CSI 873 Fall 2017 Final

Hand Written Digits Recognition Problem

Using The Support Vector Machine

Bruce Goldfeder

December 13, 2017

1 Final Exam Instructions

Implement your Support Vector Machine for the hand written digits recognition in MATLAB using available quadprog subroutine for solving a quadratic optimization problem. You can use any other programming language as long as it allows using a subroutine for solving a quadratic constrained optimization problems.

1. Use the provided discretized hand written digits data sets (both training and testing).

2. Formulate the dual soft margin SVM in MATLAB by specifying all the required matrices and vectors.

3. Train the dual soft-margin SVM (the one that incorporates a non-separable case) to classify 3s vs. 6s only. Select 500 training data points (250 for 3s, 250 for 6s). Use the dual radial basis function machine $\gamma = 0.05$. Use C = 100 as the penalty parameter. Increase if necessary.

4. Calculate the error/accuracy for testing examples.

5. Reduce the original number of pixels (784) uniformly by 50%, 75%, 90% and 95%. Calculate the testing accuracy for all the cases. Describe the observations.

6. Apply the SVD decomposition to the training data to prepare a lower quality data. Reduce the original 784 dimension by 50%, 75%, 90% and 95%. Calculate the testing accuracy for all the cases. Describe the observations.

7. Reduce the original number of number of training examples (500) by 50%, 75%, 90% and 95%. Calculate the testing accuracy for all the cases. Describe the observations.

8. Train the dual radial basis function machine $\gamma = 0.05$ to classify even vs. odd numbers. Select 1000 training data points (100 for each digit), use all 784 pixels. Calculate the error for testing examples.

9. Run 10 SVMs to train to detect a particular digit (e.g. 2) against the rest digits (e.g. 0, 1, 3, 4, 5, 6, 7, 8, 9). In the training use the value y = +1 for a particular digit and y = -1 for the rest of them. Obtain 10 different separating hyperplanes h0,. . . ,h9 that separate each 0,1,. . . ,9 from the rest digits.

10. While testing the digits you may find out that a particular digit may be classified not uniquely. For example, some tested digit can be on the positive side of h3, h5 and h8, meaning that this digit can be classified as 3, 5 or 8. Alternatively, you may find out that the tested digit is on negative side for all hyperplanes. To resolve the problem classify this digit as the one that corresponds to the hyperplane with the maximum classification number

$$\sum_{i=1}^{l} y_i \alpha_i^* K(x_i, x) - b$$

11. Calculate the error rate. Compare the results with those obtained for the artificial neural network, Bayes naïve classifier, and k-nearest neighbor algorithms

12. Document your experiments, prepare the report, submit it, and have a great holiday season!

2 Answers

I created a subsampling of the original data set to create a data set of 250 training examples and 250 test examples. The utility code to accomplish this is listed at the end of the report.

(1,2) I created the dual soft margin SVM using the MIT CVXOPT library (http://cvxopt.org/) Python Software for Convex Optimization. I constructed the matrices using a Gram matrix and providing the training label matrices using -1 and 1 values and creating the matrices as defined by Mathieu Blondel (the lead developer of the Scikit Learn SVM library for python (http://mblondel.org/)).
(3) I trained the SVM using the prescribed parameters of 250 3's, 250 6's, radial basis function machine $\gamma = 0.05$ and C = 100 as the penalty parameter.
For the results I will be applying a 95% confidence interval calculated using the formulas from chapter 5:

$$\sigma_{errors(h)} = \sqrt{\frac{p(1-p)}{n}}$$

```
(4) Results of SVM on full images:
For the full quality images:
   288 support vectors out of 500 points
   Full data set 491 out of 500 predictions correct
   Full data set Accuracy of  0.982
```

$$\sigma_{errors_s(h)} = \sqrt{\frac{.982(.018)}{500}} = .005945$$

Applying the formula

$$\mu \pm z_N\sigma; 1.96 \times .005945 = .0.01165$$

giving 0.982 $\pm$ .01165; a 95% confidence interval (ci95) of [.9704,.9937]

```
(5) Results of SVM using reduced quality images:
For the 50% image reduction
141 support vectors out of 500 points
   50% data set 496 out of 500 predictions correct
   50% data set Accuracy of  0.992
   ci95 is [ 0.9842 , 0.9998 ]

For the 75% image reduction
90 support vectors out of 500 points
   75% data set 487 out of 500 predictions correct
   75% data set Accuracy of  0.974
   ci95 is [ 0.9601 , 0.9879 ]

For the 90% image reduction
67 support vectors out of 500 points
   90% data set 480 out of 500 predictions correct
   90% data set Accuracy of  0.96
   ci95 is [ 0.9428 , 0.9772 ]

For the 95% image reduction
90 support vectors out of 500 points
   95% data set 464 out of 500 predictions correct
   95% data set Accuracy of  0.928
   ci95 is [ 0.9053 , 0.9507 ]
```

The full image came to an accuracy of 98.2%. An interesting observation is that the 50% reduced pixel size came to a higher accuracy of 99.2%. This can be do to the large amount of features/pixels that are not relevant to the categorization and can in fact reduce it slightly. At 75%,90%, and 95% the accuracy degrades very slightly with 95% reduction in features coming to an excellent 92.8% accuracy.

```
(6) Results on applying SVD decomposition to prepare lower quality data
SVD for 50 % reduction
   288 support vectors out of 500 points
   Full data set 491 out of 500 predictions correct
   Full data set Accuracy of  0.982
   ci95 is [ 0.9703 , 0.9937 ]

SVD for 75 % reduction
   266 support vectors out of 500 points
   Full data set 490 out of 500 predictions correct
   Full data set Accuracy of  0.98
   ci95 is [ 0.9677 , 0.9923 ]

SVD for 90 % reduction
   222 support vectors out of 500 points
   Full data set 487 out of 500 predictions correct
   Full data set Accuracy of  0.974
   ci95 is [ 0.9601 , 0.9879 ]

SVD for 95 % reduction
   165 support vectors out of 500 points
   Full data set 484 out of 500 predictions correct
   Full data set Accuracy of  0.968
   ci95 is [ 0.9526 , 0.9834 ]
```

For the SVD decomposition, the 50% reduction provided the same results as the full image. This is consistent in that the majority of the effective information is in the top 50% most important pixels. There was a more muted decay and a very good retention of categorization even at 95% reduction in the pixel amount giving a 96.8% accuracy.

```
(7) Results on reducing the number of samples:
For 50% reduction in samples:
   195 support vectors out of 250 points
   Full data set 235 out of 250 predictions correct
   Full data set Accuracy of  0.94
   ci95 is [ 0.9106 , 0.9694 ]

For 75% reduction in samples:
   115 support vectors out of 124 points
   Full data set 112 out of 124 predictions correct
   Full data set Accuracy of  0.903225806452
   ci95 is [ 0.8512 , 0.9553 ]

For 90% reduction in samples:
  50 support vectors out of 50 points
  Full data set 49 out of 50 predictions correct
  Full data set Accuracy of  0.98
  ci95 is [ 0.9412 , 1.0000 ]

For 95% reduction in samples:
  24 support vectors out of 24 points
  Full data set 24 out of 24 predictions correct
  Full data set Accuracy of  1.0
  ci95 is [ 1.0000 , 1.0000 ]
```

Reducing the number of samples to 250 and 124 reduced the accuracy at a larger rate than previous experiments on data quality dropping 6% and 10% respectively. However SVM is very good on very small sample sizes with no outliers and the accuracy for these small sample sizes came to 98% and 100%.

```
(8) Even versus Odd Results:
   789 support vectors out of 1000 points
   Full data set 945 out of 1000 predictions correct
   Full data set Accuracy of  0.945
   ci95 is [ 0.9309 , 0.9591 ]
```

The results of the Even versus Odd experiment were higher than I expected due to the conjunction of 5 differently shaped digits for each group. However, even with the small sample size of 100 images per digit the SVM performed at a 94.5% accuracy.

```
Zero versus Rest:
   384 support vectors out of 1000 points
   The number 0 versus Rest 963 out of 1000 predictions correct
   The number 0 versus Rest Accuracy of  0.963
   ci95 is [ 0.9513 , 0.9747 ]

One versus Rest:
   128 support vectors out of 1000 points
   The number 1 versus Rest 996 out of 1000 predictions correct
   The number 1 versus Rest Accuracy of  0.996
   ci95 is [ 0.9921 , 0.9999 ]

Two versus Rest:
   494 support vectors out of 1000 points
   The number 2 versus Rest 938 out of 1000 predictions correct
   The number 2 versus Rest Accuracy of  0.938
   ci95 is [ 0.9231 , 0.9529 ]

Three versus Rest:
   444 support vectors out of 1000 points
   The number 3 versus Rest 937 out of 1000 predictions correct
   The number 3 versus Rest Accuracy of  0.937
   ci95 is [ 0.9219 , 0.9521 ]

Four versus Rest:
   386 support vectors out of 1000 points
   The number 4 versus Rest 945 out of 1000 predictions correct
   The number 4 versus Rest Accuracy of  0.945
   ci95 is [ 0.9309 , 0.9591 ]

Five versus Rest:
   485 support vectors out of 1000 points
```

```
   The number 5 versus Rest 930 out of 1000 predictions correct
   The number 5 versus Rest Accuracy of  0.93
   ci95 is [ 0.9142 , 0.9458 ]

Six versus Rest:
   348 support vectors out of 1000 points
   The number 6 versus Rest 951 out of 1000 predictions correct
   The number 6 versus Rest Accuracy of  0.951
   ci95 is [ 0.9376 , 0.9644 ]

Seven versus Rest:
   301 support vectors out of 1000 points
   The number 7 versus Rest 961 out of 1000 predictions correct
   The number 7 versus Rest Accuracy of  0.961
   ci95 is [ 0.9490 , 0.9730 ]

Eight versus Rest:
   432 support vectors out of 1000 points
   The number 8 versus Rest 930 out of 1000 predictions correct
   The number 8 versus Rest Accuracy of  0.93
   ci95 is [ 0.9142 , 0.9458 ]

Nine versus Rest:
   344 support vectors out of 1000 points
   The number 9 versus Rest 951 out of 1000 predictions correct
   The number 9 versus Rest Accuracy of  0.951
   ci95 is [ 0.9376 , 0.9644 ]
```

The results were all greater than 93% which is excellent classification. The numbers 2,3,4,5, and 8 which have a lot of shape similarities performed on the lowest half. Very unique shapes like the number 1, came out very well.

```
(10) For testing each image against each hyperplane:
The MAX number 0 using all the hyperplanes max distance 98 out of 100 predictions correct
The MAX number 0 versus Rest Accuracy of  0.98
ci95 [ 0.9526 , 1.0000 ]
```

```
The MAX number 1 using all the hyperplanes max distance 99 out of 100 predictions correct
The MAX number 1 versus Rest Accuracy of  0.99
ci95 [ 0.9705 , 1.0000 ]

The MAX number 2 using all the hyperplanes max distance 89 out of 100 predictions correct
The MAX number 2 versus Rest Accuracy of  0.89
ci95 [ 0.8287 , 0.9513 ]

The MAX number 3 using all the hyperplanes max distance 80 out of 100 predictions correct
The MAX number 3 versus Rest Accuracy of  0.8
ci95 [ 0.7216 , 0.8784 ]

The MAX number 4 using all the hyperplanes max distance 86 out of 100 predictions correct
The MAX number 4 versus Rest Accuracy of  0.86
ci95 [ 0.7920 , 0.9280 ]

The MAX number 5 using all the hyperplanes max distance 87 out of 100 predictions correct
The MAX number 5 versus Rest Accuracy of  0.87
ci95 [ 0.8041 , 0.9359 ]

The MAX number 6 using all the hyperplanes max distance 88 out of 100 predictions correct
The MAX number 6 versus Rest Accuracy of  0.88
ci95 [ 0.8163 , 0.9437 ]

The MAX number 7 using all the hyperplanes max distance 89 out of 100 predictions correct
The MAX number 7 versus Rest Accuracy of  0.89
ci95 [ 0.8287 , 0.9513 ]

The MAX number 8 using all the hyperplanes max distance 81 out of 100 predictions correct
The MAX number 8 versus Rest Accuracy of  0.81
ci95 [ 0.7331 , 0.8869 ]

The MAX number 9 using all the hyperplanes max distance 89 out of 100 predictions correct
```

```
The MAX number 9 versus Rest Accuracy of  0.89
ci95 [ 0.8287 , 0.9513 ]
```

In order to determine this section I ran each of the test images through each of the hyperplanes resulting in an 1000 x 10 matrix of returns. The overall accuracy came to 88.6%. This seems reasonable as in the prior question we were constraining our decisions to a pure binary classifier. In this question we are opening this up to choose the best classifier out ten.

3 Comparison of Algorithms

The overall error rate for classification for any input using the last methodology came to 88.6% overall accuracy. This compares to the Artificial Neural Network which came to a 63.92% accuracy using 4 hidden nodes, the Bayes Naive Classifier which had an accuracy of 83.73, and K-Nearest Neighbor which came to 93.2% accuracy with k=7.

4 Running the Code

You will need to have the data set in a subfolder called 'data' and run python csi873Final_bgoldfeder.py The code is listed below:

```python
# -*- coding: utf-8 -*-
"""
Created on Wed Nov 29 18:06:07 2017

@author: bruce

CSI 873 Fall 2017 Dr. Griva

Final Exam

"""
import numpy as np
from numpy import linalg
import cvxopt
import cvxopt.solvers
import matplotlib.pyplot as plt
import os

def ReadInFiles(path,trnORtst):
    # This reads in all the files from a directory filtering on what the file
    # starts with
    fullData = []
    fnames = os.listdir(path)
```

```
    for fname in fnames:
        if fname.startswith(trnORtst):
            print (fname)
            data = np.loadtxt(path + "\\" + fname)
            fullData.append(data)
    #numFiles = len (fullData)
    #print(numFiles)

    return fullData

def ReadInOneList(fullData,maxRows):
    # This function combines all of the data into one array for ease of use
    # It contains a capping ability to configure how many results to use
    allData = []
    numFiles = len (fullData)
    for j in range (numFiles):
        # allows for smaller data set sizes
        numRows = len (fullData[j])
        #print('numrows,maxrows ',numRows,maxRows)
        if (maxRows < numRows):
            numRows = maxRows

        for k in range(numRows):
            allData.append(fullData[j][k])
    return np.asarray(allData)

def getData(dpath,trnNum,tstNum):

    # Read in the Training data first
    datasetTrn = ReadInFiles(dpath,'train')
    my_data = ReadInOneList(datasetTrn,trnNum)

    # Convert the 0-255 to 0 through 1 values in data
    my_data[:,1:] /= 255.0


    # randomize the rows for better training
    #np.random.shuffle(my_data)
    inNum,cols = my_data.shape
    just_trn_data = my_data[:,1:]
    answerTrn = my_data[:,0]

    # Read in the test data
    #dpath2 = os.getcwd()+'\data3'
    dataset2 = ReadInFiles(dpath,'test')
    my_test = ReadInOneList(dataset2,tstNum)

```

```
    tstNum,cols = my_test.shape
    #print('num rows ',tstNum)

    # Convert the 0-255 to 0 through 1 values in data
    my_test[:,1:] /= 255.0

    just_test_data = my_test[:,1:]
    answerTest = my_test[:,0]

    return just_trn_data,answerTrn,just_test_data,answerTest

def reduceDataQuality(dsize,just_trn_data,just_test_data):
    # 50% Reduced pixel data and label sets
    fiftyPtrnData = np.delete(just_trn_data, list(range(0, just_trn_data.shape[1], 2)), axis=1)
    fiftyPtstData = np.delete(just_test_data, list(range(0, just_test_data.shape[1], 2)), axis=1)

    # 75% Reduced pixel data and label sets
    seventyfivePtrnData = np.delete(fiftyPtrnData, list(range(0, fiftyPtrnData.shape[1], 2)), axis=1)
    seventyfivePtstData = np.delete(fiftyPtstData, list(range(0, fiftyPtstData.shape[1], 2)), axis=1)

    # 90% Reduced pixel data and label sets
    ninetyPtrnData = just_trn_data[:,::10].copy()
    ninetyPtstData = just_test_data[:,::10].copy()

    # 95% Reduced pixel data and label sets
    ninetyfivePtrnData = ninetyPtrnData[:,::2].copy()
    ninetyfivePtstData = ninetyPtstData[:,::2].copy()

    if dsize == 50:
        just_trn_data = fiftyPtrnData
        just_test_data = fiftyPtstData
    elif dsize == 75:
        just_trn_data = seventyfivePtrnData
        just_test_data = seventyfivePtstData
    elif dsize == 90:
        just_trn_data = ninetyPtrnData
        just_test_data = ninetyPtstData
    elif dsize == 95:
        just_trn_data = ninetyfivePtrnData
        just_test_data = ninetyfivePtstData

    return just_trn_data,just_test_data

```

```

def HeatMap(numberIn):
    #heat map to show numbers
    plt.matshow(numberIn.reshape(28,28))
    plt.colorbar()
    plt.show()

def linear_kernel(x1, x2):
    return np.dot(x1, x2)

def polynomial_kernel(x, y, p=3):
    return (1 + np.dot(x, y)) ** p

# radial basis function where gamm = -(1/(2*sigma**2))
def rbf(x, y, gamma):
    return np.exp(-1 * gamma * linalg.norm(x-y)**2 )

class SVM(object):
    def __init__(self, trnData, trnAns, tstData, tstAns,kernel=rbf, \
                C=None, gamma=.05, trnNum=250, tstNum=250):
        self.kernel = kernel
        self.C = C
        self.gamma = gamma
        if self.C is not None: self.C = float(self.C)
        self.trnNum = trnNum
        self.tstNum = trnNum
        self.trnData = trnData
        self.trnAns = trnAns
        self.tstData = tstData
        self.tstAns = tstAns

    def fit(self, X, y):
        n_samples, n_features = X.shape

        # Gram matrix For RBF using gamma
        K = np.zeros((n_samples, n_samples))
        for i in range(n_samples):
            for j in range(n_samples):
                K[i,j] = self.kernel(X[i], X[j],self.gamma)

        P = cvxopt.matrix(np.outer(y,y) * K)
        q = cvxopt.matrix(np.ones(n_samples) * -1)
        A = cvxopt.matrix(y, (1,n_samples))
        b = cvxopt.matrix(0.0)

        tmp1 = np.diag(np.ones(n_samples) * -1)
        tmp2 = np.identity(n_samples)
```

```
        G = cvxopt.matrix(np.vstack((tmp1, tmp2)))
        tmp1 = np.zeros(n_samples)
        tmp2 = np.ones(n_samples) * self.C
        h = cvxopt.matrix(np.hstack((tmp1, tmp2)))

        # solve QP problem
        cvxopt.solvers.options['show_progress'] = False
        solution = cvxopt.solvers.qp(P, q, G, h, A, b)

        # Lagrange multipliers
        a = np.ravel(solution['x'])
        #print("alphas? ",a[a>1])

        # Support vectors have non zero lagrange multipliers
        sv = a > 1e-1
        #sv = (a > 1e-2) & (self.C - a > 1e-2)
        ind = np.arange(len(a))[sv]
        self.a = a[sv]
        self.sv = X[sv]
        self.sv_y = y[sv]
        print("%d support vectors out of %d points" % (len(self.a), n_samples))

        # Intercept
        self.b = 0
        for n in range(len(self.a)):
            self.b += self.sv_y[n]
            inside_sum = self.a * self.sv_y * K[ind[n],sv]
            each_b = self.sv_y[n] - np.sum(inside_sum,axis=0)
            #print("for i_0 = ",ind[n]," and alpha_i0 = ",self.a[n]," b is ",each_b)
            #self.b -= np.sum(self.a * self.sv_y * K[ind[n],sv])
            self.b -= np.sum(inside_sum)

        self.b /= len(self.a)
        print("b is ",self.b)

    def project(self, X):
        # Calculates sigma a * sv_y * K(xi,sv)
        y_predict = np.zeros(len(X))
        for i in range(len(X)):
            s = 0
            for a, sv_y, sv in zip(self.a, self.sv_y, self.sv):
                s += a * sv_y * self.kernel(X[i], sv, self.gamma)
            y_predict[i] = s
        #print ("project return value is ",y_predict + self.b)
        return y_predict + self.b

    # Checks the sign of the projection in order to determine which side of
```

```
    # the decision boundary it is on
    def predict(self, X):
        return np.sign(self.project(X))

    def predict2(self, X):
        return self.project(X)

#todo This needs to be adapted to our 784 byte vectors and the binary
# Classifier e.g. 3's and 6's are 1 and -1 respectively

if __name__ == "__main__":


    def test_3v6(dset,trnData, trnAns, tstData, tstAns,trnN, tstN):

        # Adjust the data quality
        if dset != 100:
            trnData, tstData = reduceDataQuality(dset,trnData,tstData)

        # Test1 is the dual soft margin SVM to classify 3s vs 6s only
        # Get the training data for 250 3s and 250 6s
        test1 = SVM(trnData, trnAns, tstData, tstAns, \
                    kernel=rbf,C=100,gamma=.05, trnNum=trnN, tstNum=tstN)
        X_Train_3s = test1.trnData[test1.trnNum*3:(test1.trnNum*4)]
        y_Train_3s = test1.trnAns[test1.trnNum*3:(test1.trnNum*4)]

        X_Train_6s = test1.trnData[test1.trnNum*6:(test1.trnNum*7)]
        y_Train_6s = test1.trnAns[test1.trnNum*6:(test1.trnNum*7)]

        #HeatMap(X_Train_3s[0])
        #HeatMap(X_Train_3s[test1.trnNum-1])
        #HeatMap(X_Train_6s[0])
        #HeatMap(X_Train_6s[test1.trnNum-1])

        #print("first 3 ",y_Train_3s[0], " last 3 ",y_Train_3s[test1.trnNum-1])
        #print("first 6 ",y_Train_6s[0], " last 6 ",y_Train_6s[test1.trnNum-1])
        np.savetxt("y_Train_3s.txt",y_Train_3s)
        np.savetxt("y_Train_6s.txt",y_Train_6s)
        # Get the test data for 250 3s and 250 6s
        X_Test_3s = test1.tstData[test1.tstNum*3:(test1.tstNum*4)]
        y_Test_3s = test1.tstAns[test1.tstNum*3:(test1.tstNum*4)]

        X_Test_6s = test1.tstData[test1.tstNum*6:(test1.tstNum*7)]
        y_Test_6s = test1.tstAns[test1.tstNum*6:(test1.tstNum*7)]


        #HeatMap(X_Test_3s[0])
```

```
        #HeatMap(X_Test_3s[test1.tstNum-1])
        #HeatMap(X_Test_6s[0])
        #HeatMap(X_Test_6s[test1.tstNum-1])

        #print("first 3 ",y_Test_3s[0], " last 3 ",y_Test_3s[test1.tstNum-1])
        #print("first 6 ",y_Test_6s[0], " last 6 ",y_Test_6s[test1.tstNum-1])


        # The read in labels will be for data input checking only
        # I will convert the 3s labels to be -1 and
        # the 6s labels to be 1 for input into the SVM
        y_Train_3s = np.ones(test1.tstNum)
        y_Train_6s = np.ones(test1.tstNum) * -1

        X_train = np.vstack((X_Train_3s, X_Train_6s))
        y_train = np.hstack((y_Train_3s, y_Train_6s))

        y_Test_3s = np.ones(test1.tstNum)
        y_Test_6s = np.ones(test1.tstNum) * -1

        X_test = np.vstack((X_Test_3s, X_Test_6s))
        y_test = np.hstack((y_Test_3s, y_Test_6s))

        # Train the model using the full data set
        test1.fit(X_train, y_train)

        # Test model against the test data set
        y_predict = test1.predict(X_test)
        correct = np.sum(y_predict == y_test)
        print("Full data set %d out of %d predictions correct" % (correct, len(y_predict)))
        print("Full data set Accuracy of ",correct/len(y_predict))

    def test_EvO(trnData, trnAns, tstData, tstAns,trnN):

        # Test1 is the dual soft margin SVM to classify 3s vs 6s only
        # Get the training data for 250 3s and 250 6s
        test1 = SVM(trnData, trnAns, tstData, tstAns, \
                    kernel=rbf,C=100,gamma=.05, trnNum=trnN, tstNum=trnN)
        digitSize = 250
        start = 0
        end = 0
        trnEvenDataList = []
        trnOddDataList = []
        tstEvenDataList = []
        tstOddDataList = []
        for n in range(0,10):
            if n % 2 == 0: # even
```

```
                start = n*digitSize
                end = start + trnN
                print("n is ",n,"start",start,"end",end)
                d1 = trnData[start:end].tolist()
                print("d1 len",len(d1))
                for dx in d1:
                    trnEvenDataList.append(dx)

                print("dataListEvenTrain len is ",len(trnEvenDataList))
                HeatMap(np.asarray(trnEvenDataList[len(trnEvenDataList)-1]))

                print("n is ",n,"start",start,"end",end)
                d1 = tstData[start:end].tolist()
                print("d1 len",len(d1))
                for dx in d1:
                    tstEvenDataList.append(dx)

                print("dataListEvenTest len is ",len(tstEvenDataList))
                HeatMap(np.asarray(tstEvenDataList[len(tstEvenDataList)-1]))

            else:

                start = n*digitSize
                end = start + trnN
                print("n is ",n,"start",start,"end",end)
                d1 = trnData[start:end].tolist()
                print("d1 len",len(d1))
                for dx in d1:
                    trnOddDataList.append(dx)

                print("dataListODDTrain len is ",len(trnOddDataList))
                HeatMap(np.asarray(trnOddDataList[len(trnOddDataList)-1]))

                print("n is ",n,"start",start,"end",end)
                d1 = tstData[start:end].tolist()
                print("d1 len",len(d1))
                for dx in d1:
                    tstOddDataList.append(dx)

                print("dataListODDTest len is ",len(tstOddDataList))
                HeatMap(np.asarray(tstOddDataList[len(tstOddDataList)-1]))

        trnEvenData = np.asarray(tstEvenDataList)
        #print("shape of trainevendata",trnEvenData.shape)
        trnOddData = np.asarray(trnOddDataList)
        tstEvenData = np.asarray(tstEvenDataList)
        tstOddData = np.asarray(tstOddDataList)
```

```

        trnEvenLabs = np.ones(trnN*5)
        trnOddLabs = np.ones(trnN*5) * -1

        tstEvenLabs = np.ones(trnN*5)
        tstOddLabs = np.ones(trnN*5) * -1

        X_train = np.vstack((trnEvenData,trnOddData))
        print("shape of xtrain",X_train.shape)
        y_train = np.hstack((trnEvenLabs,trnOddLabs))

        X_test = np.vstack((tstEvenData,tstOddData))
        y_test = np.hstack((tstEvenLabs,tstOddLabs))

        ##### Test the input data for correctness ####
        HeatMap(X_train[0])
        HeatMap(X_train[499])
        HeatMap(X_train[500])
        HeatMap(X_train[999])

        #print("first 3 ",y_Train_3s[0], " last 3 ",y_Train_3s[test1.trnNum-1])
        #print("first 6 ",y_Train_6s[0], " last 6 ",y_Train_6s[test1.trnNum-1])
        np.savetxt("labels4evenodd.txt",y_train)


        # Train the model using the odds and even sets of numbers
        test1.fit(X_train,y_train)

        # Test model against the test data set
        y_predict = test1.predict(X_test)
        correct = np.sum(y_predict == y_test)
        print("Full data set %d out of %d predictions correct" % (correct, len(y_predict)))
        print("Full data set Accuracy of ",correct/len(y_predict))


    def test_OnevAll(trnData, trnAns, tstData, tstAns,trnN):

        # This is the dual soft margin SVM to classify One versus All
        # for all of the 10 numbers, resulting in 10 different
        # hyperplanes.
        # Get the training data of 100 for each number
        test1 = SVM(trnData, trnAns, tstData, tstAns, \
                    kernel=rbf,C=100,gamma=.05, trnNum=trnN, tstNum=trnN)
        digitSize = 250
        start = 0
        end = 0
        trnDataList = []
```

```
        tstDataList = []
        for n in range(0,10):

            start = n*digitSize
            end = start + trnN
            #print("n is ",n,"start",start,"end",end)
            d1 = trnData[start:end].tolist()
            #print("d1 len",len(d1))
            for dx in d1:
                trnDataList.append(dx)

            #print("dataListTrain len is ",len(trnDataList))
            #HeatMap(np.asarray(trnDataList[len(trnDataList)-1]))

            #print("n is ",n,"start",start,"end",end)
            d1 = tstData[start:end].tolist()
            #print("d1 len",len(d1))
            for dx in d1:
                tstDataList.append(dx)

        # The training and test data sets of 100 numbers each
        X_train = np.asarray(trnDataList)
        print("shape of traindata",X_train.shape)

        X_test = np.asarray(tstDataList)

        ##### Test the input data for correctness ####
        #HeatMap(X_train[0])
        #HeatMap(X_train[499])
        #HeatMap(X_train[500])
        #HeatMap(X_train[999])

        # Loop over the digits 0 - 9 and test versus the Rest
        ds = 100
        y_train = []
        y_test = []
        maxClassList = []
        for x in range(0,10):
            bottom = [-1] * (x*ds)
            oneLabels = [1] * ds
            top = [-1] * (1000 - (x*ds+ds))

            y_t = bottom + oneLabels + top

            print("len y_train",len(y_train))

            y_train = np.asarray(y_t)
```

```
            y_train = y_train.astype(np.double)
            np.savetxt("labels4oneVrest.txt",y_train)

            y_test = np.asarray(y_t)
            y_test = y_test.astype(np.double)
            print("shape of X",X_train.shape,"y",y_train.shape)

            # Train the model using the odds and even sets of numbers
            test1.fit(X_train,y_train)

            # Test model against the test data set
            y_predict = test1.predict(X_test)
            correct = np.sum(y_predict == y_test)
            print("The number",str(x),"versus Rest %d out of %d predictions correct" % (correct, len(y_predict)))
            print("The number",str(x),"versus Rest Accuracy of ",correct/len(y_predict))

            # Find the maximum classification number for each test sample
            y_predict_max = test1.predict2(X_test)

            print("max array shape",y_predict_max.shape)
            maxClassList.append(y_predict_max)


        values = np.vstack(maxClassList)
        np.savetxt("values.txt",values)
        predictMax = np.argmax(values,axis=0)
        np.savetxt("max2.txt",predictMax)

        for x in range(0,10):
            correct = np.sum(predictMax[x*ds:((x+1)*ds)] == x)
            print("The MAX number",str(x),"using all the hyperplanes max distance %d out of %d predictions correct" % (correct, ((x+1)*ds)-x*ds))
            print("The MAX number",str(x),"versus Rest Accuracy of ",correct/ds)


######## First Get the full #######
        # dsize is the percent reduction number as an integer e.g. 75
        # 100 is the full image size

    #dpath = os.getcwd() + "\\data4\\"
    dpath = os.getcwd() + "\\data\\"

    trnNum=250
    tstNum=250
    dsize =100
    trnData, trnAns, tstData, tstAns = getData(dpath,trnNum,tstNum)
```

```

############### Reduced Pixel Quality Reduction ################
    ###########data destructed levels 50,75,90,95#######

    test_3v6(100,trnData, trnAns, tstData, tstAns,trnNum,tstNum)
    test_3v6(50,trnData, trnAns, tstData, tstAns,trnNum,tstNum)
    test_3v6(75,trnData, trnAns, tstData, tstAns,trnNum,tstNum)
    test_3v6(90,trnData, trnAns, tstData, tstAns,trnNum,tstNum)
    test_3v6(95,trnData, trnAns, tstData, tstAns,trnNum,tstNum)

################## SVD Reduced Quality Section #################

    U,D,Vt = linalg.svd(trnData,full_matrices=False)
    X_a = np.dot(np.dot(U, np.diag(D)), Vt)
    print(np.std(trnData), np.std(X_a), np.std(trnData - X_a))
    print(D.shape)
    #np.savetxt("svdFull.txt",D)
    #np.savetxt("svd50.txt",D)
    percentVals = [392,196,78,39]
    for p in percentVals:
        D[p:]=0
        X_bar50 = np.dot(np.dot(U, np.diag(D)), Vt)
        print("SVD for",round(1-p/784,2),"% reduction")
        test_3v6(100,X_bar50, trnAns, tstData, tstAns,trnNum,tstNum)

################## Reduced Number of Samples Section
    trnData_50 = trnData[::2,:].copy()
    tstData_50 = tstData[::2,:].copy()
    trnAns_50 = trnAns[::2].copy()
    tstAns_50 = tstAns[::2].copy()
    trnNum = 125
    tstNum = 125
    #print("trn50 ",trnData_50.shape[0]," tst50 ",tstData_50.shape[0])

    test_3v6(100,trnData_50, trnAns_50, tstData_50, tstAns_50,trnNum,tstNum)

    # this will result in 62 and 63 (odd/even) outputs
    # to even out to 62 images per number, I remove every 125th
    remove_even_63s = [0,125,250,375,500]

    r75 = trnData_50[::2,:].copy()
    s75 = tstData_50[::2,:].copy()
    trnData_75 = np.delete(r75,remove_even_63s,axis=0)
    tstData_75 = np.delete(s75,remove_even_63s,axis=0)
    trnAns_75 = np.delete(trnAns_50[::2],remove_even_63s,axis=0)
    tstAns_75 = np.delete(tstAns_50[::2],remove_even_63s,axis=0)
    #np.savetxt("r75.txt",trnAns_75)
```

```
    #np.savetxt("trnData_75.txt",tstAns_75)

    trnNum = 62
    tstNum = 62
    #print("trn75 ",trnData_75.shape[0]," tst75 ",tstData_75.shape[0])
    #print(tstAns[62*3]," ",tstAns[62*4])

    test_3v6(100,trnData_75, trnAns_75, tstData_75, tstAns_75,trnNum,tstNum)

    # The 90% test for number of examples
    trnData_90 = trnData[::10,:].copy()
    tstData_90 = tstData[::10,:].copy()

    trnAns_90 = trnAns[::10].copy()
    tstAns_90 = tstAns[::10].copy()
    #np.savetxt("trnData_90.txt",tstAns_90)

    trnNum = 25
    tstNum = 25

    test_3v6(100,trnData_90, trnAns_90, tstData_90, tstAns_90,trnNum,tstNum)

    # this will result in 12 and 13 (odd/even) outputs
    # to even out to 12 images per number, I remove every 25th
    remove_even_13s = [0,25,50,75,100]

    r95 = trnData_90[::2,:].copy()
    s95 = tstData_90[::2,:].copy()
    trnData_95 = np.delete(r95,remove_even_13s,axis=0)
    tstData_95 = np.delete(s95,remove_even_13s,axis=0)
    trnAns_95 = np.delete(trnAns_90[::2],remove_even_13s,axis=0)
    tstAns_95 = np.delete(tstAns_90[::2],remove_even_13s,axis=0)
    #np.savetxt("r95.txt",trnAns_95)
    #np.savetxt("trnData_95.txt",tstAns_95)

    trnNum = 12
    tstNum = 12
    #print("trn95 ",trnData_95.shape[0]," tst95 ",tstData_95.shape[0])

    test_3v6(100,trnData_95, trnAns_95, tstData_95, tstAns_95,trnNum,tstNum)

############# Odd versus Even Section ###########################

    trnN = 100
    test_EvO(trnData, trnAns, tstData, tstAns,trnN)

################# One versus All Section #########################
```

```

    trn = 100
    test_OnevAll(trnData, trnAns, tstData, tstAns,trnN)
```