CS5785_HW1_Writeup

September 13, 2017

1 Digit Recognizer

Group Members: Huajun Bai(hb364), Hao Zheng(hz466)

1.1 Q1a Loading Data

```
In [2]: # Importing library
        import collections
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import sklearn.metrics.pairwise as skl
        from scipy import stats
        from scipy.optimize import brentq
        from scipy.interpolate import interp1d
        from sklearn import linear_model, cross_validation, preprocessing, metrics
```

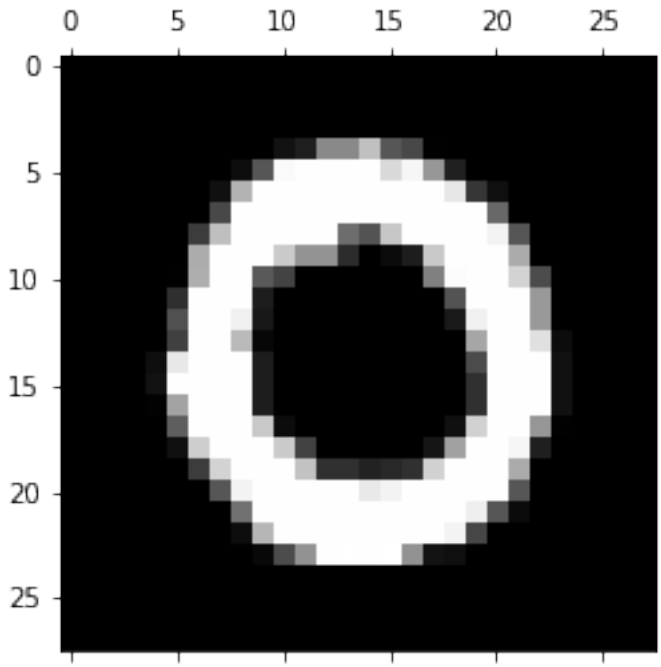
```
In [3]: # load data
        data =  np.loadtxt(fname = 'train.csv', delimiter = ',', skiprows=1)
        n,p = data.shape
```

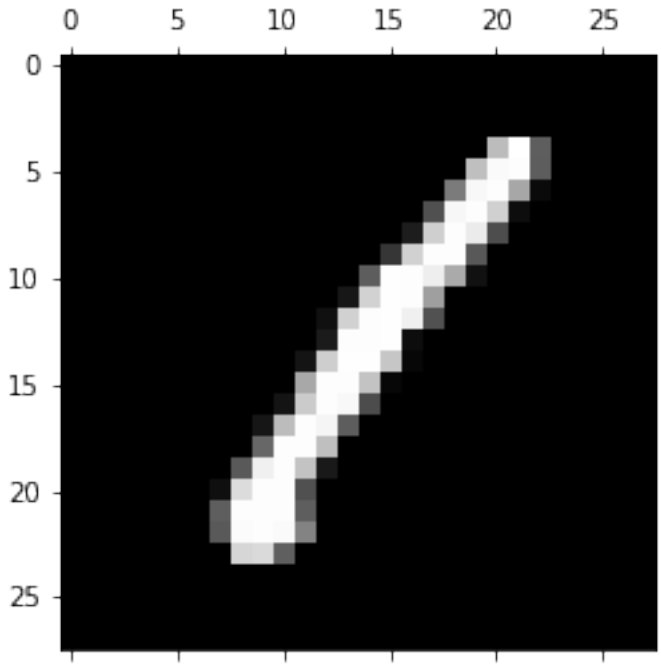
1.2 Q2b

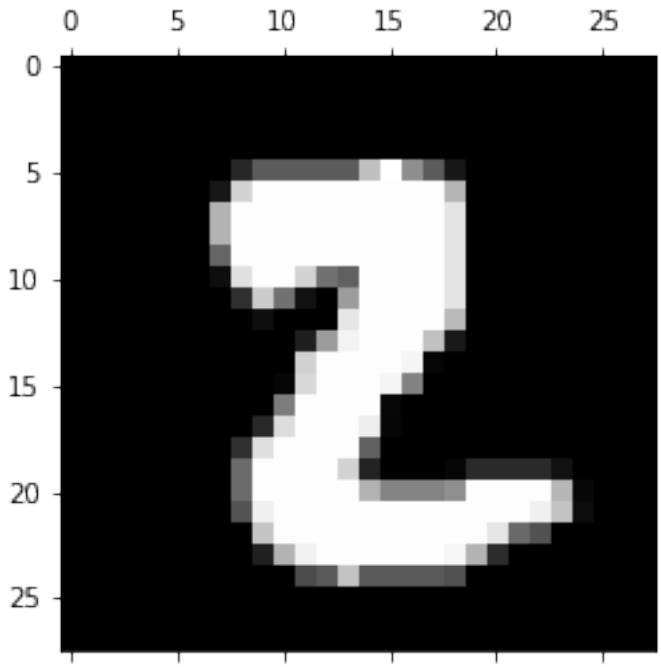
Below is the function to display any digit d. And a display of digits 0-9 in the data.

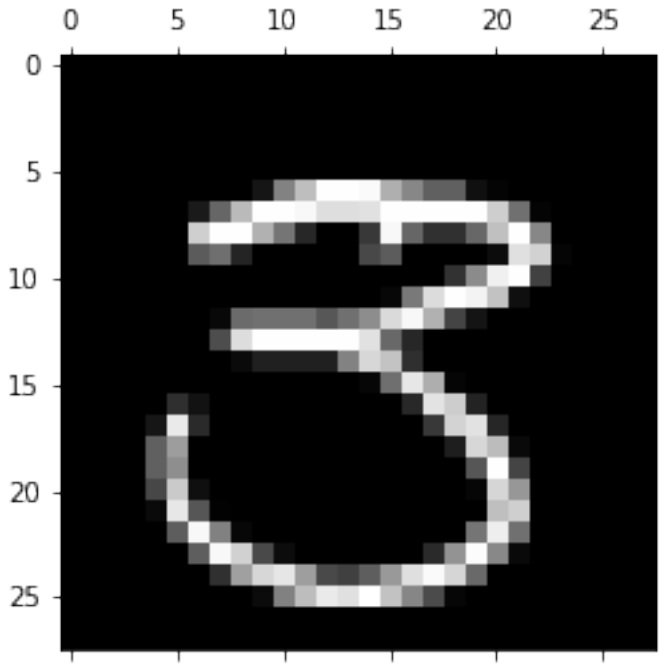
```
In [4]: # A function to display a digit d
        def display(data, d):
            for i in range(n):
                if data[i, 0] == d:
                    plt.matshow(data[i,1:785].reshape(28,28), cmap='gray')
                    plt.show()
                    break;
```

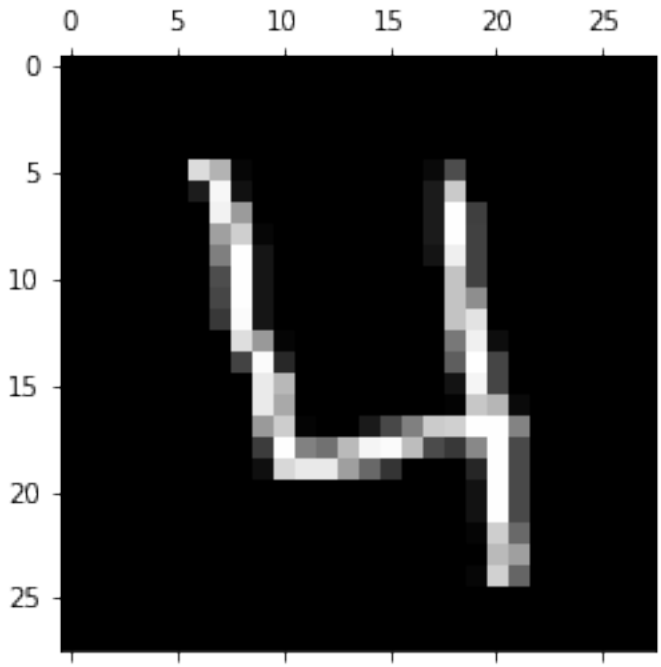
```
In [6]: for j in range(10):
            display(data, j)
```

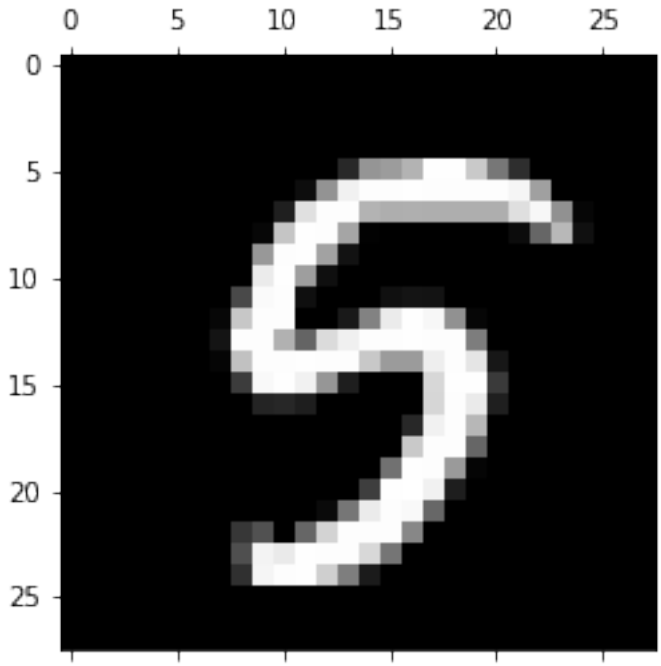

0
5
10
15
20
25

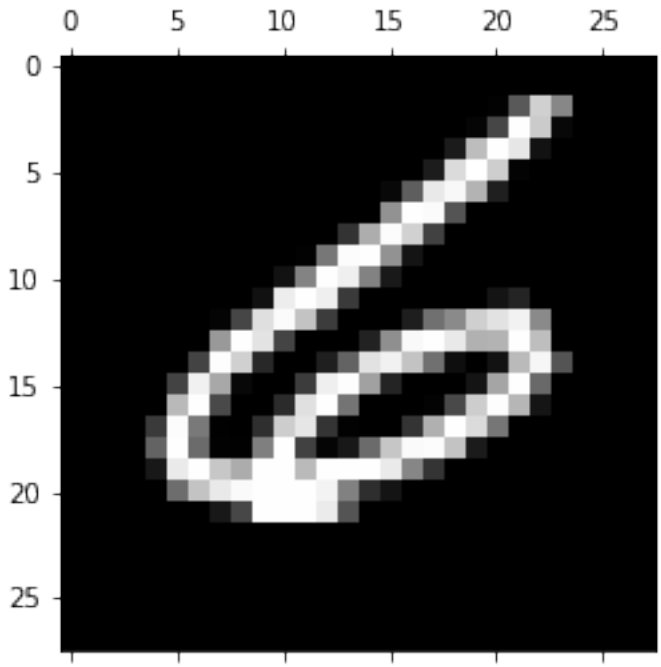

0
5
10
15
20
25

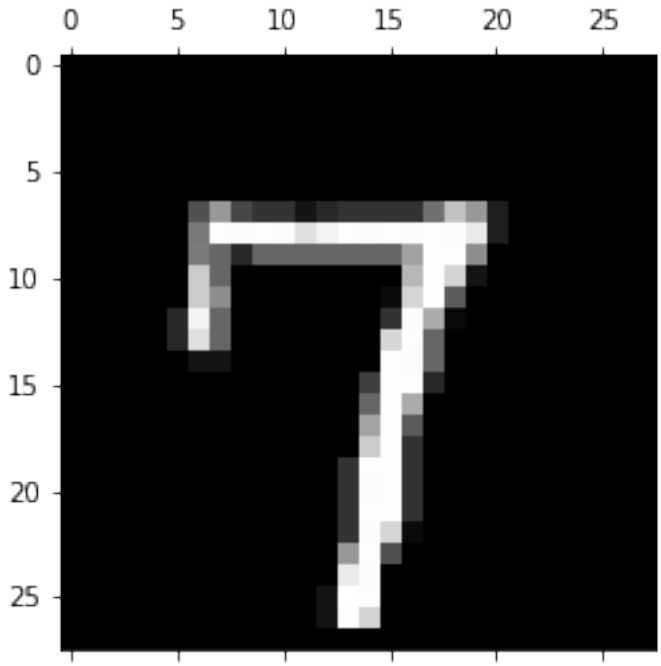

0
5
10
15
20
25
0
5
10
15
20
25

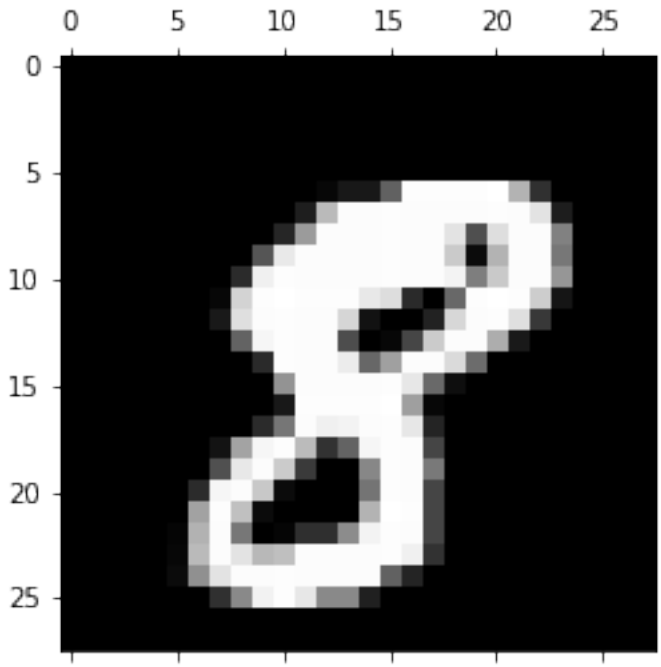

0
5
10
15
20
25
0
5
10
15
20
25

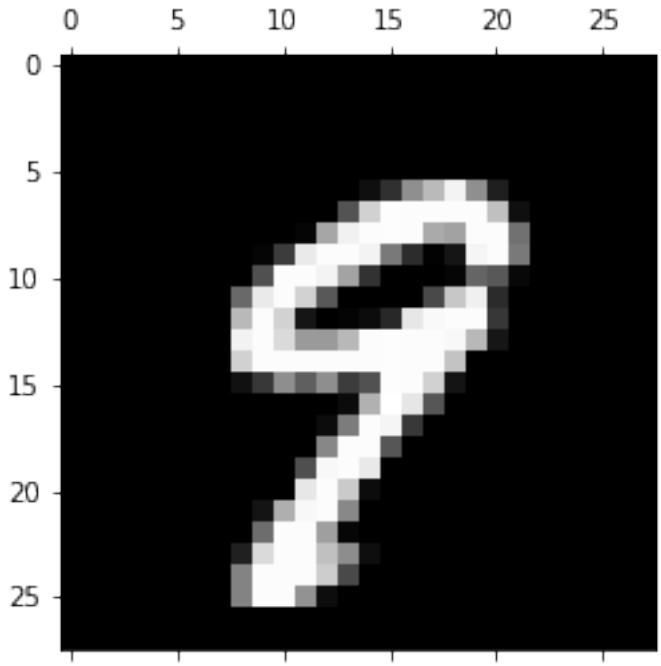

0
5
10
15
20
25


0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25

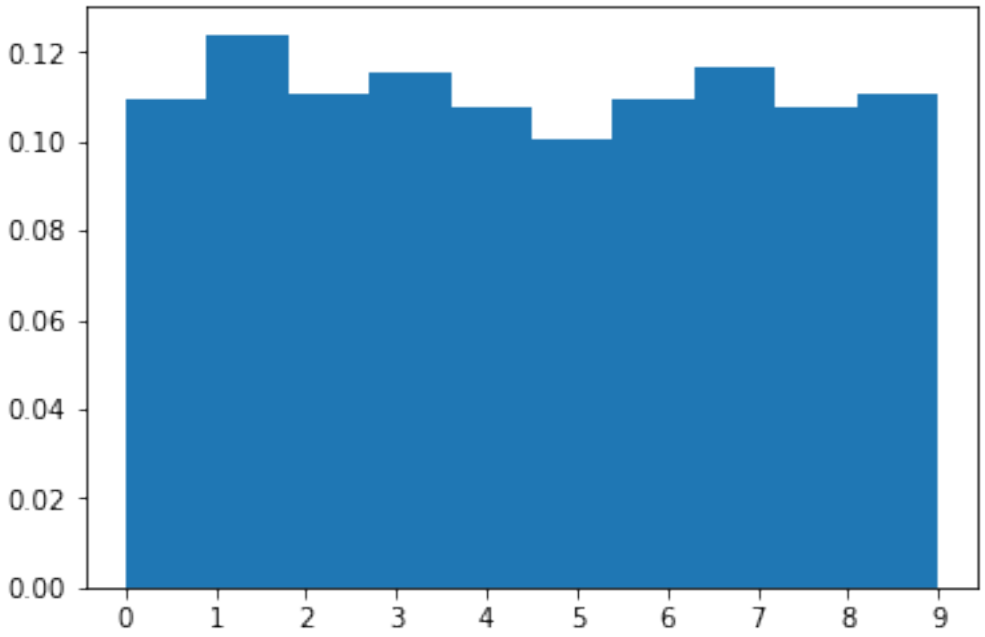
1.3 Q1c

Below is data for number of each digit in the data set. The distribution is not uniform, so the graph is not even.

```
In [165]: collections.Counter(data[:,0])

Out[165]: Counter({0.0: 4132,
                   1.0: 4684,
                   2.0: 4177,
                   3.0: 4351,
                   4.0: 4072,
                   5.0: 3795,
                   6.0: 4137,
                   7.0: 4401,
                   8.0: 4063,
                   9.0: 4188})
```

```
In [90]: plt.hist(data[:,0], normed = True, histtype = 'bar')
         plt.xticks(np.arange(10), ('0','1','2','3','4','5','6','7','8','9'))
         plt.show()
```



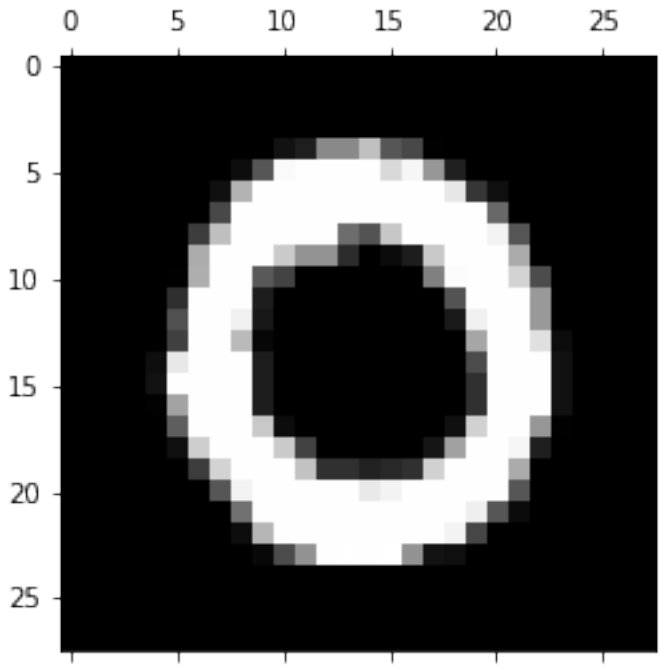
1.4 Q1d

Below you can find code and graphs for each digit and their nearest neighbor. No.3 is a outlier, where nearest neighbor outputs a '5' as its nearest neighbor.

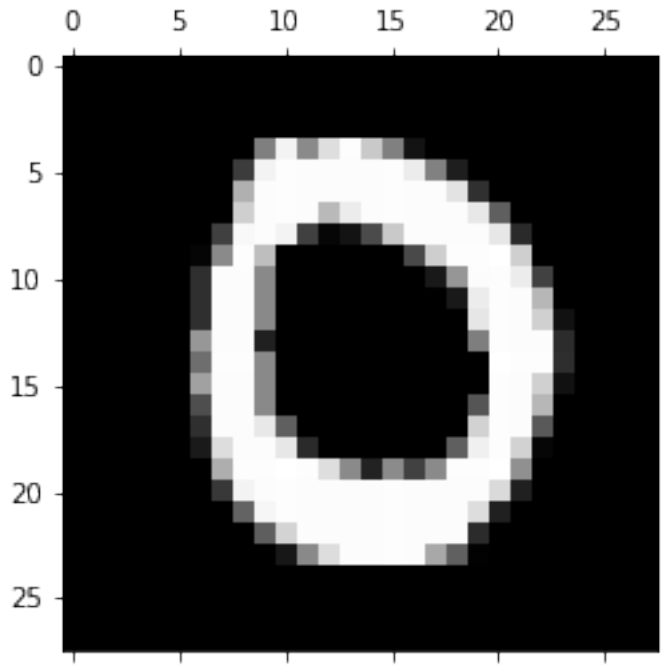
```
In [11]: def findNearest(index):
             diff = np.subtract(np.array(data[:, 1:785]), np.array(data[index, 1:785]));
             distM = np.square(diff) # delete index itself
             # Assume no same data set exist, i.e. no 0 dist except itself
             return np.argsort(np.sum(distM, axis=1)).item(1)

In [12]: # Part d
         # No.3 outlier
         for i in range(10):
             for j in range(n):
                 if(data[j,0] == i):
                     plt.matshow(data[j,1:785].reshape(28,28), cmap='gray')
                     print ("this digit is " + repr(i)+ " from No." + repr(j));
                     plt.show()
                     nearIndex = findNearest(j)
                     print ("Nearest digit is "+ repr(data[nearIndex, 0])+ " from No." + repr(ne
                     plt.matshow(data[nearIndex,1:785].reshape(28,28), cmap='gray')
                     plt.show()
                     break
```

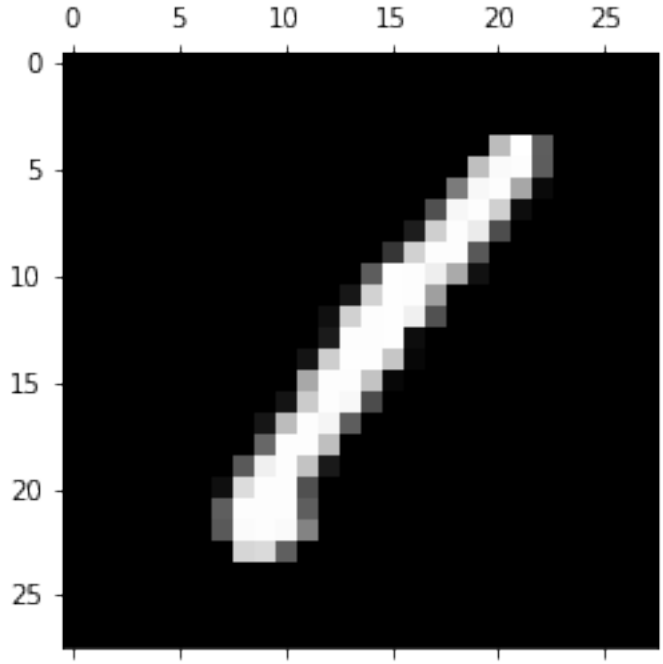
```
this digit is 0 from No.1
```



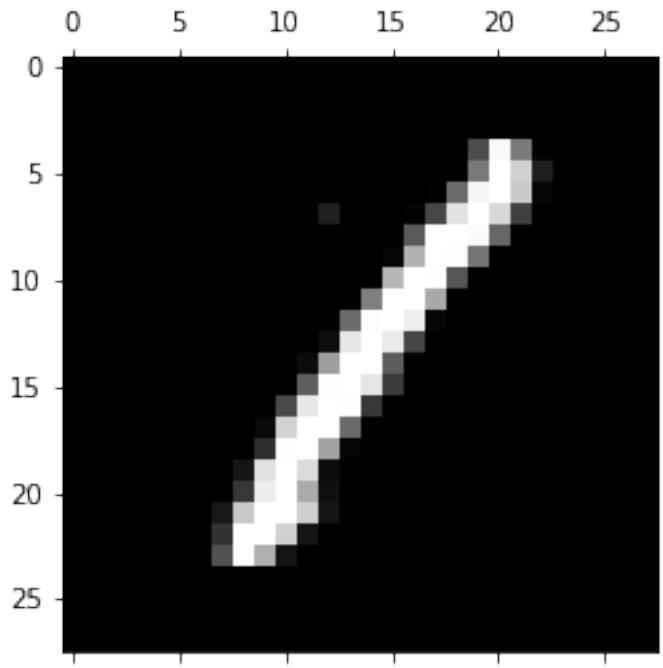

```
Nearest digit is 0.0 from No.12950
```



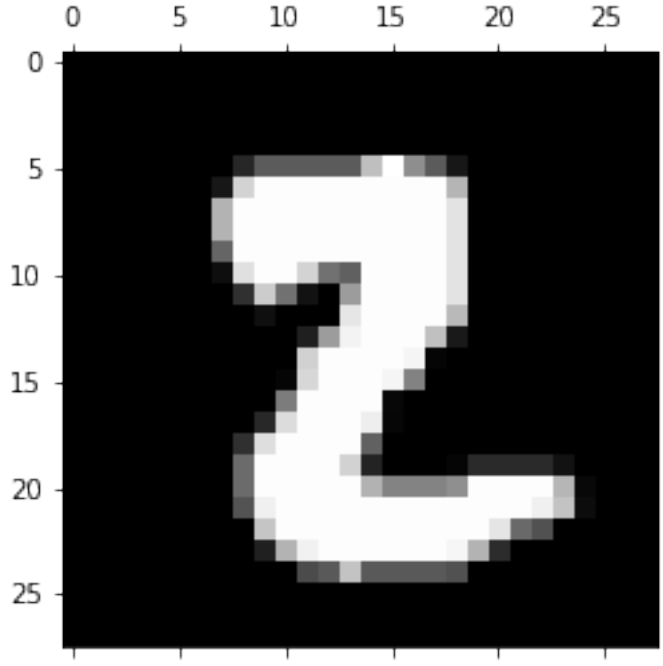

```
this digit is 1 from No.0
```



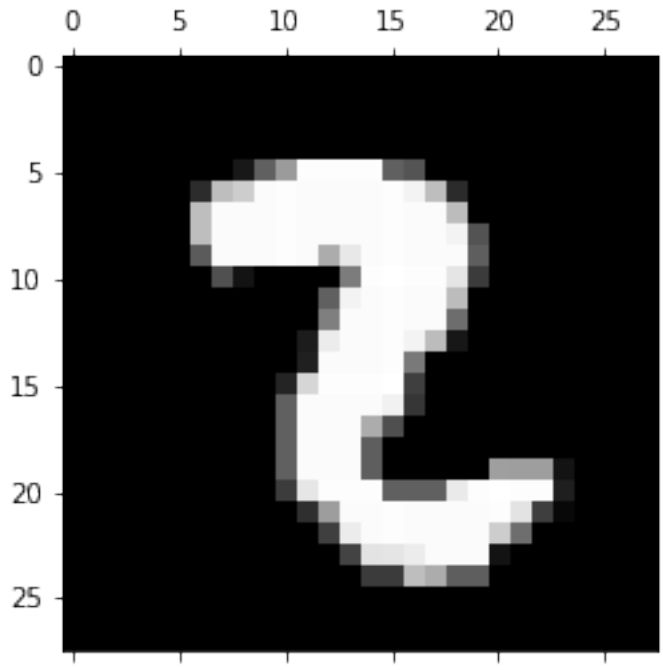
```
Nearest digit is 1.0 from No.29704
```



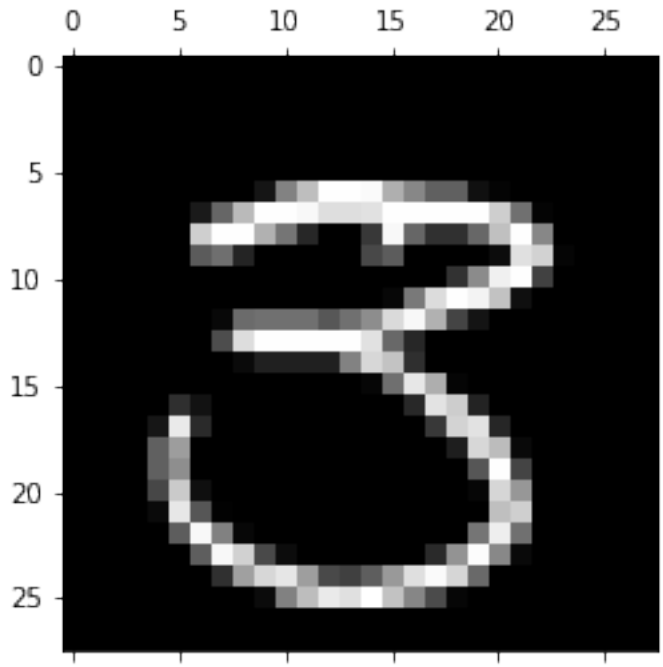

```
this digit is 2 from No.16
```



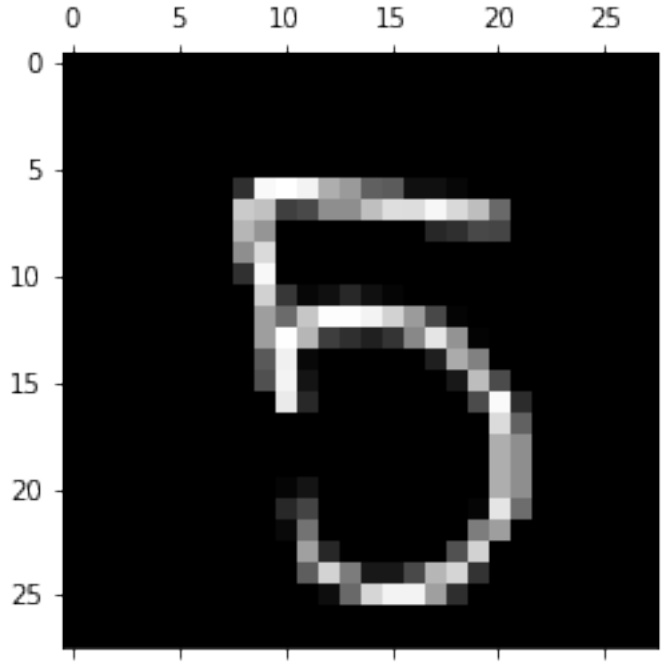
```
Nearest digit is 2.0 from No.9536
```



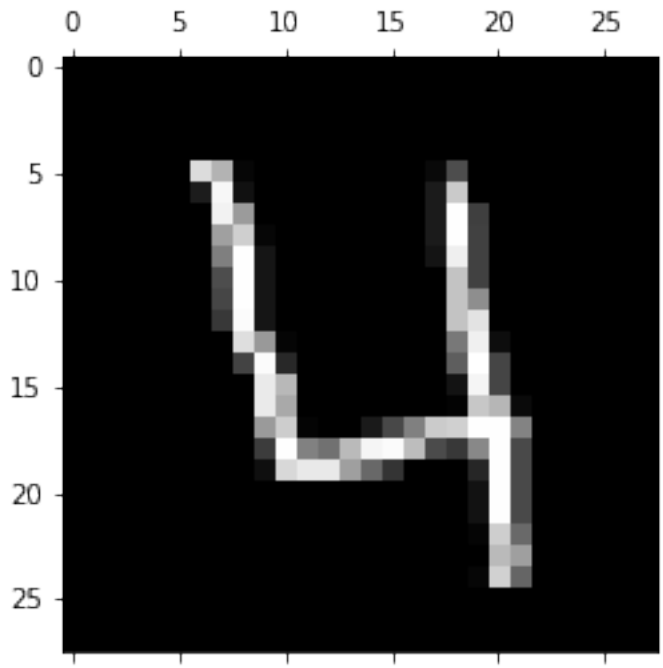

```
this digit is 3 from No.7
```



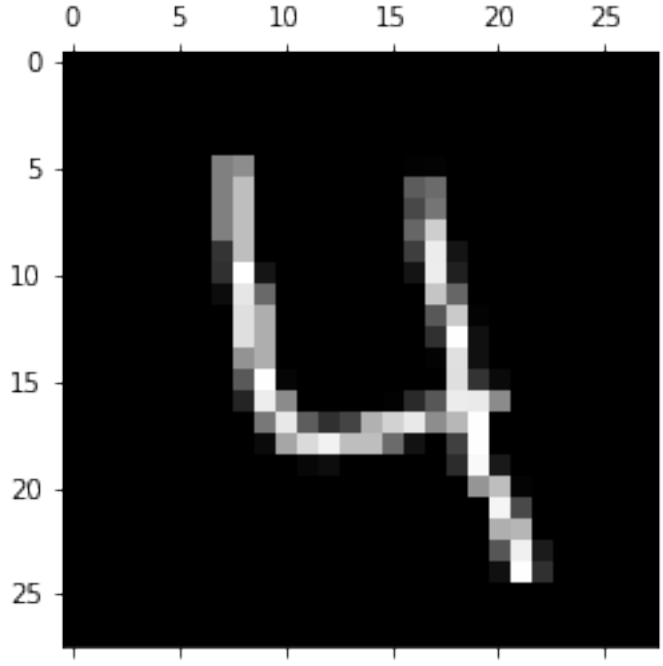

```
Nearest digit is 5.0 from No.8981
```



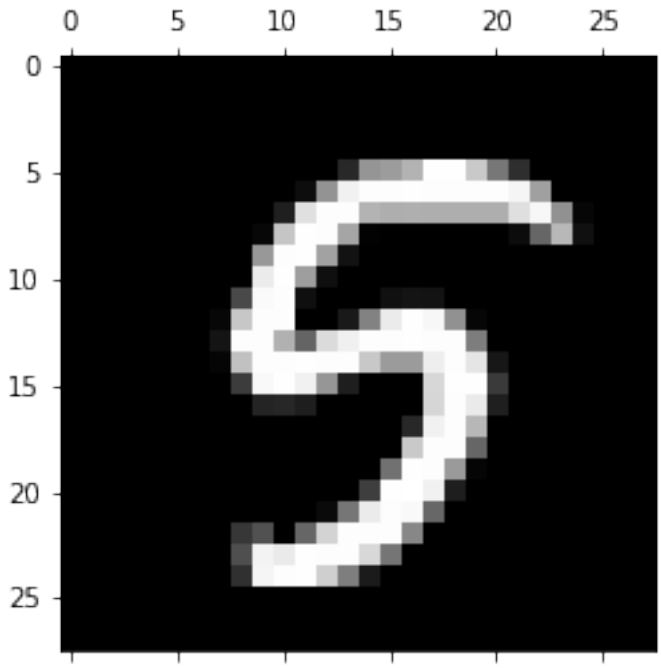
```
this digit is 4 from No.3
```



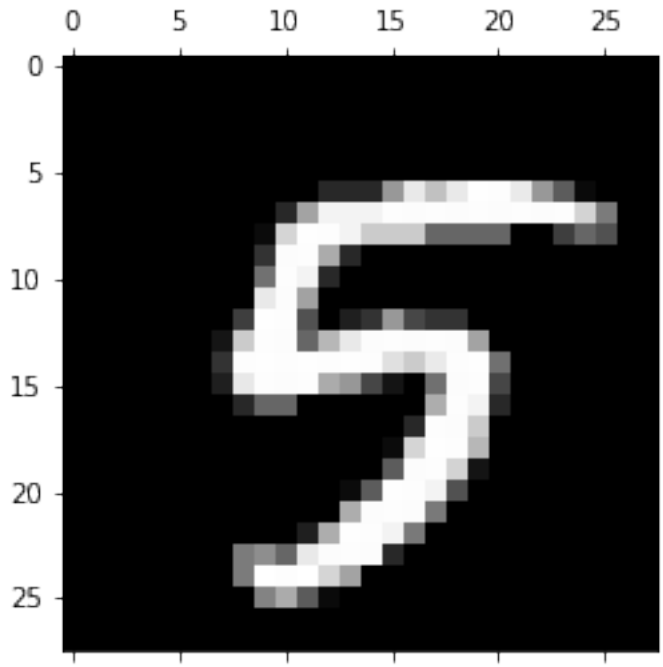

```
Nearest digit is 4.0 from No.14787
```



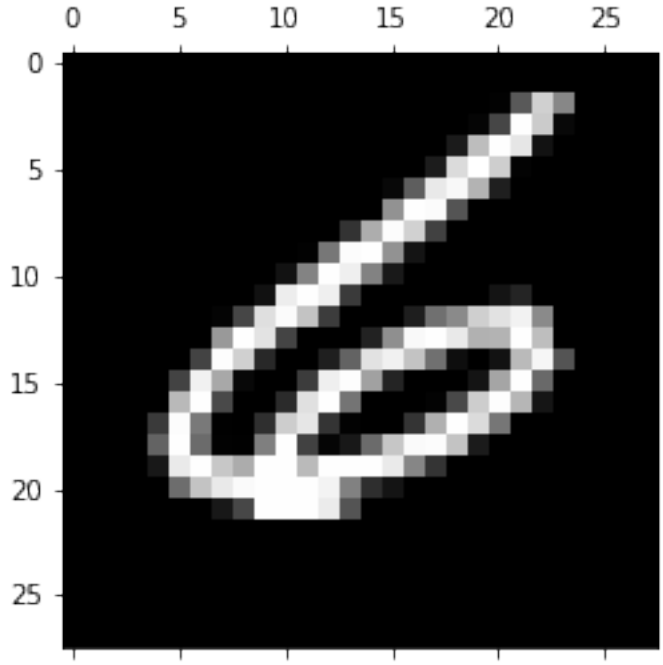
```
this digit is 5 from No.8
```



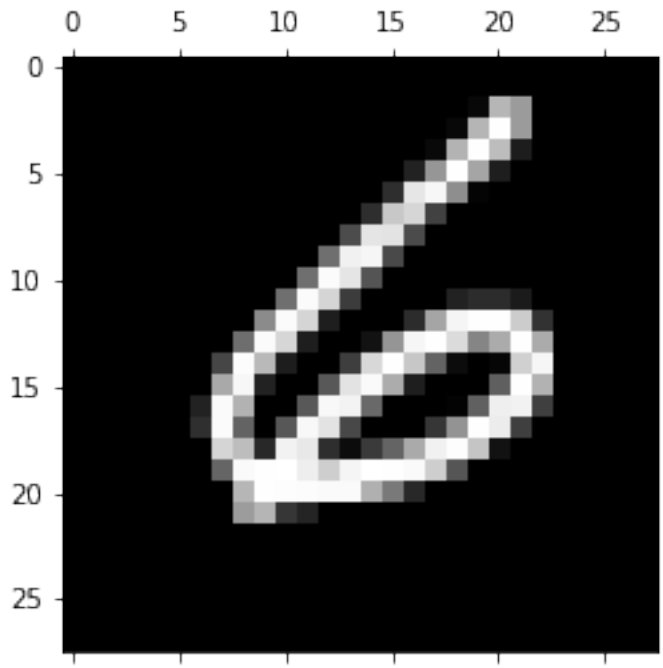

```
Nearest digit is 5.0 from No.30073
```



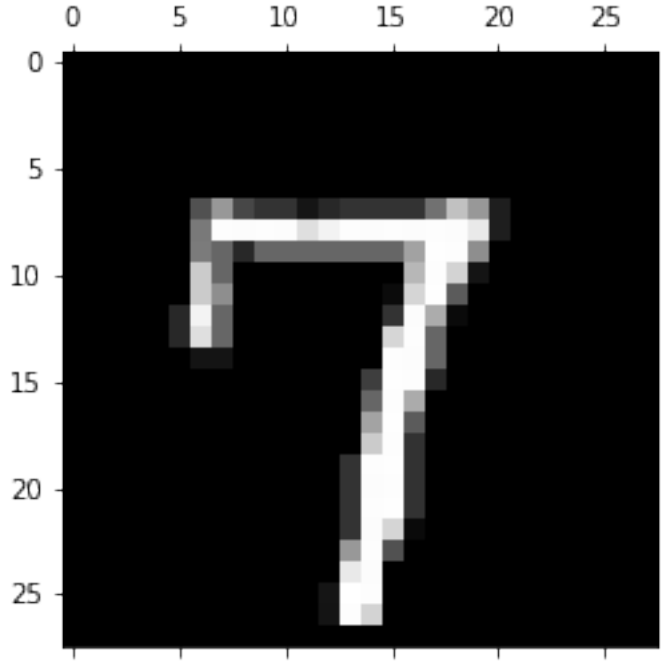

```
this digit is 6 from No.21
```



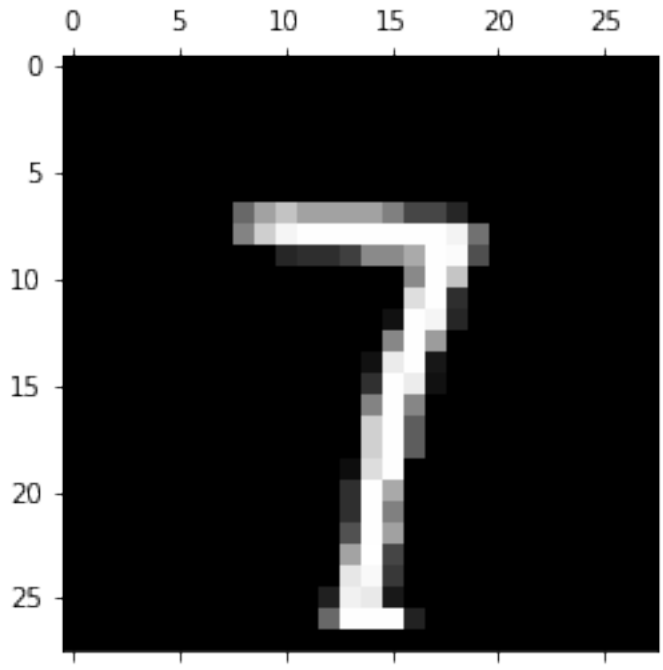
```
Nearest digit is 6.0 from No.16240
```



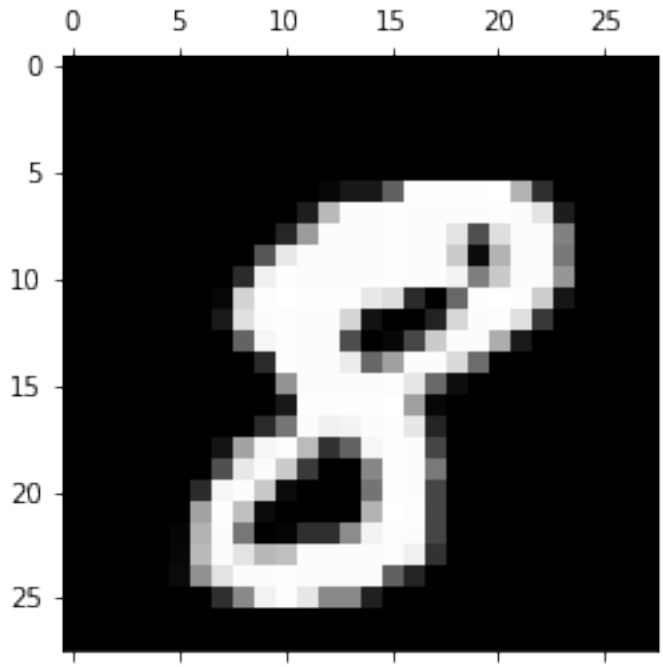

```
this digit is 7 from No.6
```



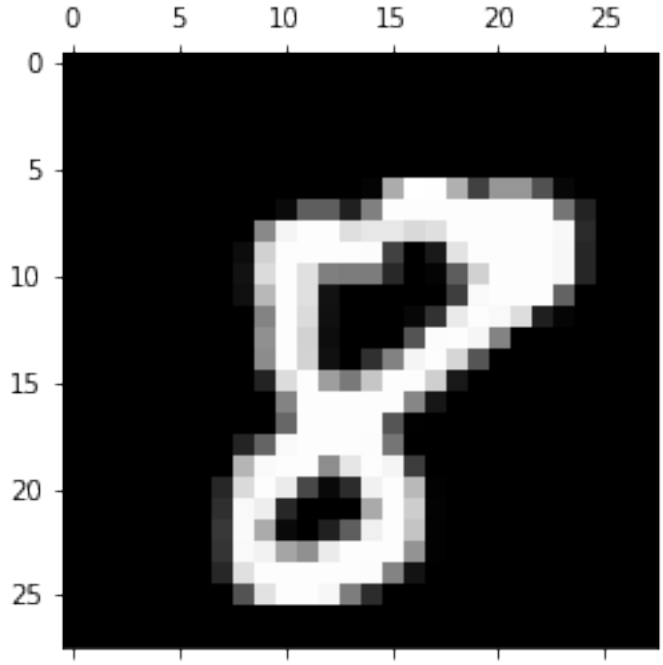
```
Nearest digit is 7.0 from No.15275
```



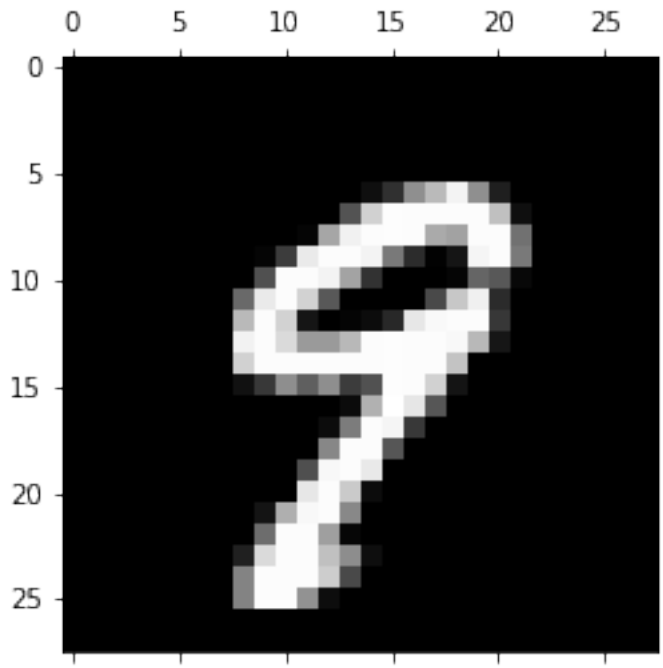

```
this digit is 8 from No.10
```



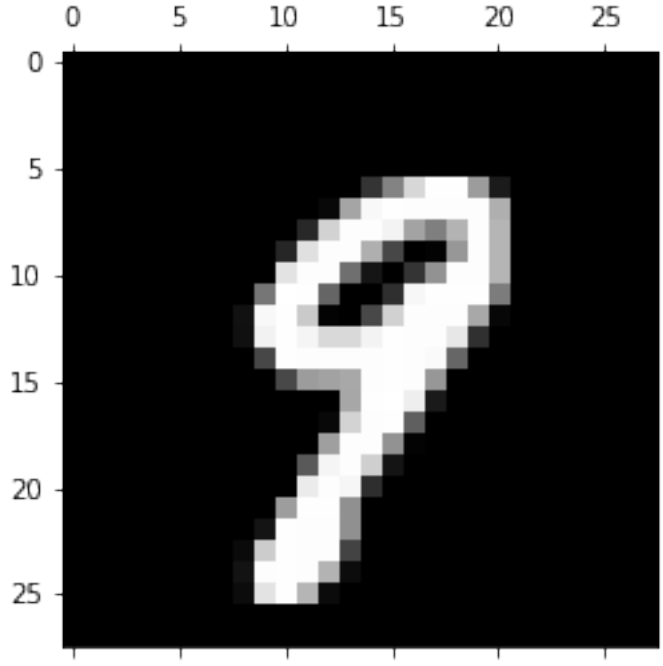

```
Nearest digit is 8.0 from No.32586
```



```
this digit is 9 from No.11
```




```
Nearest digit is 9.0 from No.35742
```

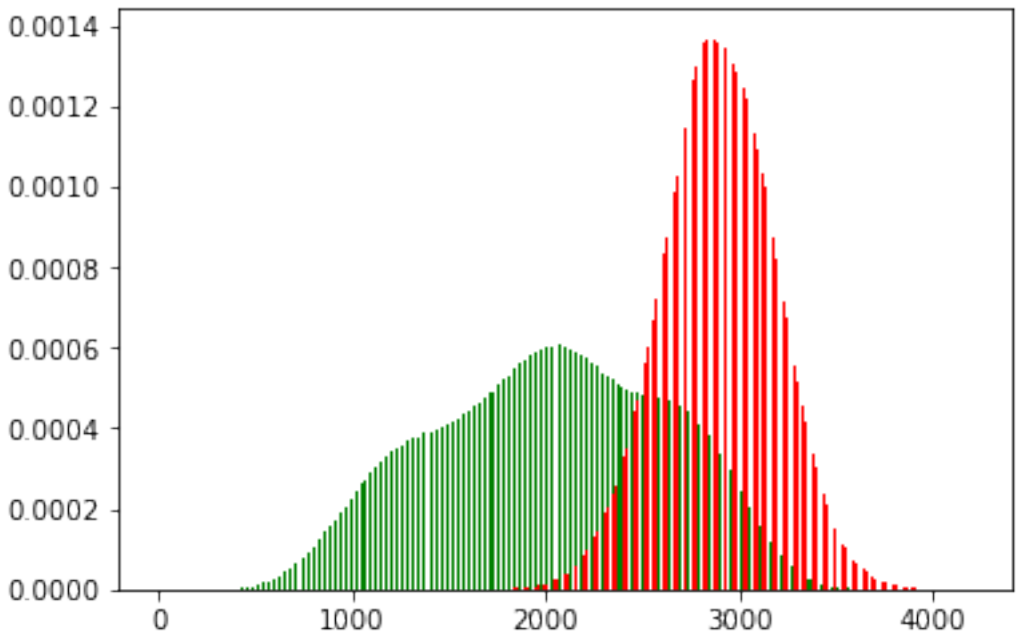


1.5 Q1e

Computing pairwise distances of all 0's and 1's. Plotting histogram of genuine (00 and 11 distances combined) and imposter(01 distances) distances. Genuine distances are marked green and imposter distances are marked red below.

```
In [18]: zeros = data[data[:,0]==0]
         ones = data[data[:,0]==1]
         pairDist00 = skl.pairwise_distances(zeros)
         pairDist11 = skl.pairwise_distances(ones)
         pairDist01 = skl.pairwise_distances(zeros, ones)
         pairGE = np.append(pairDist00, pairDist11)

In [20]: plt.hist(pairGE.flatten(), bins = 'auto', normed = True, rwidth = 0.5, color = 'g')
         #plt.hist(pairDist11.flatten(), bins = 'auto', normed = True, rwidth = 0.3, color = 'g'
         plt.hist(pairDist01.flatten(), bins = 'auto', normed = True, rwidth = 0.5, color = 'r')
         plt.show()
```



1.6 Q1f

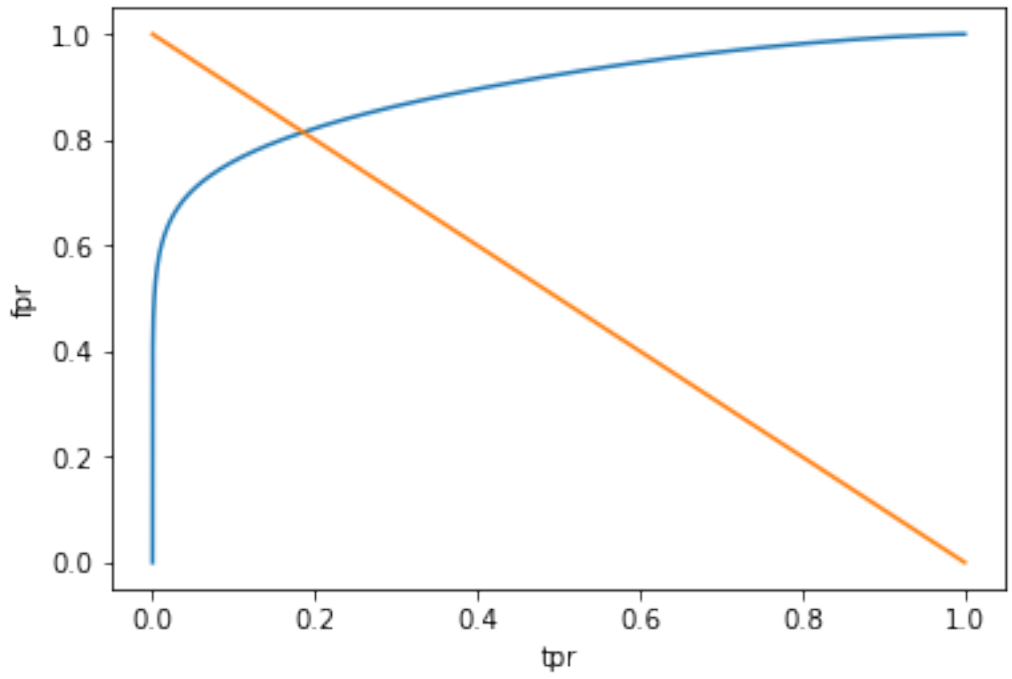
Genearting ROC curve below with true positive rate(tpr) as x-axis and false positive rate(fpr) as y-axis. We find equal error rate is 0.18554865605818877 for this training data. If we just guess randomly, i.e. whenever there is a test input, we have 50% chance to guess it as a "0" and 50%

chance to guess it as a "1". Then the expectation of error rate will be ratio of 0's vs 1's in the test set.

```
In [23]: fpr = [] # false_positive_rate
         tpr = [] # true_positive_rate
         lenGE = float(len(pairGE))
         len01 = float(len(pairDist01.flatten()))
         for thr in range(0, 4500, 5):
             tmp1 = float(np.count_nonzero(pairGE < thr))/lenGE
             tmp2 = float(float(np.count_nonzero(pairDist01<thr))/len01)
             tpr.append(tmp1)
             fpr.append(tmp2)
```

```
In [24]: # This is the ROC curve
         plt.plot(fpr, tpr)
         plt.xlabel("tpr")
         plt.ylabel("fpr")
         # This is the AUC
         auc = np.trapz(tpr,fpr)

         x = np.linspace(0.0, 1.0, num=100)
         y = 1-x
         plt.plot(y,x)
         plt.show()
```



```
In [25]: eer = brentq(lambda x : 1. - x - interp1d(fpr, tpr)(x), 0., 1.)
         print ("EER is " + repr(eer))
```

```
EER is 0.18554865605818877
```

1.7 Q1g

Below is our implementation of kNN, which is able to take in a test set of dimension m*p, i.e. m data points each with p features and output labels for each of m points.

```
In [27]: #implement kNN
         def kNN(data, label, test, k):
             # ndarray data has dimension n*p
             # label has dimension n
             # ndarray test has dimension m*p
             # y has dimension m
             y = []
             Dist = skl.pairwise_distances(data, test) # n*m
             #idx = np.argpartition(Dist, k, axis = 0)
             #ypredict = label[idx, np.arange(Dist.shape[1])[None,:]] #
             #return stats.mode(ypredict[:k], axis = )
             for i in range(test.shape[0]):
                 idx = np.argpartition(Dist[:,i], k)
                 y.append(stats.mode(label[idx[:k]]).mode[0])
             return y
```

1.8 Q1h

Here is 3-fold analysis on training data with k=1. The average accuracy is 0.9643809523809667.

```
In [29]: k = 1
         matrixList = []
         error = []
         for trainKF, testKF in (cross_validation.KFold(len(data), n_folds=3)):
             predicted = kNN(data[trainKF,1:785], data[trainKF, 0], data[testKF, 1:785], k)
             error.append(np.mean(predicted != data[testKF, 0]))
             matrixList.append(metrics.confusion_matrix(data[testKF, 0], predicted))
```

```
In [30]: print ("Average accuracy on k=1, 3-fold is: " + repr(1-np.mean(error)))
```

```
Average accuracy on k=1, 3-fold is: 0.96438095238095234
```

1.9 Q1i

Display the confusion matrix of 3-fold computation below. Number 8 is the most tricky to classify.

```
In [46]: conf = np.zeros((10,10), dtype=np.int)
         for m in matrixList:
             conf = np.add(conf, m)
         print("Here is the confusion matrix:")
         print(conf)
         np.fill_diagonal(conf, 0)
         missC = np.sum(conf, axis = 1)
         print ("\n Number of misclassification for each number below:")
         print(missC)
         print("\n The most tricky number to predict is " + repr(np.argmax(missC)))
```

```
Here is the confusion matrix:
[[4099    1    5    1    0    6   16    0    2    2]
 [   0 4646    9    3    5    2    4    9    4    2]
 [  29   24 4000   21    5    5    5   70   13    5]
 [   2    8   31 4148    0   75    1   26   37   23]
 [   2   39    0    0 3892    1   12   14    1  111]
 [   9    5    1   61    6 3622   48    4   14   25]
 [  30    7    0    1    6   19 4073    0    1    0]
 [   1   43   15    2   12    0    0 4264    0   64]
 [  12   33   18   74   13   59   16   12 3782   44]
 [  12    9    3   25   60   13    3   73   12 3978]]

 Number of misclassification for each number below:
[ 33  38 177 203 180 173  64 137 281 210]

 The most tricky number to predict is 8
```

1.10 Q1j

Import test data and generate predictions for test data with my kNN. Submission screenshot included.

```
In [47]: test =  np.loadtxt(fname = 'test.csv', delimiter = ',', skiprows = 1)
```

```
In [52]: results = kNN(data[:,1:785], data[:, 0], test, 1)
```

```
In [53]: df = pd.DataFrame(results)
         df.index.name='ImageId'
         df.index+=1
         df.columns=['Label']
         df.to_csv('results.csv', header=True)
```

```
In [54]: from IPython.display import Image
         Image("digits.png")
```

```
Out[54]:
```

All Successful Selected

Submission and Description	Public Score	Use for Final Score
results.csv 4 days ago by HuajunBai CS5785 HW1 submitting	0.97114	☐