningful distances.

However, both plots are similar in that distance grows proportionally to the sqrt of dimensionality. This can be seen as average distance increases with every increase in D in both data sets, and how both plots are able to shrink the distances between 0 and 1 by dividing distance by the sqrt of dimensionality.

4 Perceptron

[59]:
```
runClassifier.trainTestSet(perceptron.Perceptron({'numEpoch': 1}), datasets.
↪TennisData)
```

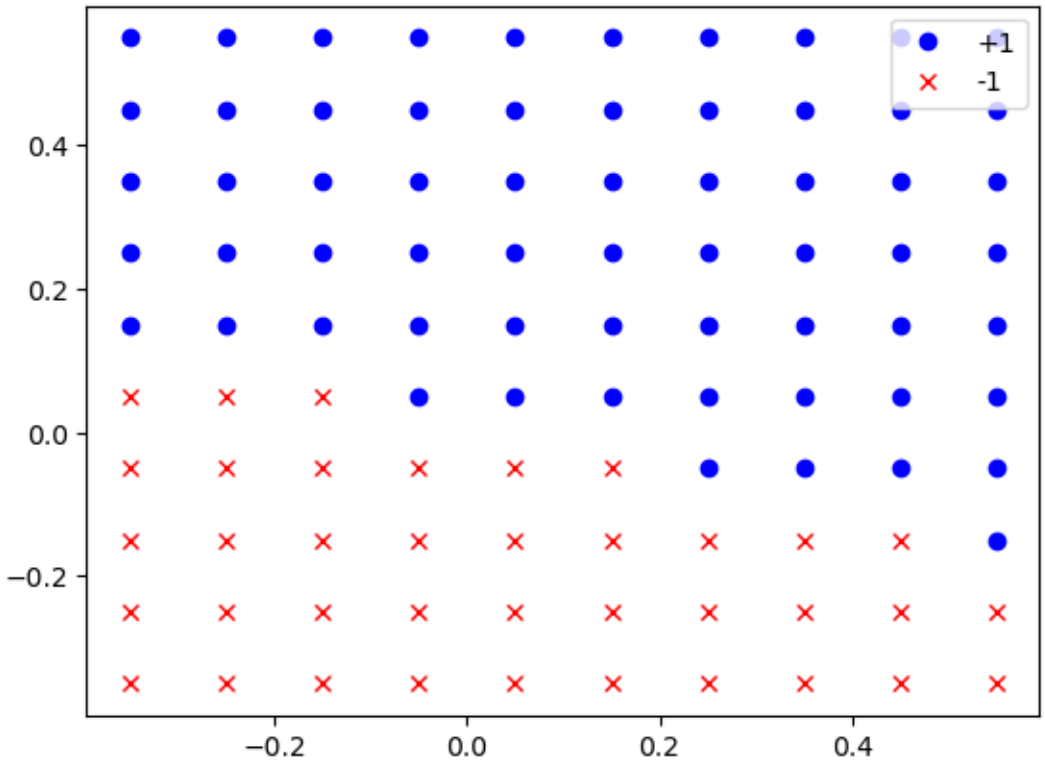
```
Training accuracy 0.642857, test accuracy 0.666667
```

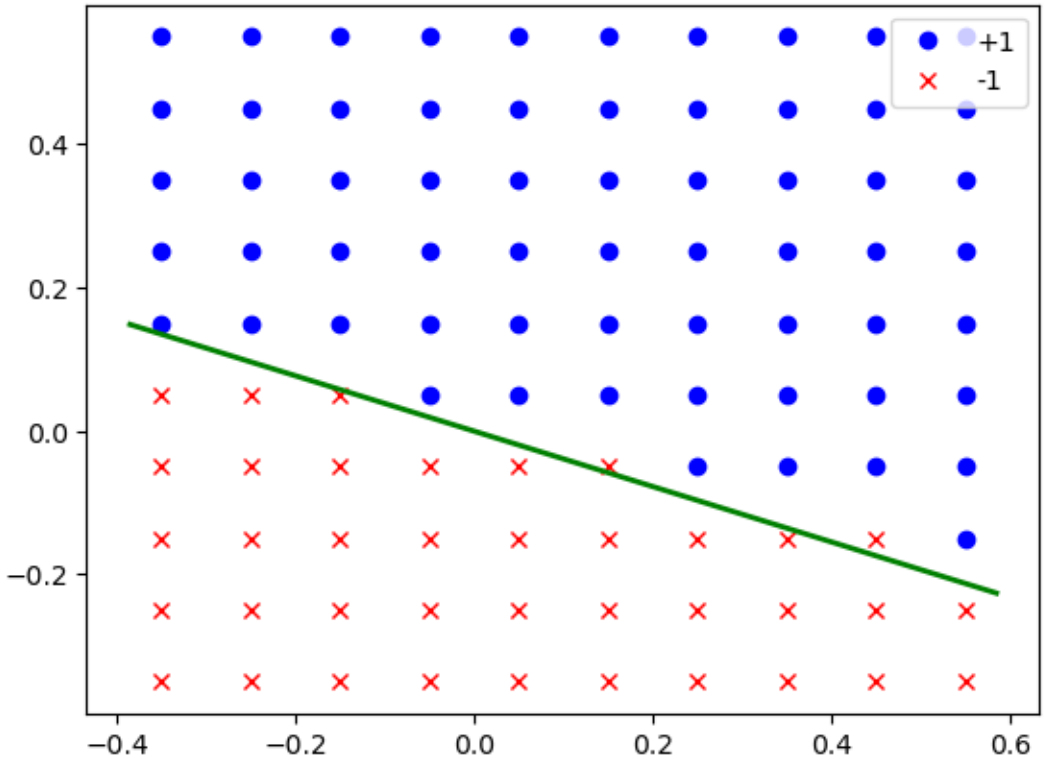
[60]:
```
runClassifier.trainTestSet(perceptron.Perceptron({'numEpoch': 2}), datasets.
↪TennisData)
```

```
Training accuracy 0.857143, test accuracy 1
```

[61]:
```
h = perceptron.Perceptron({'numEpoch': 200})
```

[62]:
```
runClassifier.plotData(datasets.TwoDDiagonal.X, datasets.TwoDDiagonal.Y)
runClassifier.plotClassifier(array([ 7.3, 18.9]), 0.0)
plt.close()
```


+1
-1
0.4
0.2
0.0
−0.2
−0.2
0.0
0.2
0.4


+1
-1
0.4
0.2
0.0
−0.2
−0.4
−0.2
0.0
0.2
0.4
0.6

[63]:
```
h.train(datasets.TwoDDiagonal.X, datasets.TwoDDiagonal.Y)
```

[64]:
```
h
```

[64]: w=array([7.3, 18.9]), b=0.0

[65]:
```
runClassifier.trainTestSet(perceptron.Perceptron({'numEpoch': 1}), datasets.
↪SentimentData)
```

```
Training accuracy 0.835833, test accuracy 0.755
```

[66]:
```
runClassifier.trainTestSet(perceptron.Perceptron({'numEpoch': 2}), datasets.
↪SentimentData)
```

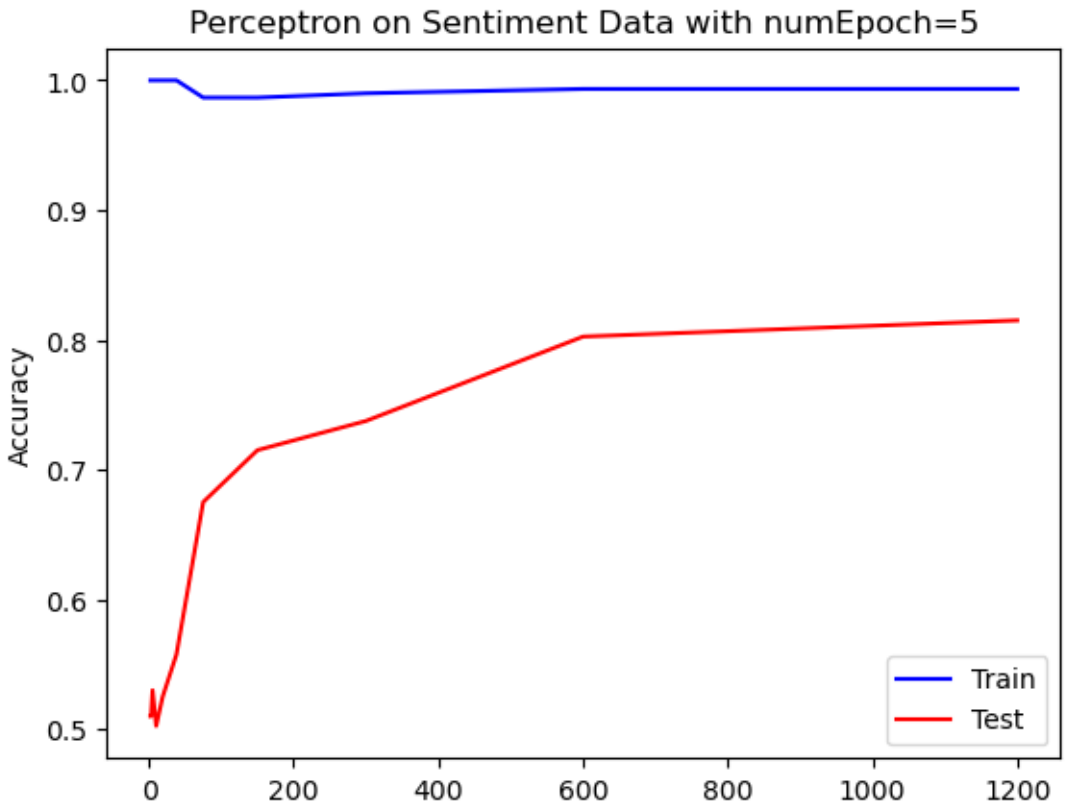
```
Training accuracy 0.955, test accuracy 0.7975
```

4.1 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.

[67]:
```
curve = 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
```

[68]:
```
runClassifier.plotCurve('Perceptron on Sentiment Data with numEpoch=5', curve)
```



[69]:
```
curve = runClassifier.hyperparamCurveSet(perceptron.Perceptron({}),
                                         'numEpoch', range(16), datasets.
 ↪SentimentData)
```

```
Training classifier with numEpoch=0…
Training accuracy 0.504167, test accuracy 0.5025
Training classifier with numEpoch=1…
Training accuracy 0.835833, test accuracy 0.755
Training classifier with numEpoch=2…
Training accuracy 0.955, test accuracy 0.7975
Training classifier with numEpoch=3…
Training accuracy 0.936667, test accuracy 0.755
Training classifier with numEpoch=4…
Training accuracy 0.995, test accuracy 0.8125
Training classifier with numEpoch=5…
Training accuracy 0.993333, test accuracy 0.815
Training classifier with numEpoch=6…
Training accuracy 0.994167, test accuracy 0.7875
Training classifier with numEpoch=7…
```

```
Training accuracy 0.993333, test accuracy 0.78
Training classifier with numEpoch=8…
Training accuracy 1, test accuracy 0.81
Training classifier with numEpoch=9…
Training accuracy 1, test accuracy 0.81
Training classifier with numEpoch=10…
Training accuracy 1, test accuracy 0.81
Training classifier with numEpoch=11…
Training accuracy 1, test accuracy 0.81
Training classifier with numEpoch=12…
Training accuracy 1, test accuracy 0.81
Training classifier with numEpoch=13…
Training accuracy 1, test accuracy 0.81
Training classifier with numEpoch=14…
Training accuracy 1, test accuracy 0.81
Training classifier with numEpoch=15…
Training accuracy 1, test accuracy 0.81
```

[70]:
```
runClassifier.plotCurve('Perceptron on Sentiment Data (hyperparameter)', curve)
```