Exercise 1 Report

March 12, 2017

Student's name: Ngo Minh Thang. Student's number: 608820

Goal: Learn to extract features and produce a classifier and assess its quality on informal level.

In [1]:
```
# Some magic function of jupyter notebook
%load_ext autoreload
%autoreload 2
%matplotlib inline
```

1 Load the data and vectorization

The first task is simply to load the data. At first I didn't realize that the digits are numbers from 0 to 9, and the files's name contain the label itself. How silly of me!

I define a function that read the data from the train[0-9].txt and test[0-9].txt files. It then combine them into one single array_X which stores the data itself, and array_Y which stores the labels.

In [2]:
```
import numpy as np
```

```
def load_data(dataset):
    '''A helper function to load the data and
        store them into numpy arrays'''
    array_X = []
    array_Y = []
    for i in range(10):
        if dataset == 'train':
            instance_X = np.loadtxt('dataset/train%d.txt' % i)
        elif dataset == 'test':
            instance_X = np.loadtxt('dataset/test%d.txt' % i)
        else:
            raise ValueError("have to be 'train' or 'test' dataset!")
        instance_Y = np.empty([instance_X.shape[0], 1])
        instance_Y.fill(i)
        array_X.append(instance_X)
        array_Y.append(instance_Y)
    array_X = np.concatenate(array_X, axis=0)
    array_Y = np.concatenate(array_Y, axis=0)
    return array_X, array_Y
```

In [3]:
```
from helpers.load_data_helper import load_data


# load the data into numpy arrays
%time train_X, train_Y = load_data('train')
%time test_X, test_Y = load_data('test')
```

```
Wall time: 1min
Wall time: 11.3 s
```

Next I define some functions to create the original image from one sample and then display it.

In [4]:
```
from IPython.display import Image, display
import numpy as np
import scipy.misc


def create_image(array):
    '''Helper function to create the original image from a sample'''
    if array.shape == (784,):
        img_arr = np.reshape(array, (28, 28))
    else:
        raise ValueError('the array have to be size (784,)!')
    scipy.misc.imsave('figures/character.jpg' , img_arr)
    image_name = 'figures/character.jpg'
    return image_name


def display_image(image_name):
    '''Display the image'''
    display(Image(filename=image_name))
```

In [5]:
```
from helpers.plotting_helper import create_image, display_image


# create image from the train sample number 20001, then display it
image = create_image(train_X[20000, :])
display_image(image)
```

In [6]:
```
# the test sample number 9001
image = create_image(test_X[9000, :])
display_image(image)
```

2 Mini essay

Beautifying the digits: If we "beautifying" the digits, I guess that it will make the "style" of each writer more prominent. This will make samples written by the same writer but different classes(digits) to be more alike, while samples in the same class will have more variety. The visualization will have a hard time partitioning correctly between samples from same classes.

Skeleton line matching: On the other hand, if we use a skeleton line matching that match every samples with only its "skeleton" structure of the digits, we are doing something "reversed" from the above method. Here we strip away all the "quirks" of the different writers, hence reduce the variety between same-label samples and increase the "distance" between samples that are not from the same classes. So the visualized clusters will be more clearly defined, same-class samples will be so close that each cluster(each class) will be just one single "point", and each cluster will be very far from other clusters.

3 Feature Extraction

I use the second method, where I reduce the bits to d = 14x14. For each sample(row), I take the average value of every 4 bits as the new bit value. So each sample vector will have 196 instead of 784 features.

```
In [7]: import numpy as np

        def feature_extraction(array):
            '''Get a 196-features dataset'''
            new_samples = []
            for i in range(array.shape[0]):
                new_features = []
                for j in range(0, array.shape[1], 4):
                    new_value = array[i, range(j, j + 4)].mean()
                    new_features.append(new_value)
                new_features = np.array(new_features)
                new_samples.append(new_features)
            array = np.array(new_samples)
            return array
```

```
In [8]: from helpers.feature_extraction import feature_extraction


        # Do feature extraction on both train and test data
        %time train_X_extracted = feature_extraction(train_X)
        %time test_X_extracted = feature_extraction(test_X)
```

```
Wall time: 3min 30s
Wall time: 43.2 s
```

```
In [9]: def create_image_extracted(array):
            '''Create image from samples after feature extraction'''
            if array.shape == (196,):
                img_arr = np.reshape(array, (14, 14))
            else:
                raise ValueError('the array have to be size (196,)!')
            scipy.misc.imsave('figures/character.jpg' , img_arr)
            image_name = 'figures/character.jpg'
            return image_name
```

Finally, I create images from the same samples before feature extraction above.

```
In [10]: from helpers.plotting_helper import create_image_extracted

         image = create_image_extracted(train_X_extracted[20000, :])
         display_image(image)
```

```
In [11]: image = create_image_extracted(test_X_extracted[9000, :])
         display_image(image)
```

This method make the images a lot more blurry and also "double" the digit itself!

4 Classification

I use Random Forest model as my classifier. Random Forest is an ensemble learning method that construct a multitude of decision trees at training time and outputting the class that is the mode of the classes of the individual trees. I take advantage of the algorithm's implementation from scikit-learn.

```
In [12]: from sklearn.ensemble import RandomForestClassifier
         from sklearn.metrics import confusion_matrix, accuracy_score


         # reshape the labels array
         train_Y = train_Y.reshape([train_Y.shape[0],])
```

```
test_Y = test_Y.reshape([test_Y.shape[0],])

# make model, fit the train set, then predict the test set
model = RandomForestClassifier(n_estimators=1000,
                               n_jobs=-1, min_samples_leaf=50)
%time model.fit(train_X_extracted, train_Y)
test_Y_pred = model.predict(test_X_extracted)

# print the classification accuracy on the test set
print('Accuracy ACC = ', accuracy_score(test_Y, test_Y_pred))
```

```
Wall time: 1min 23s
Accuracy ACC =  0.941
```

The classification accuracy is very high, no doubt because of the optimized implementation from scikit-learn. I guess a deep learning approach should yield an even greater accuracy!

In [13]:
```
# compute confusion matrix
conf_matrix = confusion_matrix(test_Y, test_Y_pred)
conf_matrix
```

Out[13]:
```
array([[ 967,    0,    1,    0,    0,    1,    3,    1,    6,    1],
       [   0, 1112,    5,    2,    1,    2,    3,    0,   10,    0],
       [  11,    1,  975,    9,    4,    2,   10,   10,    9,    1],
       [   2,    0,   14,  931,    0,   21,    1,   20,   14,    7],
       [   2,    0,    1,    2,  921,    1,    8,    0,    8,   39],
       [   4,    9,    4,   29,    4,  821,    7,    3,    7,    4],
       [  10,    4,    4,    0,    5,    5,  926,    0,    4,    0],
       [   0,   10,   24,    2,    4,    2,    0,  951,    4,   31],
       [   9,    3,    6,    6,   12,    7,   12,    8,  895,   16],
       [   7,    6,    1,   12,   47,    5,    1,    7,   12,  911]])
```

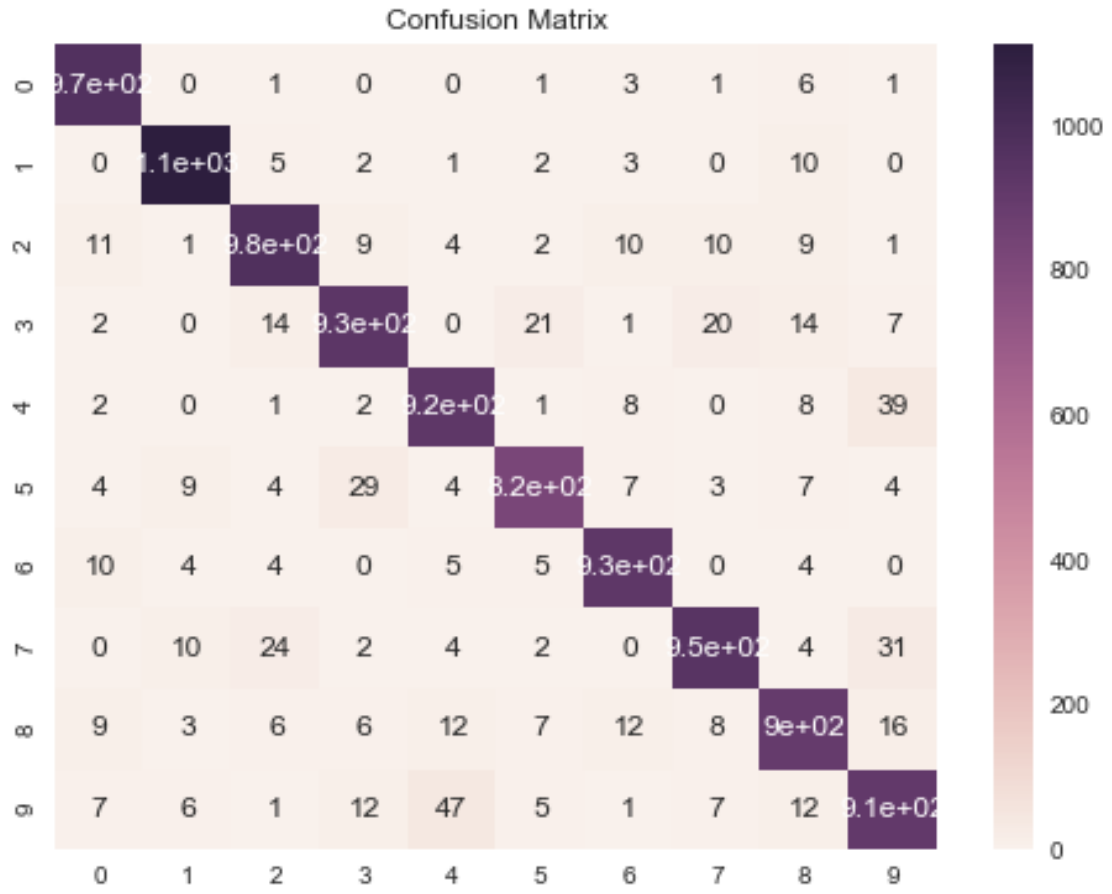
I draw a heatmap to have a more visually appealling confusion matrix.

In [14]:
```
import seaborn as sn
import matplotlib.pyplot as plt
import pandas as pd


def plot_confusion_matrix(conf_matrix, labels):
    '''Make a heatmap on confusion matrix'''
    df_conf_matrix = pd.DataFrame(conf_matrix,
                                  index=[i for i in labels], columns=[i fo
    plt.figure(figsize=(8, 6))
    sn.heatmap(df_conf_matrix, annot=True)
    plt.title('Confusion Matrix')
```

In [15]:
```
from helpers.plotting_helper import plot_confusion_matrix
```

```
labels = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
plot_confusion_matrix(conf_matrix, labels)
```



And finally, I make images of some of the misclassified samples.

```
In [16]: # index of misclassified samples
         wrongly_classified = np.where(test_Y_pred != test_Y)

In [17]: # visualize a wrongly predicted sample
         for i in (0,15, 66, 130, 200, 250, 330, 370, 440, 500):
             image = create_image(test_X[wrongly_classified[0][i], :])
             display_image(image)
             print ('It was wrongly predicted as class %d'
                    % int(test_Y_pred[wrongly_classified[0][i]]))
```

It was wrongly predicted as class 6

It was wrongly predicted as class 2

It was wrongly predicted as class 4

It was wrongly predicted as class 2

It was wrongly predicted as class 6

It was wrongly predicted as class 7

```
It was wrongly predicted as class 8
```

```
It was wrongly predicted as class 3
```

```
It was wrongly predicted as class 0
```

```
It was wrongly predicted as class 3
```

Some additional thoughts: Overall, this assignment was very difficult to me, especially the mini-essay and the feature extraction section. But it was very interesting, and I had some more great resources to continue learning!