CSCI 6364 - Machine Learning

Project 1 - MNIST

Student: Shifeng Yuan

GWid: G32115270

Language: Python

Resource: MNIST data from Kaggle

In [4]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt, matplotlib.image as mpimg
from sklearn.model_selection import train_test_split
%matplotlib inline
```

1. Dataset Details

Here we have two datasets, one is training data and another is the testing data.

1. The training data contains the data of 28000 images
2. Each image is described as 28*28=784 columns with numbers representing its lightness or darkness
3. The first column is the actual number of what the image represents
4. The testing data is the same as the training data but it does not have the "label" column which should be generated.

Data Spliting

First, we need to read the data into a variable called dataset. And we should split the data into images and labels as two parts. Usually, we divide our dataset into 2 to 3 parts. Here, I split the dataset into training data (80%) and testing data(20%)

In [5]:
```
dataset=pd.read_csv('mnistdata/train.csv')
images=dataset.iloc[0:28000,1:]
labels=dataset.iloc[0:28000,:1]
train_images,test_images,train_labels,test_labels=train_test_split(images,labels,random_state=2,test_size=0.2)
```

Inspect the Dataset

In [93]:
```
# Read the dataset and then print the head
print( len(dataset) )
print( dataset.head() )
```

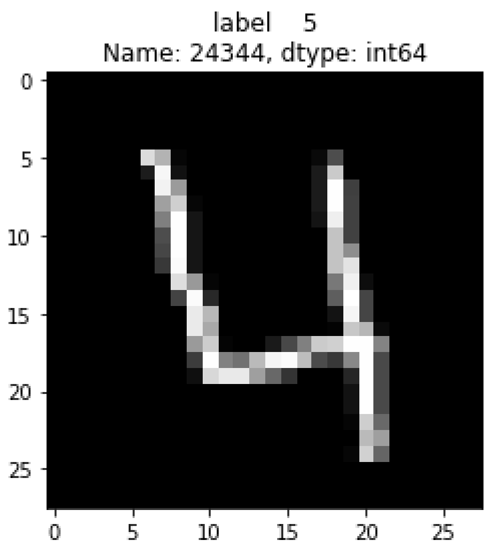
```
42000
   label  pixel0  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  \
0      1       0       0       0       0       0       0       0       0
1      0       0       0       0       0       0       0       0       0
2      1       0       0       0       0       0       0       0       0
3      4       0       0       0       0       0       0       0       0
4      0       0       0       0       0       0       0       0       0

   pixel8    ...     pixel774  pixel775  pixel776  pixel777  pixel778  \
0       0    ...            0         0         0         0         0
1       0    ...            0         0         0         0         0
2       0    ...            0         0         0         0         0
3       0    ...            0         0         0         0         0
4       0    ...            0         0         0         0         0

   pixel779  pixel780  pixel781  pixel782  pixel783
0         0         0         0         0         0
1         0         0         0         0         0
2         0         0         0         0         0
3         0         0         0         0         0
4         0         0         0         0         0

[5 rows x 785 columns]
```

Dataset Visualization

We can use the .imshow() in the matplotlib package to visualize the data as a picture. We first select a row, here it the 4th row, and then reshape it into 28*28 matrix, finally the package gives the picture.

In [61]:
```python
i=3
img=images.iloc[i].values
img=img.reshape(28,28)
plt.imshow(img,cmap='gray')
plt.title(train_labels.iloc[i])
```

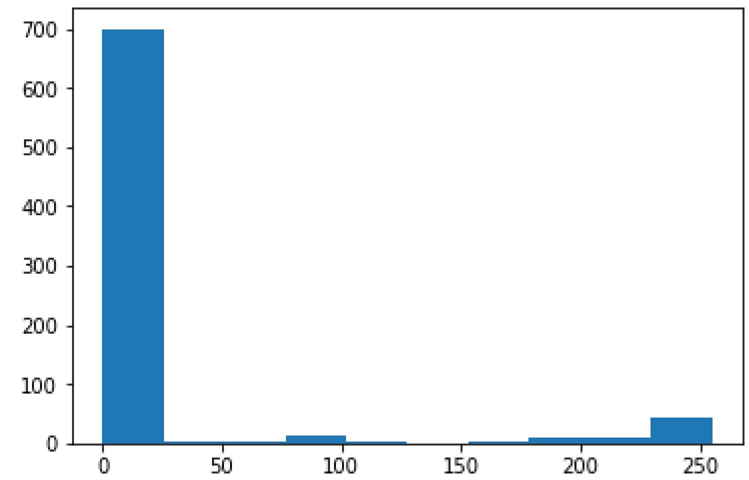
Out[61]:
```
Text(0.5,1,'label    5\nName: 24344, dtype: int64')
```



Use the .hist() to draw a histgram of the data.

In [68]:
```python
plt.hist(images.iloc[0])
```

Out[68]:
```
(array([700.,   2.,   2.,  14.,   2.,   1.,   4.,   8.,   9.,  42.]),
 array([  0. ,  25.5,  51. ,  76.5, 102. , 127.5, 153. , 178.5, 204. ,
        229.5, 255. ]),
 <a list of 10 Patch objects>)
```



2. Algorithm Description

Selection of K

The selection of value K is important for KNN, usually, we make K the square root of the size of the test sample, however, because this dataset is too big, we simply make it 5.

The package sklearn.neighbors.KNeighborsClassifier implementing the K-nearest Neighbors classification.

Using the sklearn KNeighborsClassifier package, define the metric method as euclidean. we simply use a brute force algorithm.

In [13]:
```python
from sklearn.neighbors import KNeighborsClassifier
clf=KNeighborsClassifier(n_neighbors = 5, algorithm = 'brute', p = 2, metric = 'euclidean')
clf.fit(train_images,train_labels.values.ravel())
```

Out[13]:
```
KNeighborsClassifier(algorithm='brute', leaf_size=30, metric='euclidean',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')
```

3. Algorithm Results

Start predict and measure the accuracy of the algorithm.

In [14]:
```python
predictions=clf.predict(test_images)
```

In [15]:
```
print(predictions)
```

```
[1 8 1 ... 8 5 0]
```

Confusion Matrix

In [19]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(test_labels, predictions)
print(cm)
```

```
[[519   0   1   1   0   1   1   1   0   0]
 [  0 595   1   1   0   1   1   4   1   1]
 [  6  10 551   2   0   0   1  12   2   1]
 [  1   1   1 555   0   8   1   3   6   6]
 [  1   5   0   0 539   0   4   1   0  13]
 [  0   0   0  11   0 446   8   0   4   2]
 [  1   0   0   0   1   6 581   0   2   0]
 [  0  11   3   0   3   0   0 527   0   6]
 [  3   8   3   6   2  11   6   2 499   9]
 [  4   1   1   3   8   1   1   7   2 552]]
```

The full confusion matrix shown below, and the accuracy score is 0.9578571428571429.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	519	0	1	1	0	1	1	1	0	0
1	0	595	1	1	0	1	1	4	1	1
2	6	10	551	2	0	0	1	12	2	1
3	1	1	1	555	0	8	1	3	6	6
4	1	5	0	0	539	0	4	1	0	13
5	0	0	0	11	0	446	8	0	4	2
6	1	0	0	0	1	6	581	0	2	0
7	0	11	3	0	3	0	0	527	0	6
8	3	8	3	6	2	11	6	2	499	9
9	4	1	1	3	8	1	1	7	2	552

In [46]:
```
from sklearn.metrics import accuracy_score
print(accuracy_score(test_labels,predictions))
```

```
0.9578571428571429
```

Then start predicting the test data in the test.csv

In [8]:
```
# read the test data into variable testd
testd=pd.read_csv('mnistdata/test.csv')
```

In [47]:
```
result=clf.predict(testd)
```

In [48]:
```
print(result)
```

```
[2 0 9 ... 3 9 2]
```

Choosing the 100th number in the test set so see the variance caused by the K.

In [9]:
```
img_100 = testd.iloc[99:100,:]
```

In [86]:
```
# k=5
result1 = clf.predict(img_100)
print(result1)
```

```
[4]
```

In [87]:
```
# k=9
from sklearn.neighbors import KNeighborsClassifier
clf=KNeighborsClassifier(n_neighbors = 9, algorithm = 'brute', p = 2, metric = 'euclidean')
clf.fit(train_images,train_labels.values.ravel())
result2 = clf.predict(img_100)
print(result2)
```

```
[4]
```

In [88]:
```
# k=3
from sklearn.neighbors import KNeighborsClassifier
clf=KNeighborsClassifier(n_neighbors = 3, algorithm = 'brute', p = 2, metric = 'euclidean')
clf.fit(train_images,train_labels.values.ravel())
result3 = clf.predict(img_100)
print(result3)
```

```
[4]
```

In [89]:
```python
# k=11
from sklearn.neighbors import KNeighborsClassifier
clf=KNeighborsClassifier(n_neighbors = 9, algorithm = 'brute', p = 2, metric = 'euclidean')
clf.fit(train_images,train_labels.values.ravel())
result4 = clf.predict(img_100)
print(result4)
```

```
[4]
```

It turns out that the 100th number is predicted as 4 when k=5,9,3,11. The accuracy is fine.

In [50]:
```python
# Output the result as .csv file
df=pd.DataFrame(result)
df.index.name='ImageId'
df.index+=1
df.columns=['Label']
df.to_csv('results.csv',header=True)
```

4. Runtime

For d dimension, we need O(d) runtime to compute one distance between two data, so computing all the distance between one data to other data needs O(nd) runtime, then we need O(kn) runtime to find the K nearest neibors, so, in total, it takes O(dn+kn) runtime for the classifier to classify the data.

In [10]:
```python
import time
start = time.time()
clf=KNeighborsClassifier(n_neighbors = 5, algorithm = 'brute', p = 2, metric = 'euclidean')
clf.fit(train_images,train_labels.values.ravel())
result=clf.predict(testd)
end = time.time()
print(end-start)
```

```
158.65635895729065
```

As is shown above, the "wall-clock" of the runtime is about 158.66s