LAB 7: Machine Learning using the Scikit-learn and the MNIST dataset
Part A

Yanling Wu
yw996

November 15, 2018

0.1 1. Data Set Creation

Step 1

I wrote down 50 digit numbers and 10 different characters and took pictures of them.

Step 2

Processed images that I took pictures in Visionx, the main steps are shown below:

1. Convert the images of png version to the vx format by running "vfmt -g -vx if=filename of=filename.vx";
2. Filter the images and threshold the images to binary images which only contain the background and foreground. As for my images, I used the code that I wrote and chose the threshold of 100 to filter and threshold the image to binary image. After compiling the C code, run the command "vits th=100 if=filename of=filename_th.vx" to process the images;
3. Generate a 20 * 20 bounding box that contain only the character and resize the square box into 20 * 20 by running "vimag s=20,20,1 if=filename_th.vx of=filename_20.vx" for each image;
4. After execution the command above, the images will become grey images not binary images, we need to threshold the images again. This time, the threshold of 175 can make a great binary images by running the command "vits th=175 if=filename_20.vx of=filename_20th.vx";
5. Invert the images' pixel value to make the background dark and the foreground white by using the command "vpix -neg if=filename_20th.vx of=filename_20thv.vx";
6. Calculate the COM of the images whose size are 20*20 and move the COM to the center of the images of 28*28.* The formular of calculating COM is $\bar{x} = \frac{m_{00}}{m_{10}}$ and $\bar{y} = \frac{m_{00}}{m_{01}}$. Then, in 28*28 *images, move all of the pixel values of the original 20*20 images for $ (14 -, 14-) $, which is the main idea of my C code of the COM.c. Run the command of "com if=filename_20thv.vx of=filename.vx";
7. Export the .png file from VX using vxport.

For the conveniently executing above functions, I created a bash file called *tran* to run all the command one time.

0.2 Appendix

0.2.1 vit.c

In [ ]:
```
/*********************************************************************/
/* vits:     Iterative Threshold Selection Algorithm              */
/*@ Copyright                                                     */
/*  Name: Yanling Wu                                              */
/*  NetID: yw996                                                  */
/*  ECE 5470 Lab7                                                 */
/*********************************************************************/

#include "VisXV4.h"          /* VisionX structure include file       */
#include "Vutil.h"           /* VisionX utility header files         */

VXparam_t par[] =            /* command line structure               */
{
{  "if=",   0,   " input file, vtpeak: threshold between hgram peaks"},
{  "of=",   0,   " output file "},
{  "th=",    0,   " the initial threshold (default the average of all pixel values)"},
{  "-v",    0,   "(verbose) print threshold information"},
{   0,      0,   0} /* list termination */
};
#define  IVAL   par[0].val
#define  OVAL   par[1].val
#define  TVAL   par[2].val
#define  VFLAG  par[3].val

main(argc, argv)
int argc;
char *argv[];
{

    Vfstruct (im);                 /* input image structure          */
    int y,x;                       /* index counters                 */
    int i;
    int thresh;                    /* threshold                      */
    int orithresh = 100;              /* the original threshold   */
    int temp;
    int count = 0;


    VXparse(&argc, &argv, par);    /* parse the command line         */

    if (TVAL) orithresh = atoi(TVAL);
    if (orithresh < 0 || orithresh > 255) {
        fprintf(stderr, "d= must be between 0 and 255\nUsing d=10\n");
        orithresh = 100;
    }
        fprintf(stderr, "orithresh = %d\n", orithresh);

    while ( Vfread( &im, IVAL) ) {
        if ( im.type != VX_PBYTE ) {
              fprintf (stderr, "error: image not byte type\n");
              exit (1);
      }

        thresh = orithresh;
        fprintf(stderr, "thresh = %d\n", thresh);

        if(VFLAG)
             fprintf(stderr, "thresh = %d\n", thresh);

        /* apply the threshold */
        for (y = im.ylo; y <= im.yhi; y++) {
            for (x = im.xlo; x <= im.xhi; x++) {
                 if (im.u[y][x] >= thresh) im.u[y][x] = 255;
                 else                      im.u[y][x] = 0;
            }
        }

        Vfwrite( &im, OVAL);
    } /* end of every frame section */
    exit(0);
}
```

0.2.2 COM.c

In [ ]:
```
/*******************************************************************/
/* vcorner return the left lower corner of a byte image            */
/*@ Copyright                                                      *
/*  Name: Yanling Wu                                               */
/*  NetID: yw996                                                   */
/*  ECE 5470 Lab7
/*******************************************************************/
#include "VisXV4.h"        /* VisionX structure include file    */
#include "Vutil.h"        /* VisionX utility header files      */
VXparam_t par[] =             /* command line structure            */
{ /* prefix, value,   description                         */
{    "if=",    0,   " input file  vcorner: copy lower left corner"},
{    "of=",    0,   " output file "},
{     0,       0,    0}  /* list termination */
};
#define IVAL par[0].val
#define OVAL par[1].val
```

```
int main(argc, argv)
int argc;
char *argv[];
{

        Vfstruct(im);                      /* i/o image structure          */
        Vfstruct(tm);                      /* temp image structure         */
        int y,x;                     /* index counters                 */
        int s;          /* corner image size             */
        int area = 0, x_move = 0, y_move = 0;
        int mx = 0, my = 0;
        VXparse(&argc, &argv, par);       /* parse the command line       */
        /* create VX image for result */
        s = 28;  /* set size of result image */
        Vfnewim(&tm, VX_PBYTE, bbx, 1);
        while ( Vfread(&im, IVAL) ) {     /* read image file              */
                if ( im.type != VX_PBYTE ) {    /* check image format            */
                        fprintf(stderr, "vcorner: no byte image data in input file\n");
                        exit(-1);
                }
                /* check that the input image is large enough */
        /*        if ( (im.xhi - im.xlo ) < (tm.xhi - tm.xlo) ||
            (im.yhi - im.ylo ) < (tm.yhi - tm.ylo) ) {
                        fprintf(stderr, "vcorner: input image too small\n");
                        exit(-1);
                }*/
                //compute the original COM position
                for (y = im.ylo; y <= im.yhi; y++) {
                        for(x = im.xlo; x <= im.xhi; x++) {
                                if (im.u[y][x] != 0) {
                                        area += 1;
                                        mx += x;
                                        my += y;
                                }
                        }
                }
                fprintf(stderr, "xmean=%d,ymean=%d\n", mx/area,my/area);

                x_move = 14 - mx / area ;
                y_move = 14 - my / area;
                fprintf(stderr, "x_move=%d, y_move=%d\n",x_move,y_move);

                for (y = im.ylo; y <= im.yhi; y++) {  /* compute the function */
                        for(x = im.xlo; x <= im.xhi; x++)  {
                                tm.u[y+y_move][x+x_move] = im.u[y][x];
                        }
                }
                Vfwrite(&tm, OVAL);             /* write image file              */
        }
        exit(0);
}
```

0.2.3 trans (bash)

```
In [ ]: #!/bin/sh
        # script to process the pictures to get the data set
        #i Syntax:  trans <image-file>
        ## Returns: the processed images in <image-file>.vx
        for i in 1 2 3 4 5
        do
                fname=$1.$i.PNG
                oname=$fname.vx
                tmp=tmp.$$.vx
                tmp2=tmp.$$_th.vx
                tmp3=tmp2.$$_20.vx
                if [ "$#" != "1" ]
                then
                echo "Error: a single file name argument is required"
                exit
                fi

                rm -f $oname
                vfmt -g -vx if=$fname of=$tmp
                vits th=100 if=$tmp of=$tmp2
                vimag s=20,20,1 if=$tmp2 of=$tmp3
                vits th=175 if=$tmp3 of=$tmp
                vpix -neg if=$tmp of=$tmp2
                com if=$tmp2 of=$tmp3
                vxport -png if=$tmp3 of=$oname
                rm -f $tmp
                rm -f $tmp2
                rm -f $tmp3
        done
```

0.3 2. Experiments with Logistic Regression Classifier

0.3.1 Step 1.

For this section, we need to use logistic regression classifier to train and test the standard test mnist dataset and the dataset we created. The below code is to realize this goal. First, we download the mnist dataset and use them to train the logistic regression model. Then, output the first forty test images and their labels as well as their predicted results.

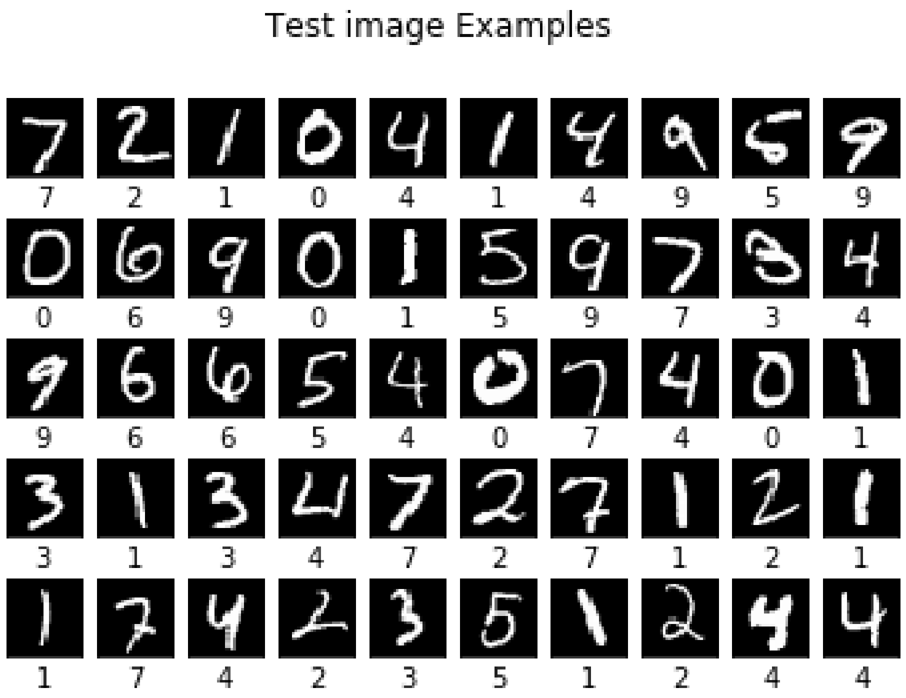
```
In [1]: #1 import required modules
        import time
        import io
        import matplotlib.pyplot as plt
        import numpy as np
        from scipy.io.arff import loadarff

        from sklearn.datasets import get_data_home
        from sklearn.externals.joblib import Memory
        from sklearn.linear_model import LogisticRegression
        from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import StandardScaler
        from sklearn.utils import check_random_state
        from urllib.request import urlopen
```

```
In [2]: #2.  read the NMIST dataset
        memory = Memory(get_data_home())
        @memory.cache()
        def fetch_mnist():
            content = urlopen(
                'https://www.openml.org/data/download/52667/mnist_784.arff').read()
            data, meta = loadarff(io.StringIO(content.decode('utf8')))
            data = data.view([('pixels', '<f8', 784), ('class', '|S1')])
            return data['pixels'], data['class']
        X, y = fetch_mnist()
```

```
In [3]: #3 rescale the data, use the traditional train/test split
        X = X / 255.
        # NEW  Refromat the the labels to be integers rather than byte arrays
        y_trans = []
        for i in range(len(y)):
            y_trans.append(int(y[i]))
        y = np.asarray(y_trans)

        X_train, X_test = X[:60000], X[60000:]
        y_train, y_test = y[:60000], y[60000:]
```

Comment:
This is a good point and we need to convert the byte format of y into int format, which is important since we will calculate the socre of predicting results. If there is no force conversion, the score will be zero.

```
In [4]: ### Bounus 1: show the first 40 images
        ### ALways a good idea to validate that the data appears as you expect
        ###
        ### for sci-kit learn the images are represented as vectors of 784 elements
        ### currently scaled from 0 to 1

        for i in range(50):
            l1_plot = plt.subplot(5, 10, i + 1)
            l1_plot.imshow(255 * X_test[i].reshape(28, 28), interpolation='nearest',
                           cmap=plt.cm.gray)
            l1_plot.set_xticks(())
            l1_plot.set_yticks(())
            #l1_plot.set_xlabel('Class %s' % y_test[i].decode())
            l1_plot.set_xlabel('%i' % int(y_test[i]))
        plt.suptitle('Test image Examples')
        plt.show()
```



```
In [5]: ## Data standardization
        ## by the mean and standared deviation of the training set
        scaler = StandardScaler()
        X_train = scaler.fit_transform(X_train)
        X_test = scaler.transform(X_test)
```

Comment:

Standardize features by removing the mean and scaling to unit variance. Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set. Mean and standard deviation are then stored to be used on later data using the transform method.

Standardization of a dataset is a common requirement for most of machine learning estimtors since they might behave badly if the individual features do not look like standard normally distributed data such as Gaussian with mean and unit variance.

The following code can help us to show the images after standardization.

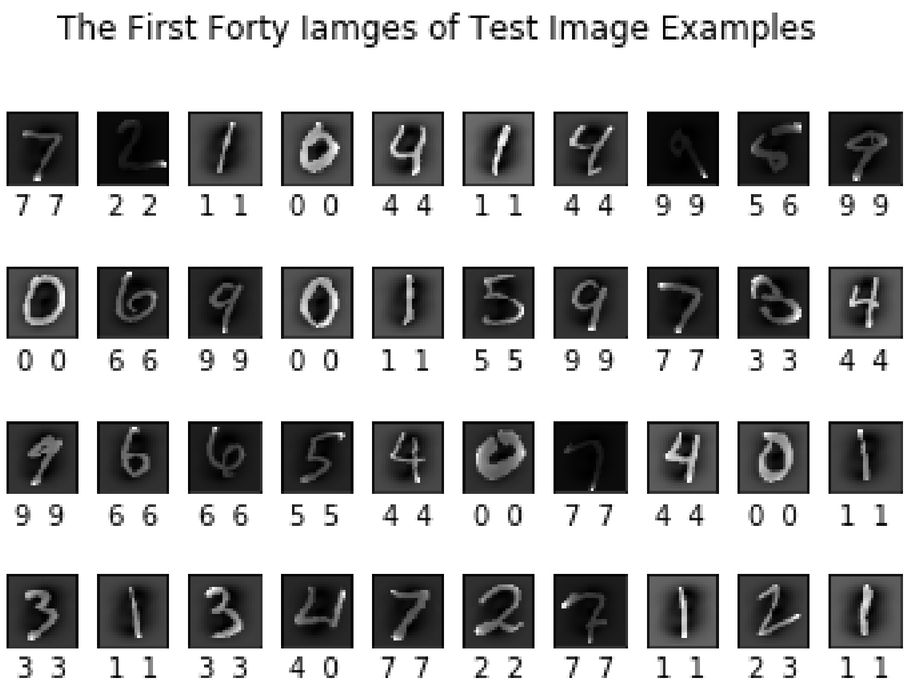
```
In [7]: #Show the first forty images after Standardization
        for i in range(50):
            l1_plot = plt.subplot(5, 10, i + 1)
            l1_plot.imshow(255 * X_test[i].reshape(28, 28), interpolation='nearest',
                           cmap=plt.cm.gray)
            l1_plot.set_xticks(())
            l1_plot.set_yticks(())
            #l1_plot.set_xlabel('Class %s' % y_test[i].decode())
            l1_plot.set_xlabel('%i' % int(y_test[i]))
        plt.suptitle('Test image Examples after Standardization')
        plt.show()
```

Test image Examples after Standardization
7 2 1 0 4 1 4 9 5 9
0 6 9 0 1 5 9 7 3 4
9 6 6 5 4 0 7 4 0 1
3 1 3 4 7 2 7 1 2 1
1 7 4 2 3 5 1 2 4 4

```
In [8]: #train and test classifier
        # Turn up tolerance for faster convergence
        clf = LogisticRegression(C=50. / 1000,
                                 multi_class='multinomial',
                                 penalty='l1', solver='saga', tol=0.1)
        # Train the classifier
        clf.fit(X_train, y_train)

        #Evaluate the classifier
        sparsity = np.mean(clf.coef_ == 0) * 100
        score = clf.score(X_test, y_test)
        # print('Best C % .4f' % clf.C_)
        print("Sparsity with L1 penalty: %.2f%%" % sparsity)
        print("Test score with L1 penalty: %.4f" % score)
```

```
Sparsity with L1 penalty: 17.24%
Test score with L1 penalty: 0.9021
```

```
In [9]: #show the first forty images results
        y_predict = clf.predict(X_test)
        for i in range(40):
            l1_plot = plt.subplot(4, 10, i + 1)
            plt.subplots_adjust(wspace = .3, hspace = .3)
            l1_plot.imshow(255 * X_test[i].reshape(28, 28), interpolation='nearest',
                           cmap=plt.cm.gray)
            l1_plot.set_xticks(())
            l1_plot.set_yticks(())
            #l1_plot.set_xlabel('Class %s' % y_test[i].decode())
            l1_plot.set_xlabel('%i  %i' % (int(y_test[i]), int(y_predict[i])))
        plt.suptitle('The First Forty Iamges of Test Image Examples')
        plt.show()
```



** Discussion: **

For this section, we trained the logistic regression classifier using mnist dataset. The first forty images shows above. The labels and the predicted labels show below the images. We can see most of the predicted results are correct since the score of the prediction is 0.9021, which is pretty high. Besides, the recognition results have relationship with how the digits wrote, meaning if a digit wrote commonly and standardedly, it will be easier to recognize correstly.

0.3.2 Step 2.

For this step, we will test the dataset that we created in the first part. And before loading the dataset, we need to create a csv format file to store the names and labels of images. Because of 60 images, I wrote the code to help me to create this file. And create a funtion to help us to read this csv file and load the images.

```
In [23]: #Create the csv file to store the names and labels
         import csv
         csvFile = open("./idata/label.csv",'w', newline='')
         writer = csv.writer(csvFile)

         for i in range(0,10):
             for j in range(1,6):
                 name = []
                 name_str = str(i)+"."+str(j)+".PNG.vx.png"
                 name.append(name_str)
                 name.append(i)
                 writer.writerow(name)
                 del name
         count = 10
         for i in range(10,12):
             for j in range(1,6):
                 name = []
                 name_str = str(i)+"."+str(j)+".PNG.vx.png"
                 name.append(name_str)
                 name.append(count)
                 count += 1
                 writer.writerow(name)
                 del name

         csvFile.close()


         csvFile = open("./idata/label.csv")
         read = csv.reader(csvFile)
         # for line in read:
         #     print (line)
         csvFile.close()
```

```
In [24]: """ Custom datatset loader
             based on https://github.com/utkuozbulak/pytorch-custom-dataset-examples
         """
         import pandas as pd
         import imageio

         class SimpleDataset():
             def __init__(self, data_path, csv_name, transform = None ):
                 """
```

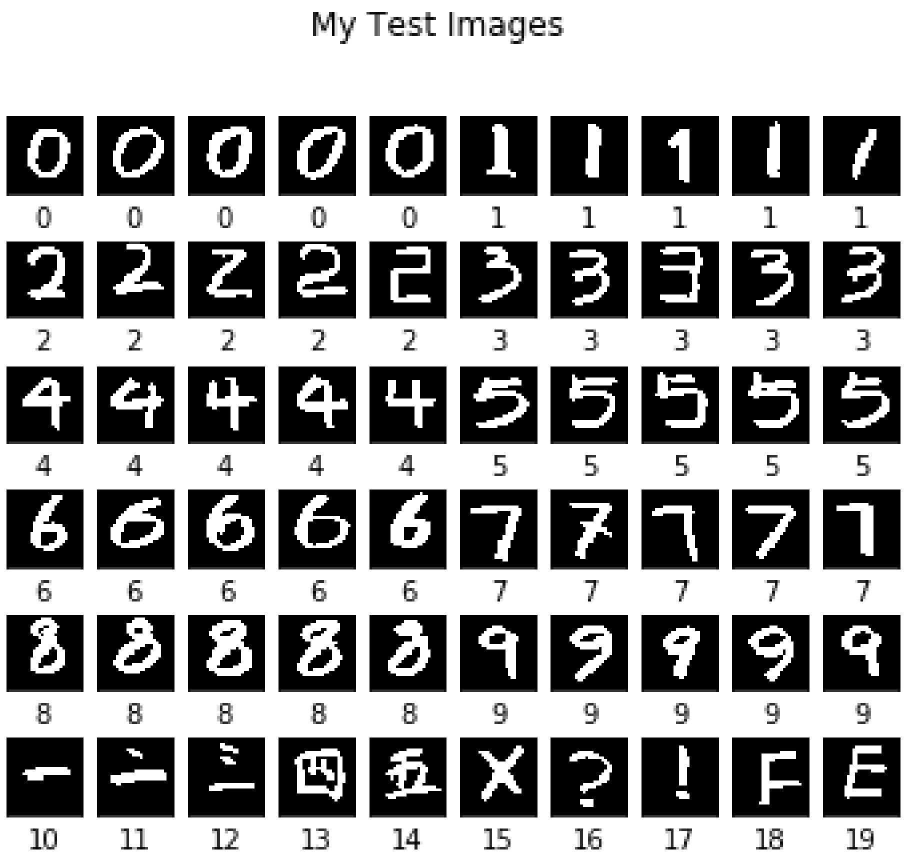
```
                Args:
                    data_path (string): path to the folder where images and csv files are locat
                    csv_name (string): name of the csv lablel file
                    transform: pytorch transforms for transforms and tensor conversion
                """
                # Set path
                self.data_path = data_path
                # Read the csv file
                self.data_info = pd.read_csv(data_path + csv_name, header=None)
                # First column contains the image paths
                self.image_arr = np.asarray(self.data_info.iloc[:, 0])
                # Second column is the labels
                self.label_arr = np.asarray(self.data_info.iloc[:, 1])
                # Calculate len
                self.data_len = len(self.data_info.index)

            def __getitem__(self, index):
                # Get image name from the pandas df
                single_image_name = self.image_arr[index]
                # Open image
                img_as_img = imageio.imread(self.data_path + single_image_name)

                # Get label(class) of the image based on the cropped pandas column
                single_image_label = self.label_arr[index]

                return (img_as_img, single_image_label)

            def __len__(self):
                return self.data_len
```

```
In [25]: #read the image file
         mydata = SimpleDataset( "./idata/", "label.csv")
         #splitting into images and labels
         X = []
         y = []
         for i in range(len(mydata)):
             X.append(mydata[i][0])
             y.append((mydata[i][1]))
         #converting into numpy arrays to enable easy reshaping and other array operations
         X = np.asarray(X)
         print("Shape of the input image", X.shape)
         y= np.asarray(y)
```

```
Shape of the input image (60, 28, 28)
```

```
In [26]: #show the images of created dataset
```

```
import warnings
warnings.filterwarnings('ignore')
#l1_plot = plt.subplot(0, 5, i + 1)
plt.figure(figsize=(6,5))
for i in range(60):
    img_plot = plt.subplot(6, 10, i + 1)
    #plt.subplots_adjust(wspace = .3, hspace = .5)
    img_plot.imshow(255 * X[i].reshape(28, 28), interpolation="nearest",
                    cmap=plt.cm.gray)
    img_plot.set_xticks(())
    img_plot.set_yticks(())
    img_plot.set_xlabel('%i' % int(y[i]))
plt.suptitle('My Test Images')
plt.show()
```

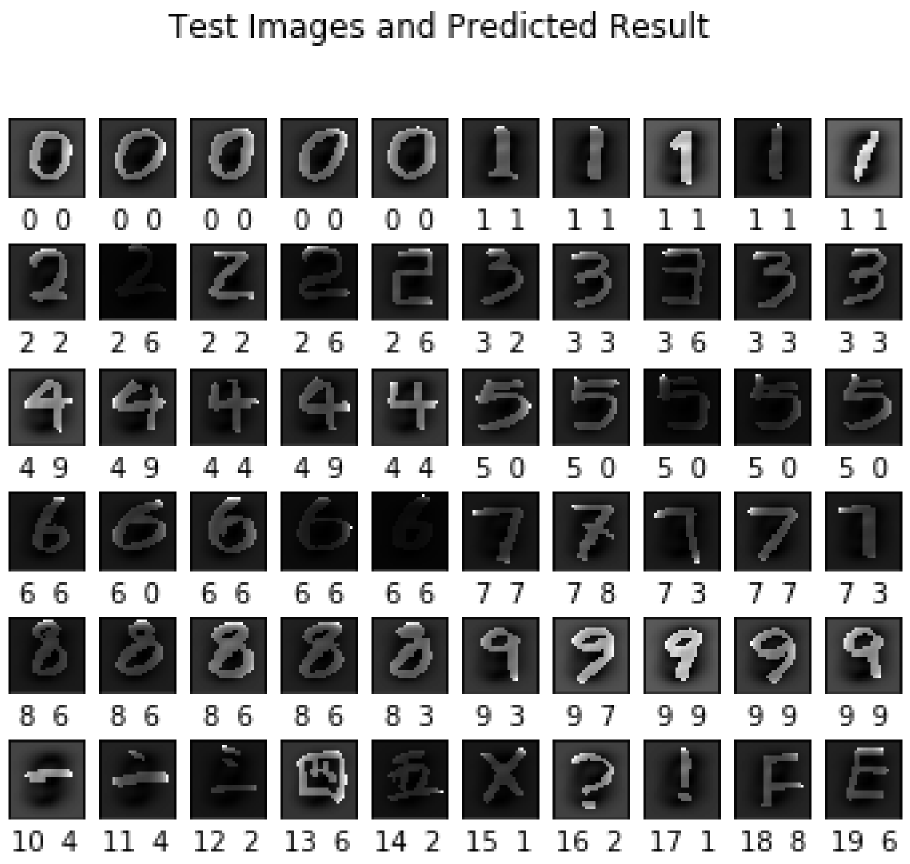


```
In [27]: #reshaping the array into flattened 784 array as an input for prediciton by the logisti
         X = X.reshape(X.shape[0], 784)
         X = X / 255.
         #data standardiation with the training set statistics is required for this clasifier
         X = scaler.transform(X)
```

```
        y_pred = clf.predict(X)

        score = clf.score(X, y)

        print("Test score with L1 penalty: %.4f" % score)
        print("y_predicted_values", y_pred)
        print("y_labels", y)
```

```
Test score with L1 penalty: 0.4333
y_predicted_values [0 0 0 0 0 1 1 1 1 1 2 6 2 6 6 2 3 6 3 3 9 9 4 9 4 0 0 0 0 0 6 0 6 6 6 7 8
 3 7 3 6 6 6 6 3 3 7 9 9 9 4 4 2 6 2 1 2 1 8 6]
y_labels [ 0  0  0  0  0  1  1  1  1  1  2  2  2  2  2  3  3  3  3  3  4  4  4  4
  4  5  5  5  5  5  6  6  6  6  6  7  7  7  7  7  8  8  8  8  8  9  9  9
  9  9 10 11 12 13 14 15 16 17 18 19]
```

```
In [28]: #Show the predicted results

         warnings.filterwarnings('ignore')
         #l1_plot = plt.subplot(0, 5, i + 1)
         plt.figure(figsize=(6,5))
         for i in range(60):
             img_plot = plt.subplot(6, 10, i + 1)
            # plt.subplots_adjust(wspace = .3, hspace = .5)
             img_plot.imshow(255 * X[i].reshape(28, 28), interpolation="nearest",
                            cmap=plt.cm.gray)
             img_plot.set_xticks(())
             img_plot.set_yticks(())
             img_plot.set_xlabel('%i  %i' % (int(y[i]),int(y_pred[i])))
         plt.suptitle('Test Images and Predicted Result')
         plt.show()
```



Discussion:

All results of my test images are shown above. And the score is 0.4333, which is pretty low. Because I wrote some digits in uncommon way deliberately. Those digits that I wrote in unusual way almost are recognized wrongly and those digits that I wrote in common way are labeld correctly. Additionally, the last ten characters are actually not digits so I labeled them from eleven to nineteen, which absolutely will be recognized wrongly. All in all, my score of testing created dataset is so low.