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
```

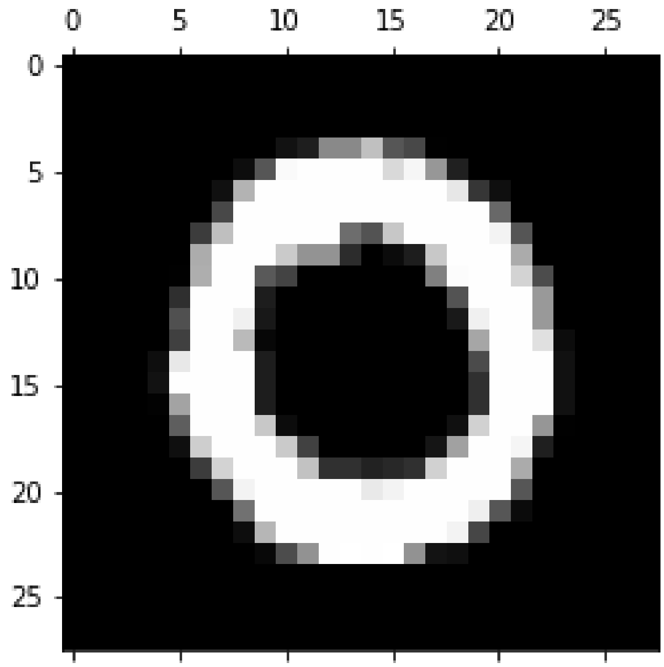
```
In [3]: # load data
        data =  np.loadtxt(fname = 'train.csv', delimiter = ',', skiprows=1)
        n,p = data.shape
```

1.2 Q2b

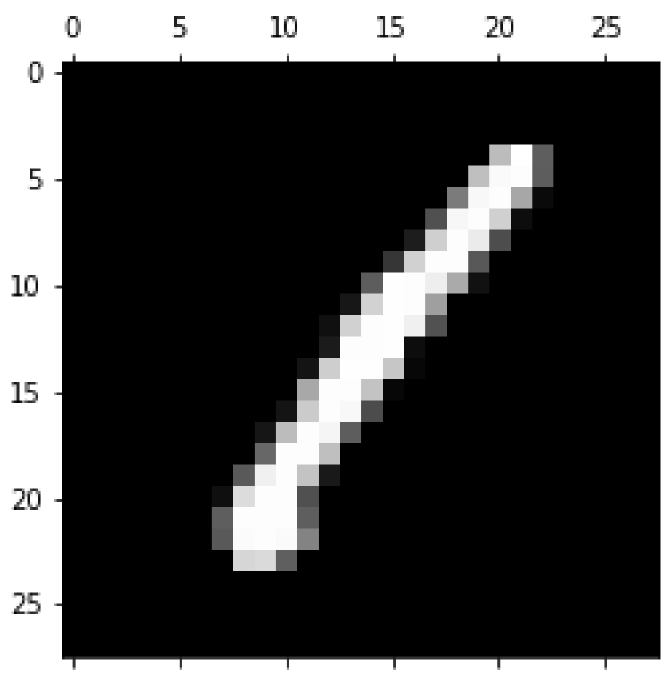
Below is the function to display any digit d. And a display of digits 0-9 in the data.

```
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;
```

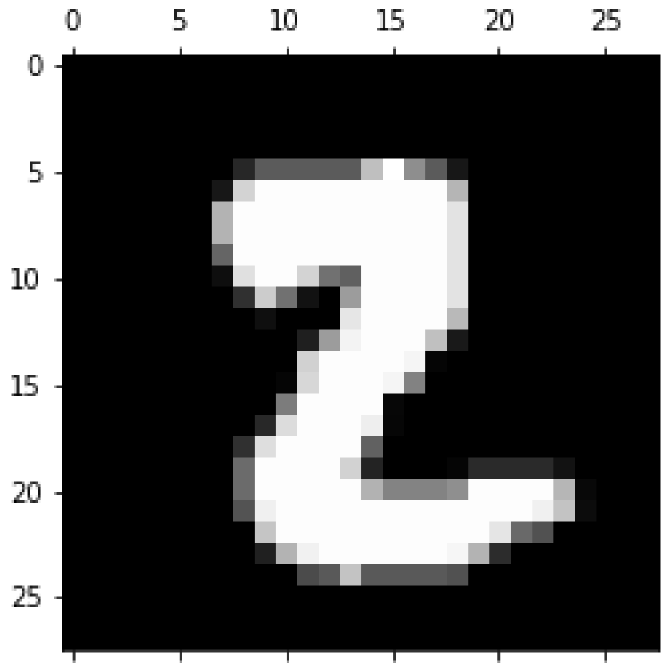
```
In [6]: for j in range(10):
            display(data, j)
```

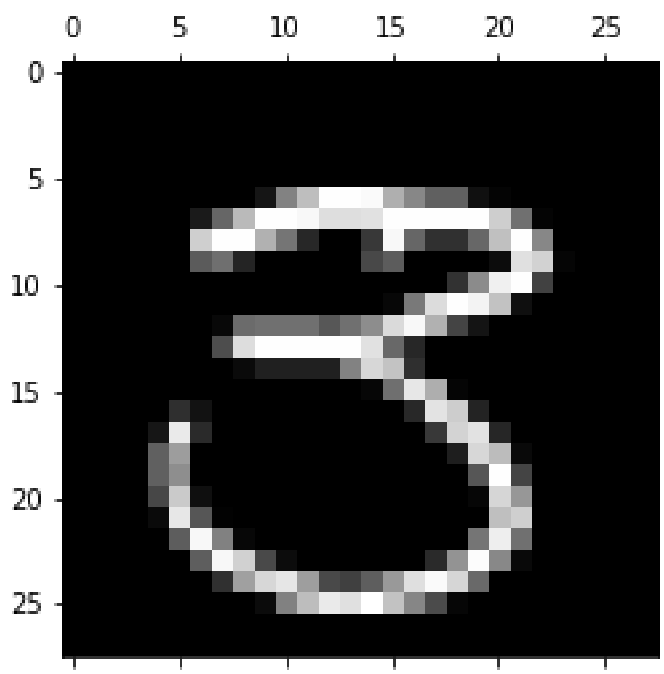


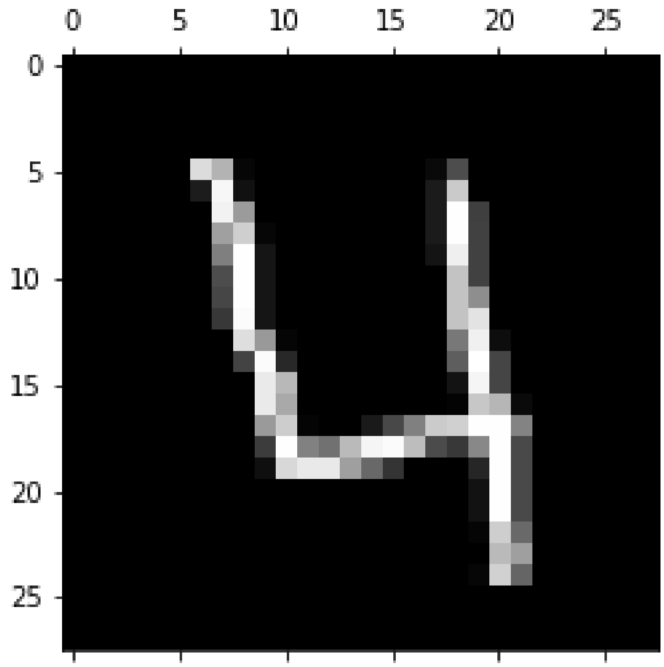
0
5
10
15
20
25

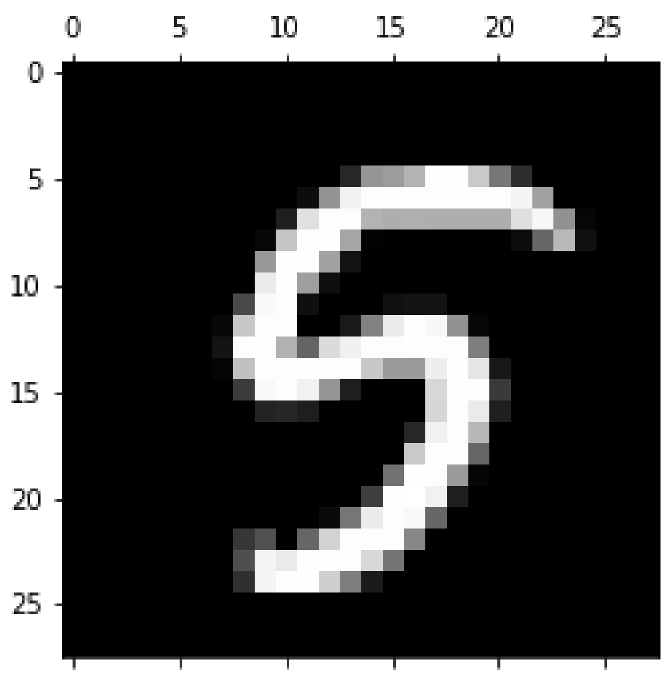


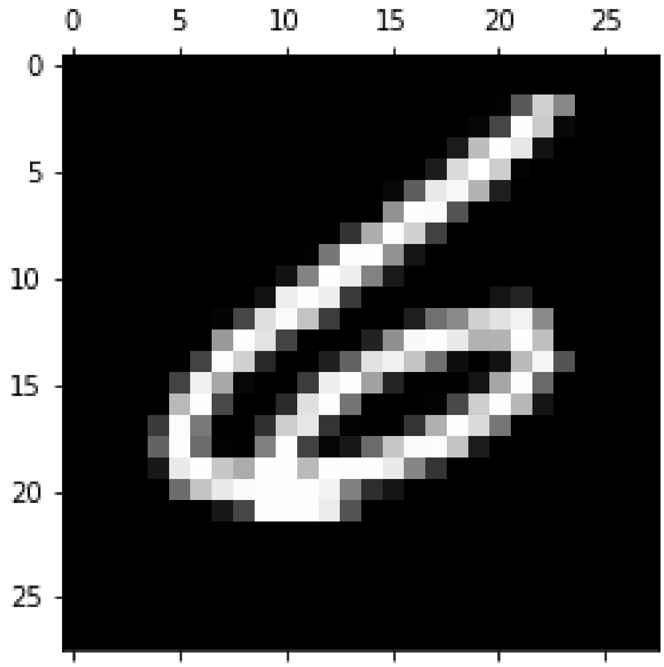
0
5
10
15
20
25

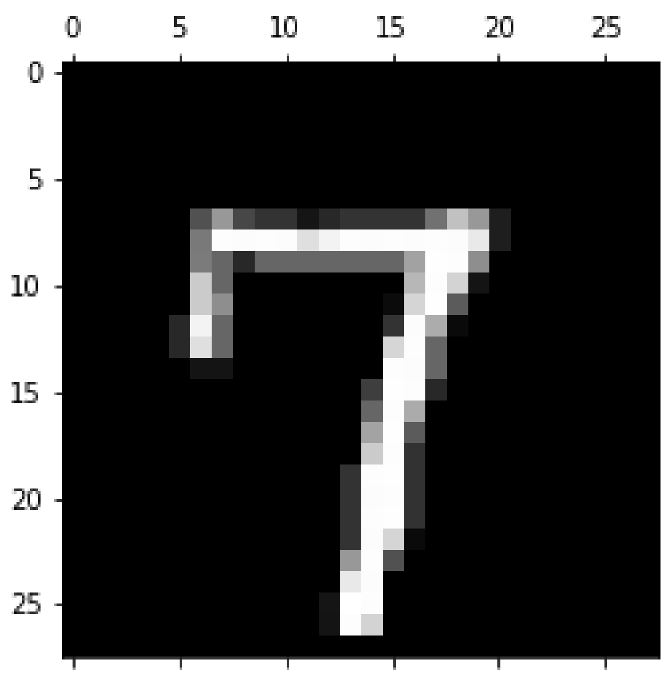

0
5
10
15
20
25

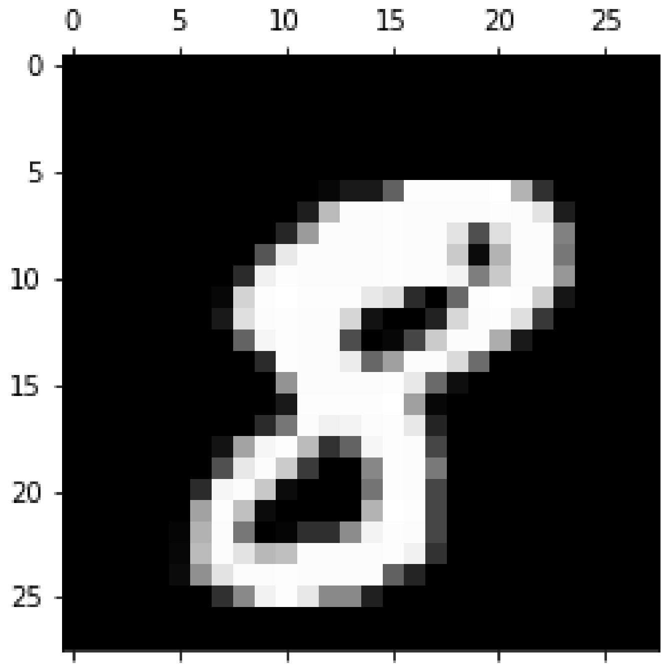

0
5
10
15
20
25

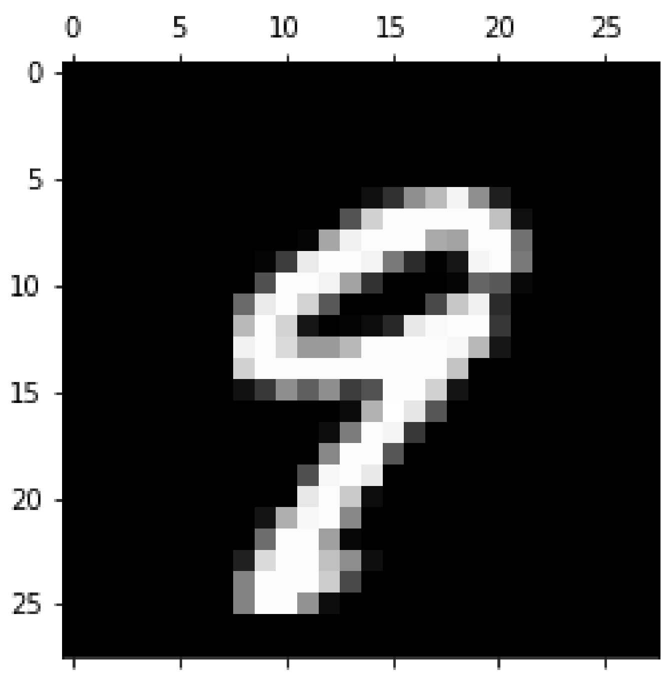

0 5 10 15 20 25
0 5 10 15 20 25


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

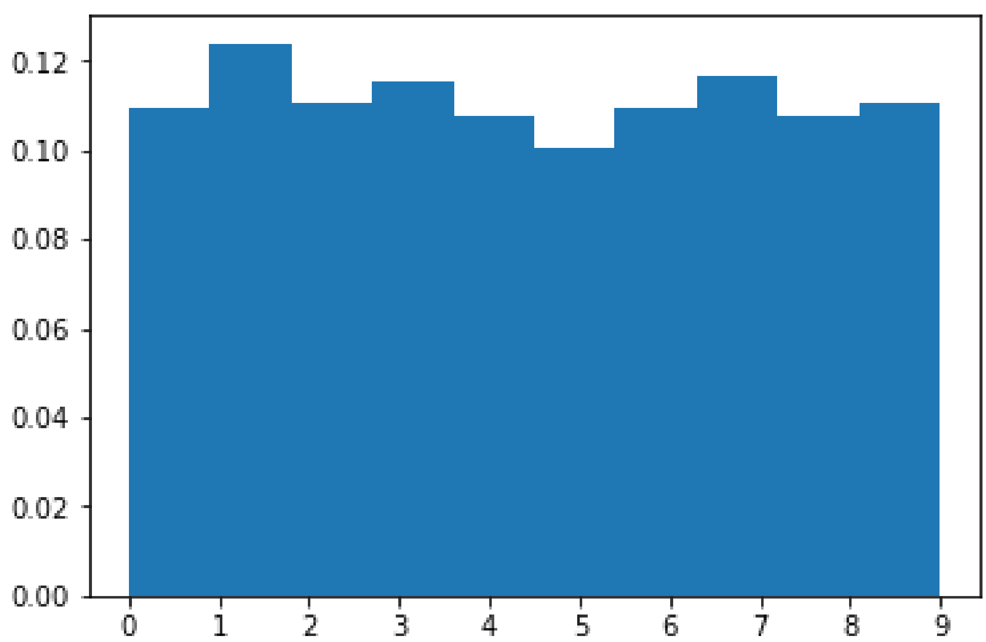
1.3 Q1c

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()
```



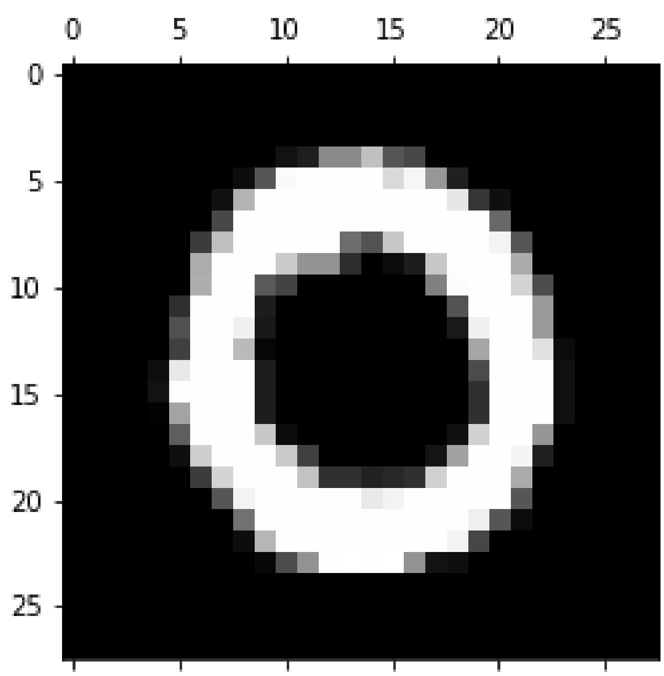
1.4 Q1d

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.

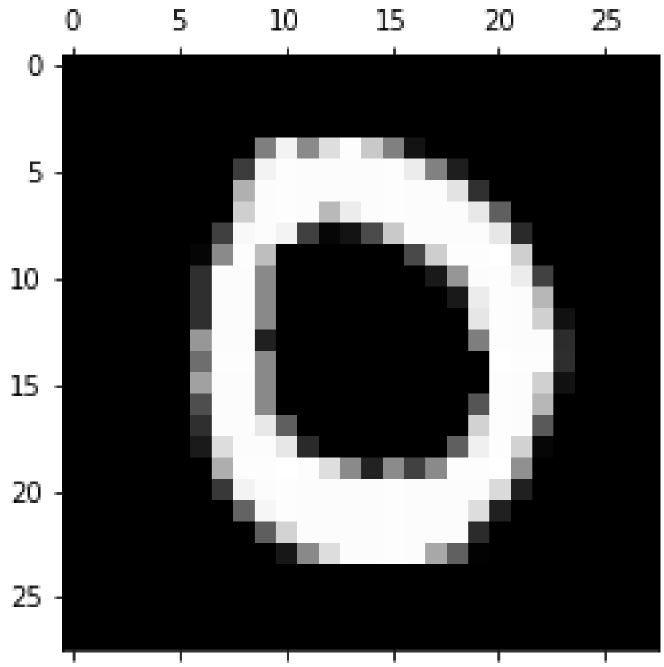
```
In [11]: def findNearest(index):
             diff = np.subtract(np.array(data[:, 1:785]), np.array(data[index, 1:785]));
             distM = np.square(diff) # delete index itself
             # Assume no same data set exist, i.e. no 0 dist except itself
             return np.argsort(np.sum(distM,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

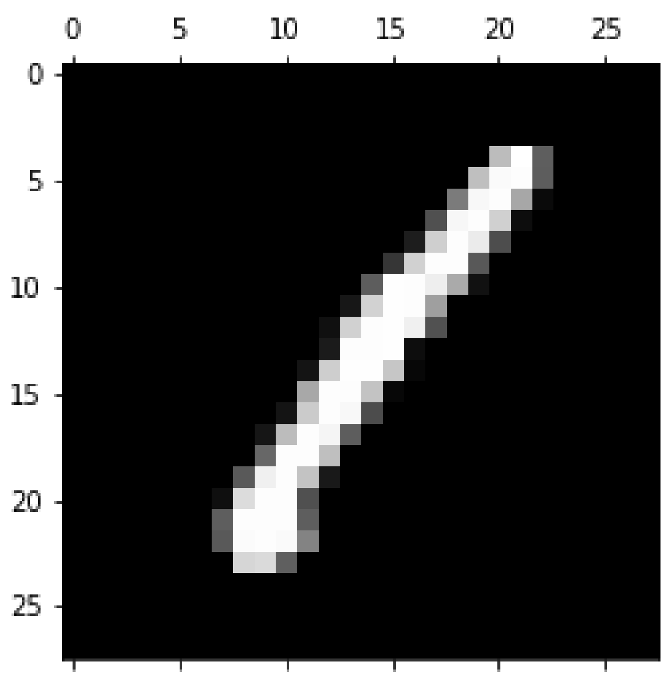
```
this digit is 0 from No.1
```



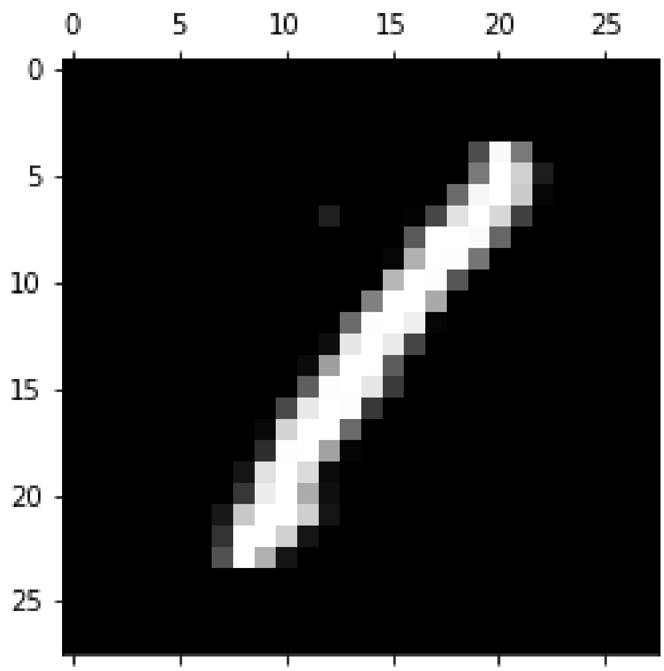
```
Nearest digit is 0.0 from No.12950
```



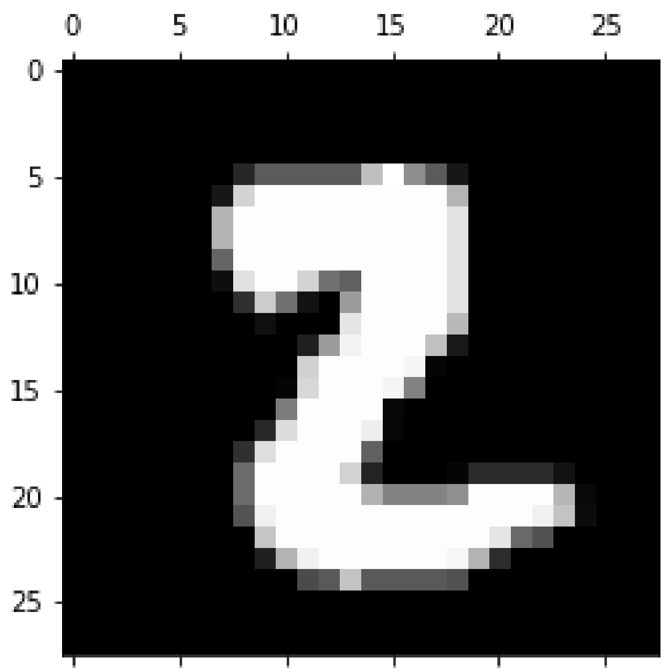
this digit is 1 from No.0



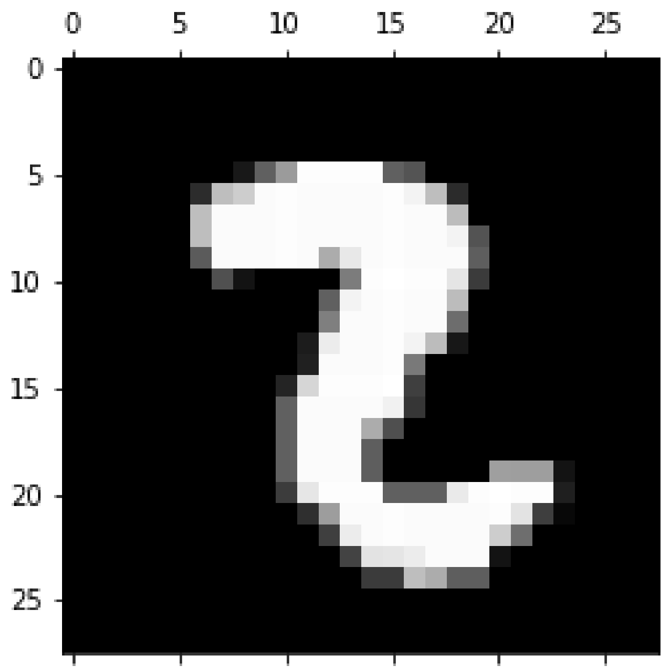
Nearest digit is 1.0 from No.29704



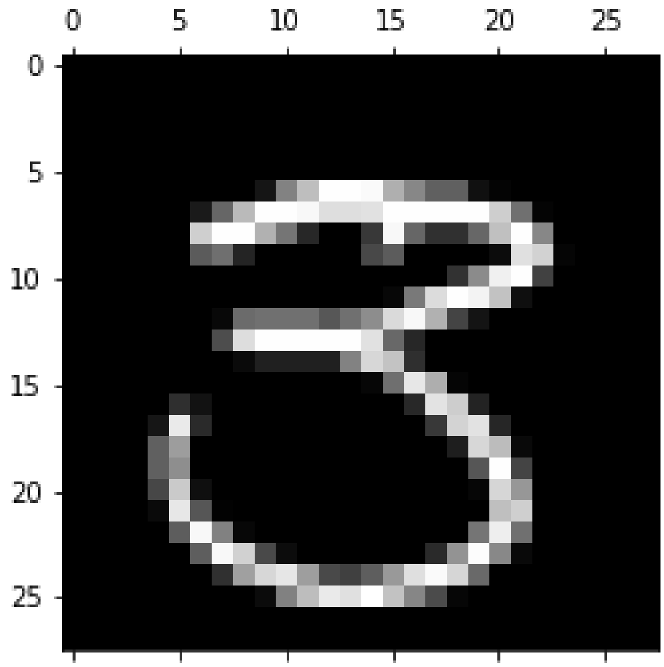
this digit is 2 from No.16



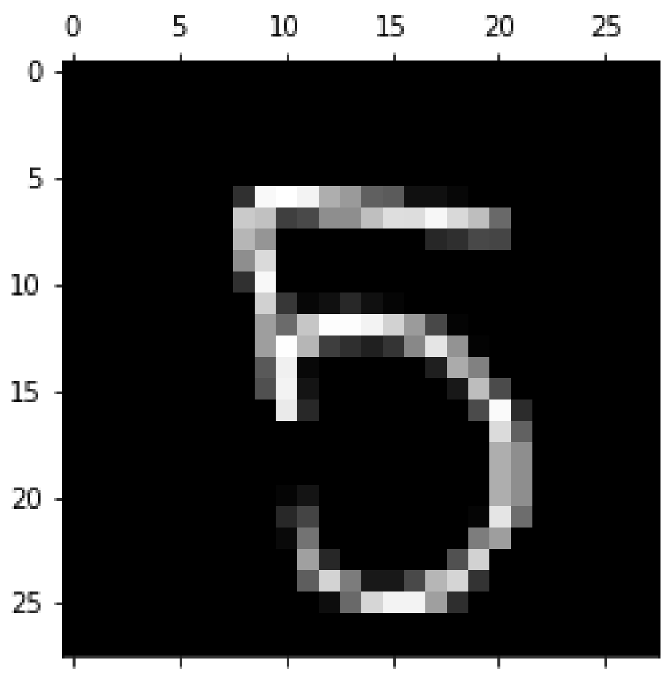
Nearest digit is 2.0 from No.9536



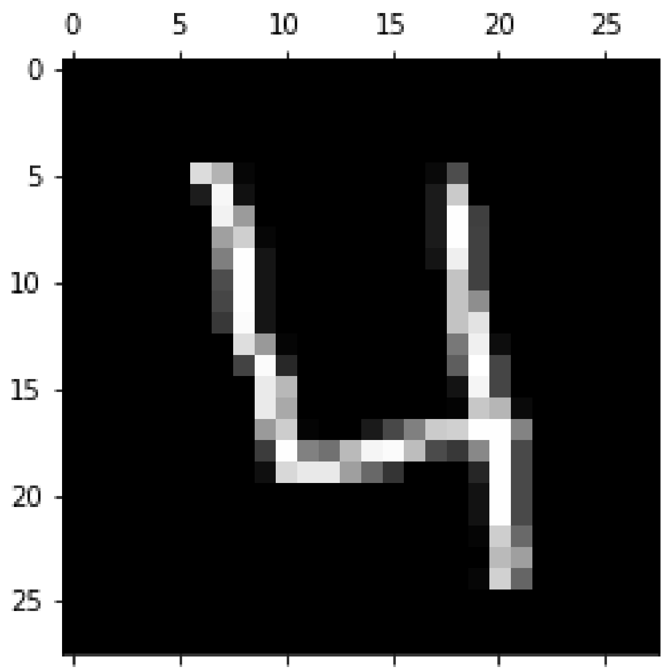
this digit is 3 from No.7



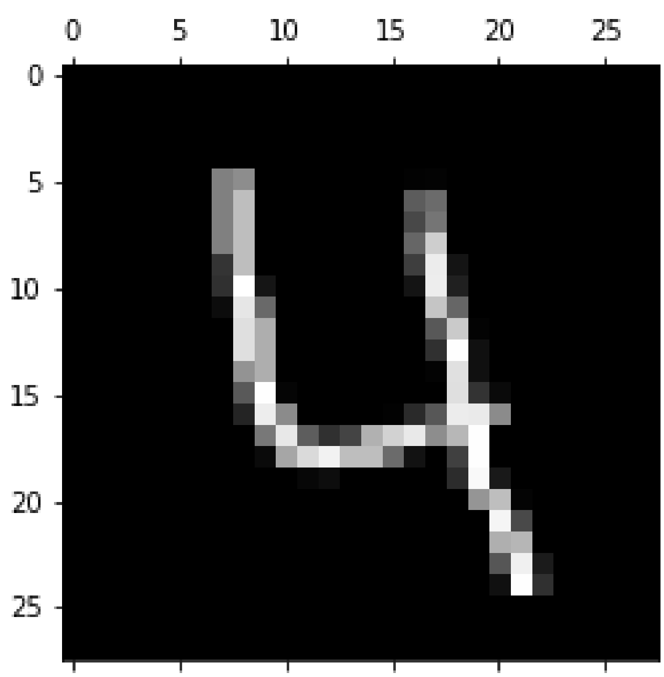
Nearest digit is 5.0 from No.8981



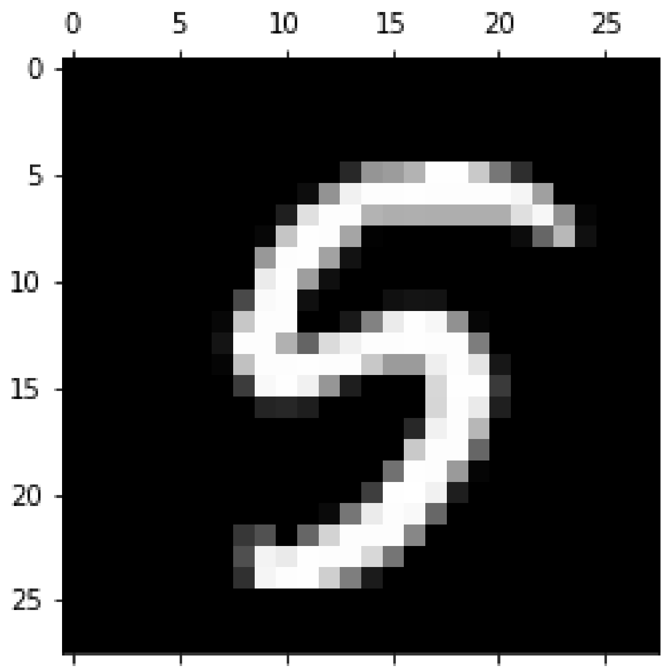
```
this digit is 4 from No.3
```



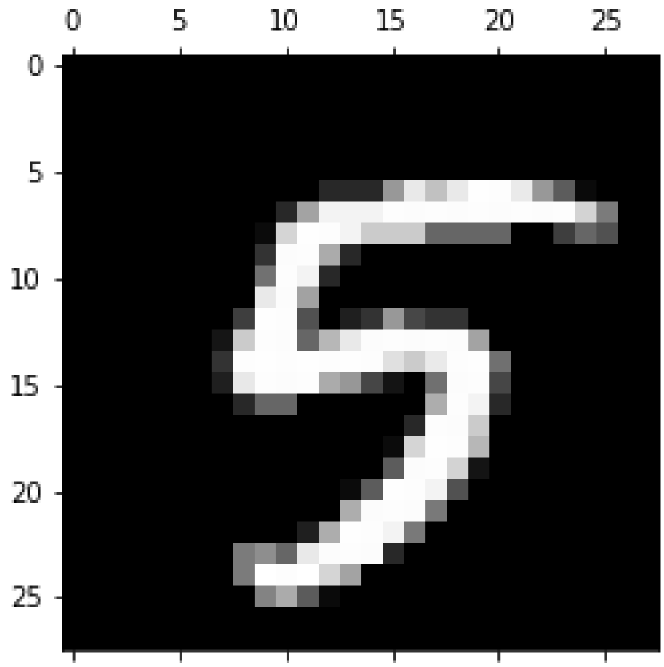

```
Nearest digit is 4.0 from No.14787
```



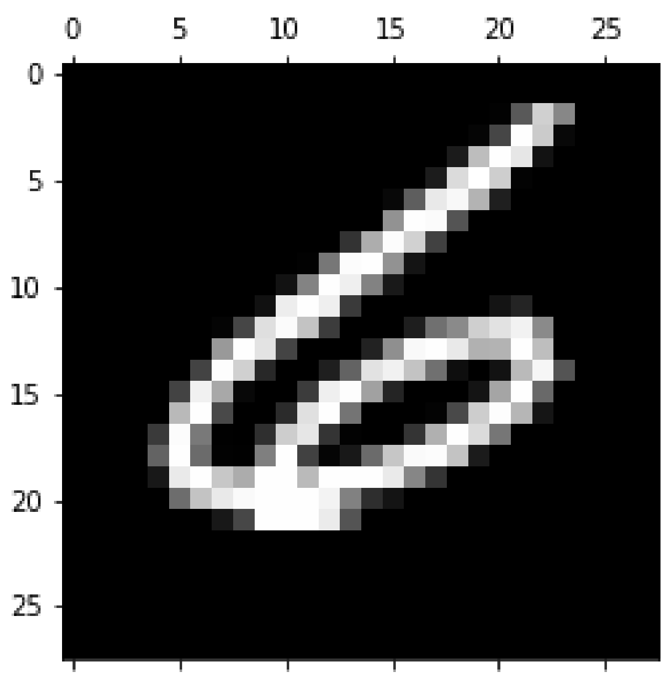
this digit is 5 from No.8



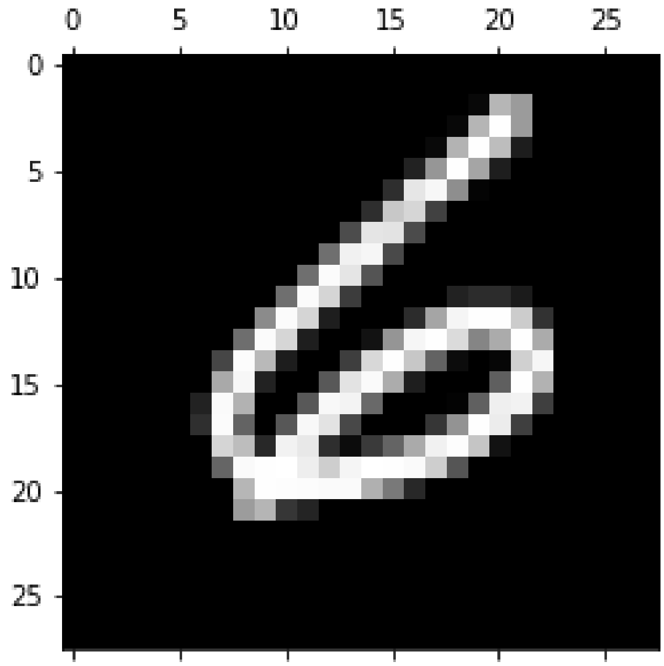
Nearest digit is 5.0 from No.30073



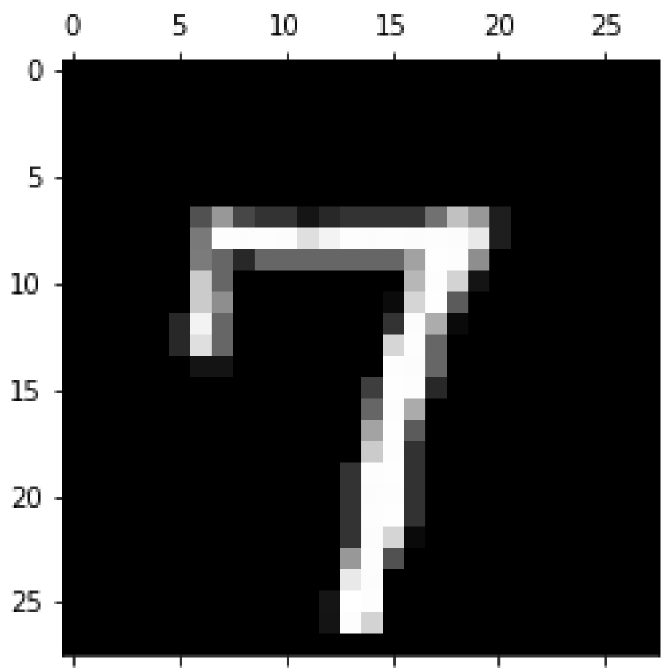
this digit is 6 from No.21



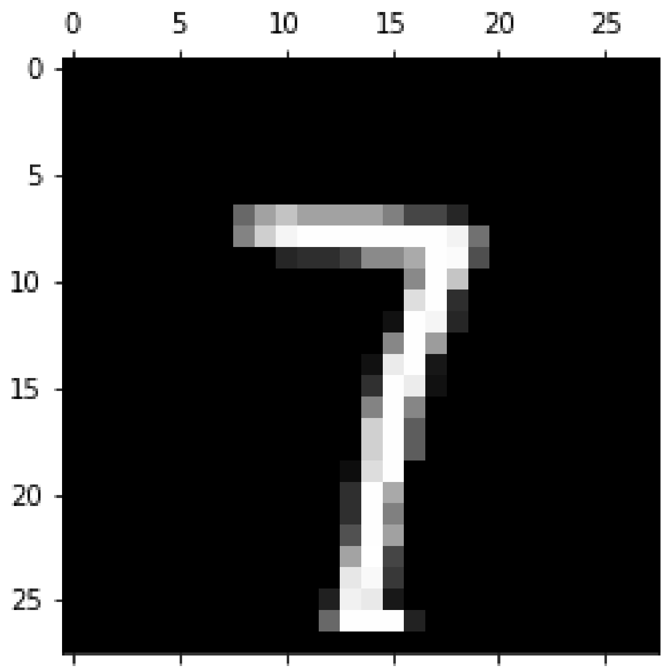
```
Nearest digit is 6.0 from No.16240
```



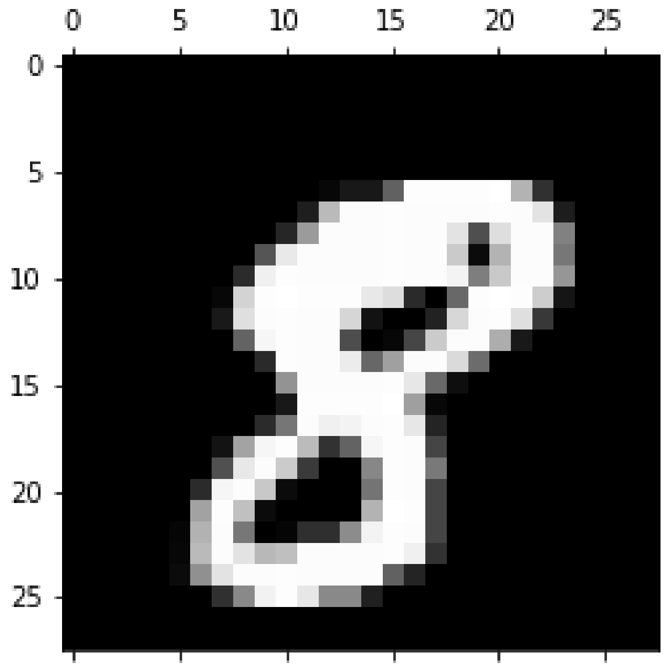

```
this digit is 7 from No.6
```



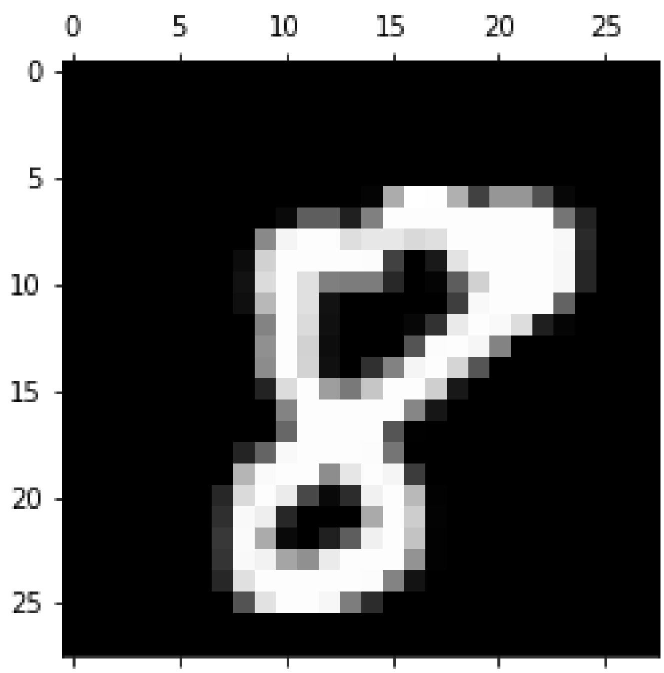
```
Nearest digit is 7.0 from No.15275
```



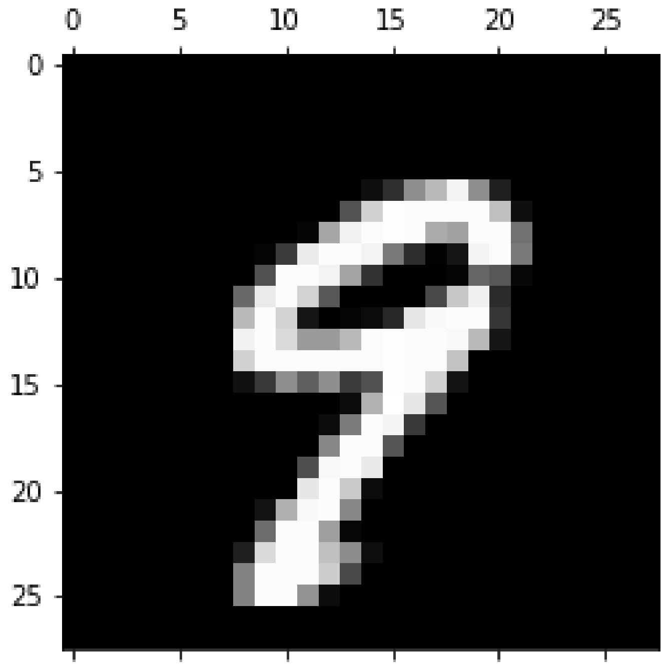

```
this digit is 8 from No.10
```



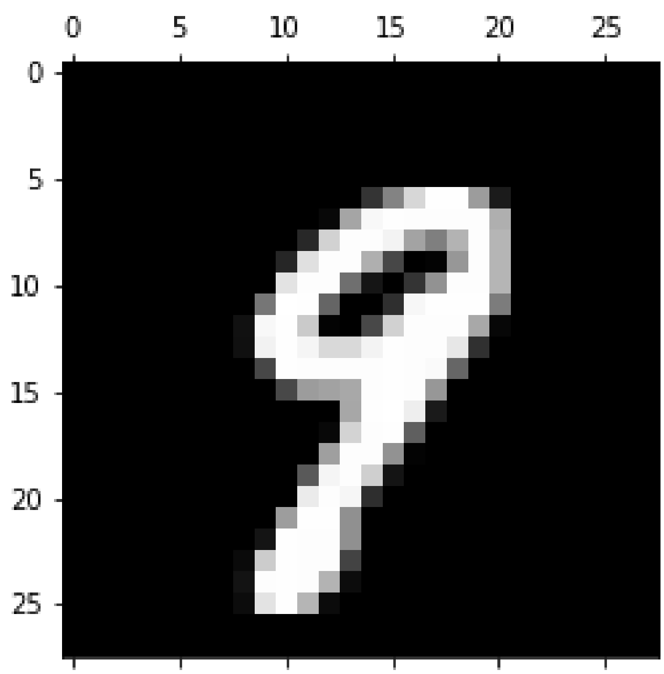
```
Nearest digit is 8.0 from No.32586
```



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




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

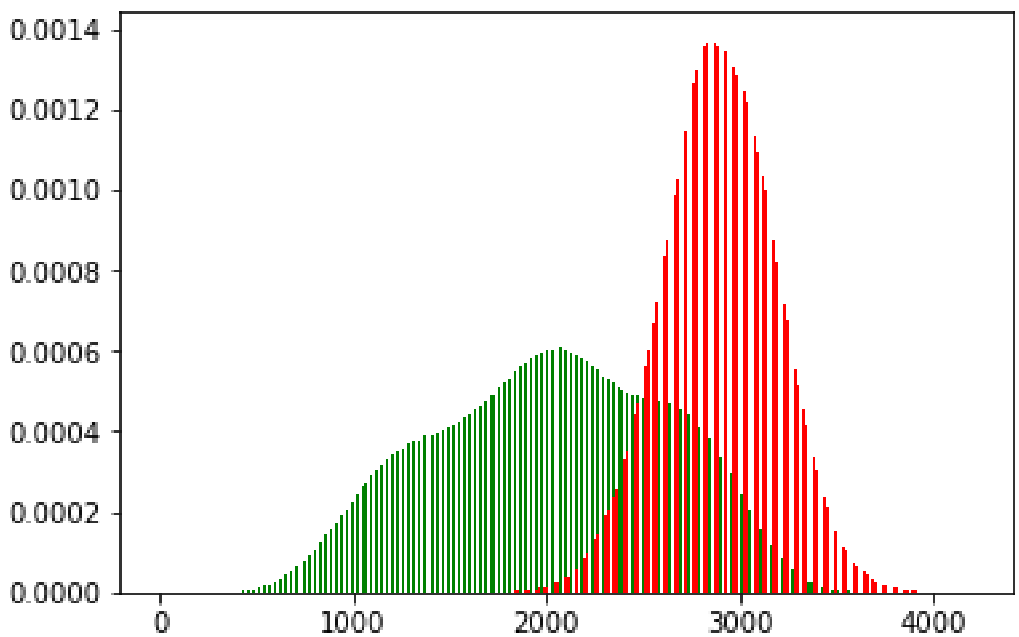


1.5 Q1e

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

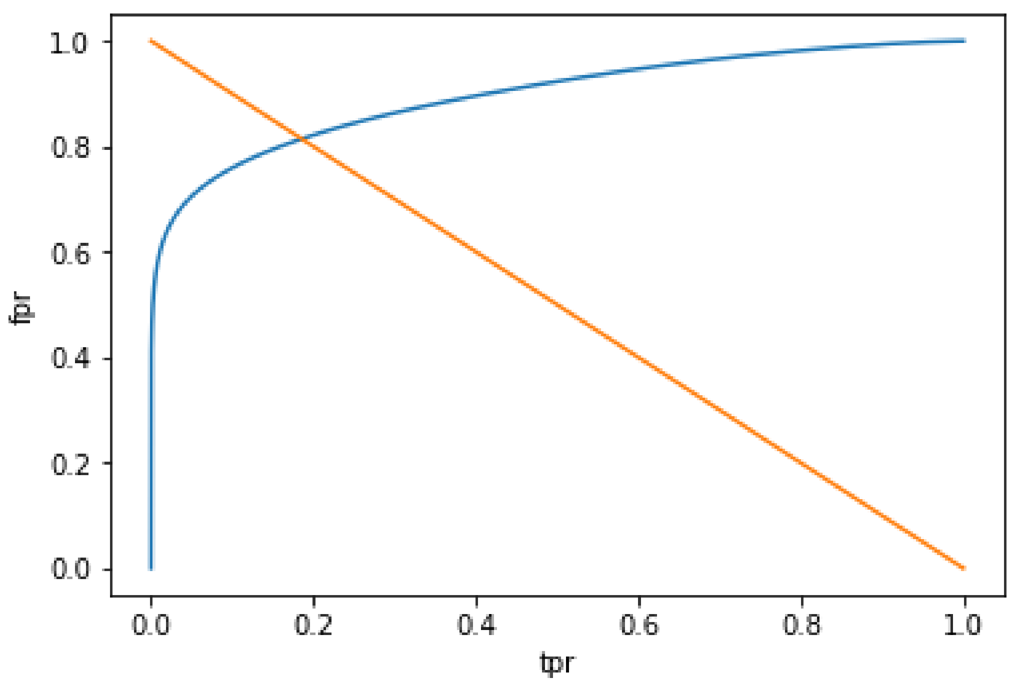
Genearting ROC curve below with true positive rate(tpr) as x-axis and false positive rate(fpr) as y-axis. We find equal error rate is 0.18554865605818877 for this training data. If we just guess randomly, i.e. whenever there is a test input, we have 50% chance to guess it as a "0" and 50%

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	☐

myTitanic

September 13, 2017

1 The Titanic Disaster

1.1 Q2a

Importing libraries and data below

```
In [1]: # import libraries
        import pandas as pd
        import numpy as np
        from sklearn import preprocessing, cross_validation, datasets, neighbors, linear_model
```

```
C:\Users\Huajun\Anaconda3\lib\site-packages\sklearn\cross_validation.py:44: DeprecationWarning:
  "This module will be removed in 0.20.", DeprecationWarning)
```

```
In [2]: # import data
        train = pd.read_csv("train.csv", dtype={"Age": np.float64},)
        test = pd.read_csv("test.csv", dtype={"Age": np.float64},)
```

```
In [3]: #Print to standard output, and see the results in the "log" section below after running
        print("\n\nTop of the training data:")
        print(train.head())
```

```
Top of the training data:
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
```

```
4                             Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

1.2 Q2b

Choose 3 features: "Sex", "Age" and "PClass" according to **this article** Then perform 5-fold cross-validation using logistic regression. Expected score (75%, 80%).

```
In [21]: # Fill missing age values for the train and test data with corresponding mean value,
         # and convert values from float to integer.
         train.loc[train["Sex"] == "male", "Sex"] = 0
         train.loc[train["Sex"] == "female", "Sex"] = 1

         test.loc[test["Sex"] == "male", "Sex"] = 0
         test.loc[test["Sex"] == "female", "Sex"] = 1

         train["Age"] = train["Age"].fillna(train["Age"].mean())
         train['Age'] = train['Age'].astype(int)

         test["Age"] = test["Age"].fillna(test["Age"].mean())
         test['Age'] = test['Age'].astype(int)
```

```
In [39]: # choose features
         # pclass--2, sex--4, age--5
         data = pd.DataFrame.as_matrix(train)
         myData = data[:, [0,1,2,4,5]]
```

```
In [31]: logistic = linear_model.LogisticRegression()
         for trainKF, testKF in (cross_validation.KFold(len(myData), n_folds=5)):
             print('LogisticRegression score: %f'
                   % logistic.fit(myData[trainKF, 2:5], list(myData[trainKF, 1])).score(myData[t
```

```
LogisticRegression score: 0.798883
LogisticRegression score: 0.814607
LogisticRegression score: 0.775281
LogisticRegression score: 0.752809
LogisticRegression score: 0.808989
```

1.3 Q2c

Computing test prediction and submitting below. Accuracy achieved at 75.6%

In [10]:
```
test.head()
```

Out[10]:

	PassengerId	Pclass	Name	Sex
0	892	3	Kelly, Mr. James	male
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female
2	894	2	Myles, Mr. Thomas Francis	male
3	895	3	Wirz, Mr. Albert	male
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female

	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	34	0	0	330911	7.8292	NaN	Q
1	47	1	0	363272	7.0000	NaN	S
2	62	0	0	240276	9.6875	NaN	Q
3	27	0	0	315154	8.6625	NaN	S
4	22	1	1	3101298	12.2875	NaN	S

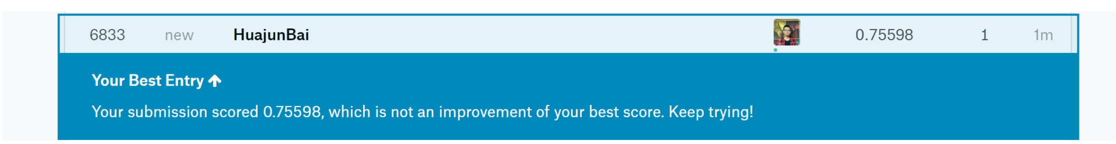
In [35]:
```
results = logistic.predict(pd.DataFrame.as_matrix(test)[:,[1,3,4]])
```

In [37]:
```
#------------RESULT SUBMISSION---------#
submission = pd.DataFrame({
        "PassengerId": test["PassengerId"],
        "Survived": results
    })

submission.to_csv('pred.csv', index=False)
```

In [38]:
```
from IPython.display import Image
Image("titanic.png")
```

Out[38]:



CS 5785 Applied Machine Learning

Homework 1

Huajun Bai / hb364

Hao Zheng / hz466

Written Exercise

Q1

Rule 1: $Var[X] = E[X^2] - E[X]^2$

Rule 2: $Cov[X, Y] = E[XY] - E[X]\,E[Y]$

$$Var(X - Y) = E[(X - Y)^2] - E[X - Y]^2 = E[X^2 - 2XY + Y^2] - (E[X] - E[Y])^2$$
$$= E[X^2] - 2E[XY] + E[Y^2] - E[X]^2 + 2E[X]E[Y] - E[Y]^2$$
$$= (E[X^2] - E[X]^2) - 2(E[XY] - E[X]E[Y]) + (E[Y^2] - E[Y]^2)$$
$$= Var[X] - 2Cov[X, Y] + Var[Y]$$

Q2

Let + denotes testing positively and - dentoes testing negatively. D denotes defective and N denotes not defective. Then

$P(+|D) = 0.95,\ P(-|N) = 0.95,\ P(D) = 1/100000,\ P(N) = 1 - P(D)$

a) By Bayes Rule,

$P(D|+) = \frac{P(+|D)P(D)}{P(+|D)P(D)+P(+|N)P(N)} = 0.019\%$

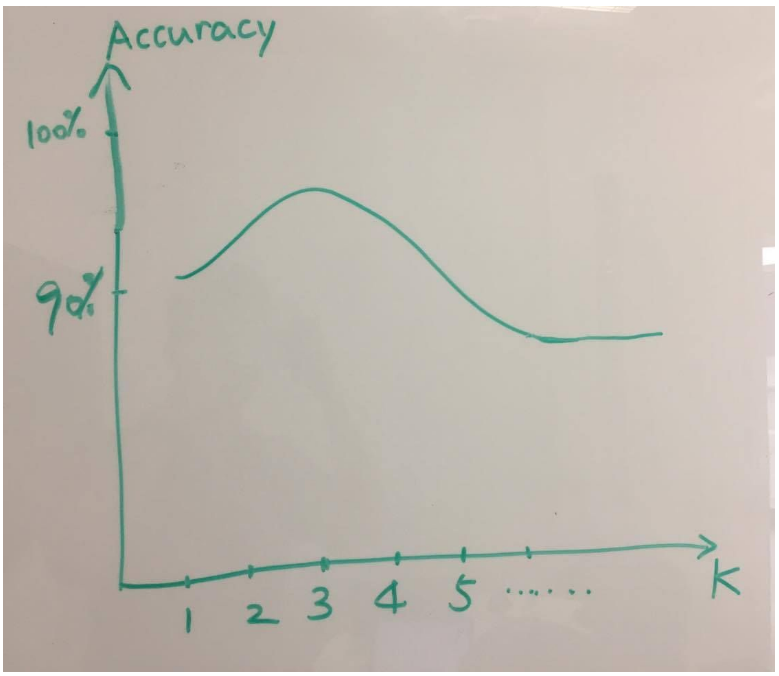
b) $10\,m * P(N)P(+|N) = 99999 * 5 = 499995$

$10\,m * P(D)P(-|D) = 100 * 0.05 = 5$

If there are 10 million widgets produced each year, then 499995 good widgets are thrown each year and 5 bad widgets are shipped to customers.

Q3

a) When k=n, the classifier just outputs the mode of all data points. Since the data set is choosen as half one class half anther class, the prediction error will be 0.5. When k=1, the classifier will choose the class of the nearest point of the input as an output result. The prediction error will depend on how overlapped the dataset is. In general, when k decreases from n to 1, the error should decrease at first, reach its minimum and slight increase at last.
b) This is two-fold analysis of error rate. When k decreases from n to 1, the error should decrease at first, reach its minimum and slight increase at last. Graph looks like this:



c) How many folds we should use? When number of fold is greater, bias is reduced but variance is increased as each set has smaller number of samples, and computation time is also increased.
   K = 10 usually is not a bad choice. Another way to choose K is to let K be a function of sample size, for example set $K = \sqrt{N}$.
d) Since the underlying assumption of kNN is the closer the points are, the more likely they are of the same class. It is intuitive to place higher weights on points that are closer to the test point when k is large. One way is to set weight for node :
   $w_v = \frac{1}{dist(v,x)} / (\Sigma_{w \in K} \frac{1}{dist(w,x)})$ where K denotes the set of k nearest nodes.
e) One reason is that as dimension becomes very high, the distance of points becomes blurred as it is calculated across all features. Distances between points will become insignificant. Another reason is that when dimension is high, the pairwise distance of points will take a lot of computation, thus undesirable.