Lab 7

# MACHINE LEARNING USING THE SCIKIT-LEARN AND THE MNIST DATASET

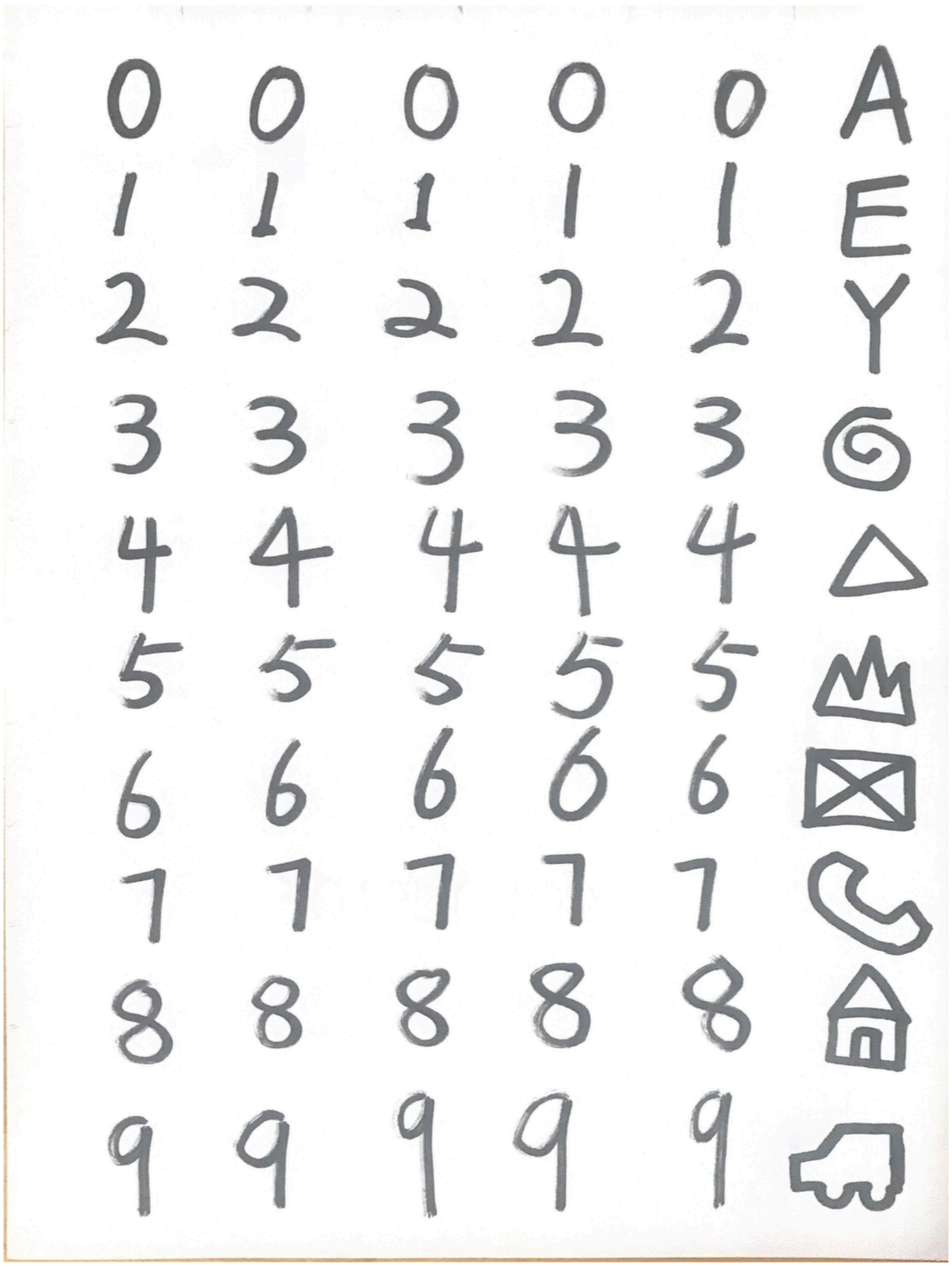
Xinyi(Blanca) Bai

Cornell University Electrical and Computer Engineering

## Table of Contents

## Part A Data Set Creation

Create 60 handwritten characters (50 of these are the digits 0-9 handwritten (5 of each) number; 10 of these to be symbols that are not the characters 0-9.



## 2. Photograph these characters into images.

I scan the whole big image, and then use screenshot to divide every character and number into a single image.

## 3. Preprocess the outline above.

First, threshold to make binary (bilevel) images and then resample to the required size and reposition according to the center of mass (COM).

The commands vfmt and vxport may be used to cornvert between VisionX image file format and other image file formats. Your final images should be stored in .png format
Upload these images to Visionx, (use vfmt to import to VX format) and then:

```
which vpix
cd home/vision/v4
ls
cd templates
```

### 3.1. Convert them to gray-level images.

I scan the image, so it is already in gray-level, we can also use

### 3.2. Threshold the images to make them containing only background & foreground (binary images).

```
/*********************************************************************/
/* threthold     Compute local max operation on a single byte image   */
/*********************************************************************/

#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  vtemp: local max filter "},
{    "of=",    0,   " output file "},
{     0,       0,   0}  /* list termination */
};
#define  IVAL   par[0].val
#define  OVAL   par[1].val

main(argc, argv)
int argc;
char *argv[];
{
Vfstruct (im);                      /* i/o image structure          */
Vfstruct (tm);                      /* temp image structure         */
int        y,x;                     /* index counters               */
  VXparse(&argc, &argv, par);       /* parse the command line       */

  Vfread(&im, IVAL);                /* read image file              */
  Vfembed(&tm, &im, 1,1,1,1);       /* image structure with border  */
  if ( im.type != VX_PBYTE ) {      /* check image format           */
     fprintf(stderr, "vtemp: no byte image data in input file\n");
     exit(-1);
  }
  for (y = im.ylo ; y <= im.yhi ; y++) {  /* compute the function */
     for (x = im.xlo; x <= im.xhi; x++)  {     /************************/
          //im.u[y][x] = MAX(tm.u[y][x],         /* You only need to     */
          //               MAX(tm.u[y][x+1],     /* change this section  */
          //                 MAX(tm.u[y+1][x],   /* for your program     */
          //                   MAX(tm.u[y][x-1], /************************/
          //                        tm.u[y-1][x]))));


          if(tm.u[y][x] >= 200 && tm.u[y][x] <= 255){
            im.u[y][x] = 255;
          }

          else {
            im.u[y][x] = 0;
          }
     }
   }
   Vfwrite(&im, OVAL);             /* write image file                */
   exit(0);
}
```

Compile and run my program

```
vcc thre.c -o thre
thre in=01.vx of=01th.vx
```

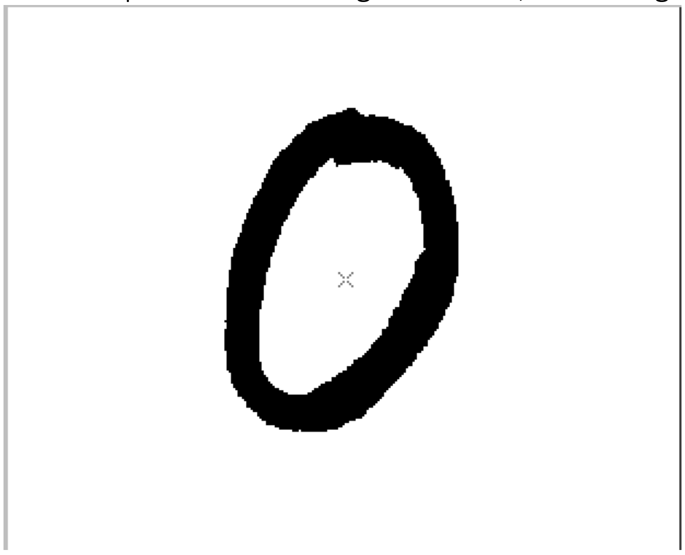
then new a file, write script to run my thre program on a batch of images(all images)

```
loopthe
```

I find I have no permission to run the script file
So I use chmod command to open all authority to that file

```
chmod 744 loopthe
```

I set the pixel value of forground as 0, the background as 255.



## 3.3. For each image, generate a 20x20 bounding box that contains only the digit

one way to do this is to first do an exhaustive search of all the foreground pixel's coordinate, and bound all of the foreground pixels with a square box; then, you resize the square box into 20x20.

I wrote a program vcom.c to calculate the central mass of a image, and use the central mass value to crop images as 400X400 size.

vcom.c file

```c
/*********************************************************************/
/* vcom return the left lower corner of a byte image            */
/*********************************************************************/
#include "VisXV4.h"
#include "Vutil.h"
VXparam_t par[] =
{ /* prefix, value,*/
{ "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[];
{
/* VisionX structure include file    */
/* VisionX utility header files      */
/* command line structure            */
/* description                         */
    Vfstruct (im);
    Vfstruct (tm);
    int y,x;
    int s;
    float bbx[6];
    int m00=0;
    int m01=0;
    int m10=0;
    int xx=0;
    int yy=0;
    VXparse(&argc, &argv, par); /* parse the command line */
    /* create VX image for result */
    s = 400; /* set size of result image */
    bbx[0] = bbx[2] = bbx[4] = bbx[5] = 0.0; bbx[1] = bbx[3] = s;
    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);
        }



        for(y = im.ylo; y <= im.yhi; y++){
            for(x = im.xlo; x <= im.xhi; x++){
            if( im.u[y][x] == 0){
                m00++;
            }
            }
        }

        for(y = im.ylo; y <= im.yhi; y++){
            for(x = im.xlo; x <= im.xhi; x++){
            if( im.u[y][x] == 0){
                m01 = x + m01;
                m10 = y + m10;
            }
            }
        }

        xx = m01 / m00;
        yy = m10 / m00;

        printf("xx = %d \n", xx);
        printf("yy = %d \n", yy);
        im.xlo = xx - 200;
        im.ylo = yy - 200;

        for(y = tm.ylo; y <= tm.yhi; y++){
            for(x = tm.xlo; x <= tm.xhi; x++){
            tm.u[y][x] = im.u[y + im.ylo][x + im.xlo];
            }
        }


        Vfwrite(&tm, OVAL);
    }
        exit(0);
}
/* write image file                */
```

Note： for imgae 54th.vx
The size of result image = 360
So I changed part of the code as below:

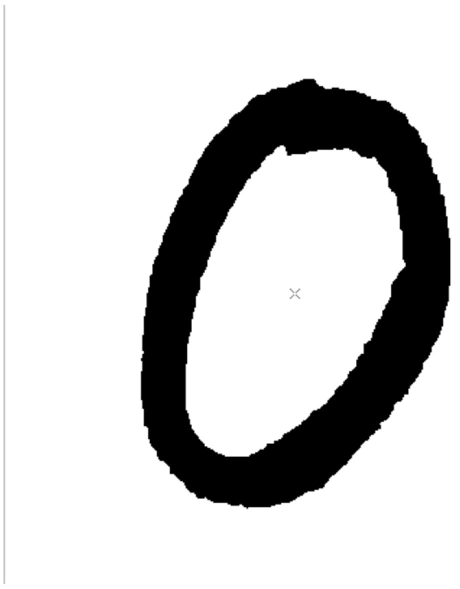
```
Im.xlo = xx - 130;
Im.ylo = yy - 130;
Vmag if=54com.vx of=54scale.vx m=0.05556
```

Compile and run my program

```
vcc vcom.c -o vcom
vcom if=02th.vx of=02com.vx
```

then new a file, write script to run my vcom program on a batch of images(all images)

```
cp loopthe loopcom
```

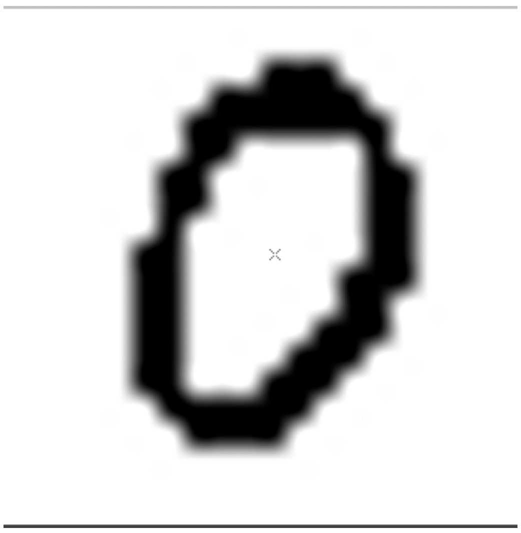


Next, scale them to 20X20 size

```
vmag if=02com.vx of=02scale.vx m=0.05
```

then new a file, write script to run vmag program on a batch of images(all images)

```
cp loopthe loopmag
```



Next, reverse the color of image.
Note: because we need to add boundary to the 20X20 images, the color of the boundary should be the same as the background. My program to add boundary, the boundary I add is black color, so I do the color reverse at this step.

```
Vpix -neg if=02scale.vx of=02inv.vx
cp loopthe loopinv
```

## 3.4. Embed the boundary of that 20x20 image with background pixels to make it 28x28.

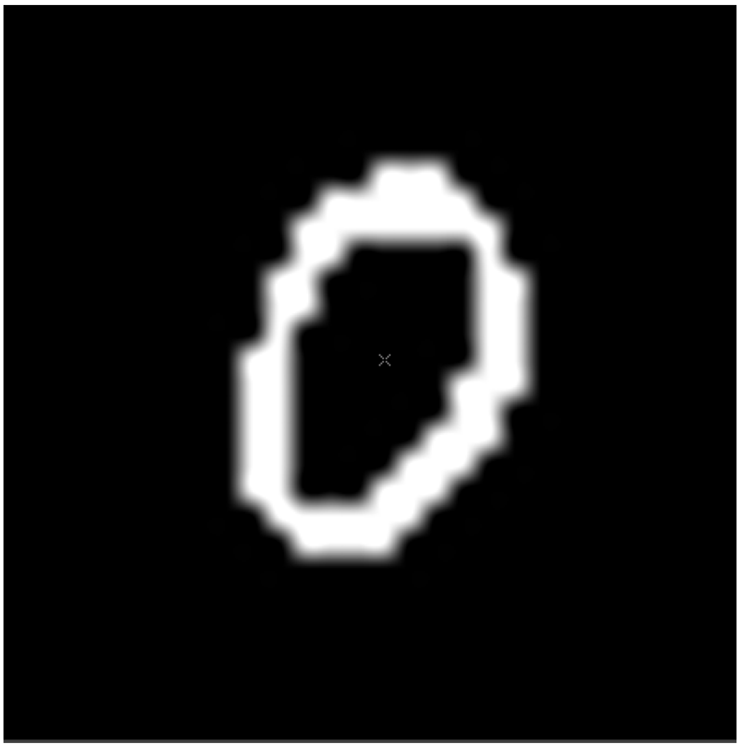
Write a program to add the boundary of the color-revered image.

```
cp thre.c vboundary.c
/*********************************************************     */
/*****************************************************************/

#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  vtemp: local max filter "},
{    "of=",    0,   " output file "},
{     0,       0,    0}  /* list termination */
};
#define  IVAL   par[0].val
#define  OVAL   par[1].val

main(argc, argv)
int argc;
char *argv[];
{
Vfstruct (im);                      /* i/o image structure          */
Vfstruct (tm);                      /* temp image structure         */
int        y,x;                     /* index counters               */
  VXparse(&argc, &argv, par);       /* parse the command line       */

  Vfread(&im, IVAL);                /* read image file              */
  Vfembed(&tm, &im, 4,4,4,4);       /* image structure with border  */
  if ( im.type != VX_PBYTE ) {      /* check image format           */
     fprintf(stderr, "vtemp: no byte image data in input file\n");
     exit(-1);
  }
```

```


    Vfwrite(&tm, OVAL);              /* write image file                */
    exit(0);
}
```

Compile and run my program, using script to add boundary on all images.

```
vcc vboundary.c -o vboundary
vboundary if=02inv.vx of=02b.vx
cp loopinc loopb
```



### 3.5 Export from VX to .png format using vxport.

Using script to convert format for all images.

```
vxport if=02b.vx of=02b -png
cp loopb looppng
```

## 4.Load the pre-processed data to scikit-learn using the python code we provided to you. By now you have the self-made train/test data for the classifier

The reason why we want you to first bound the digit in a 20x20 box before making it 28x28 is that we want to make sure that every digit you created will have the same size in the image (occupying 400/784 pixels). Assume that some of you may have very small-size handwriting, while some may write the digit really large; in that way, even if you both shift the COM to the center, the size difference of your digit will still cause trouble for the classifier to learn.

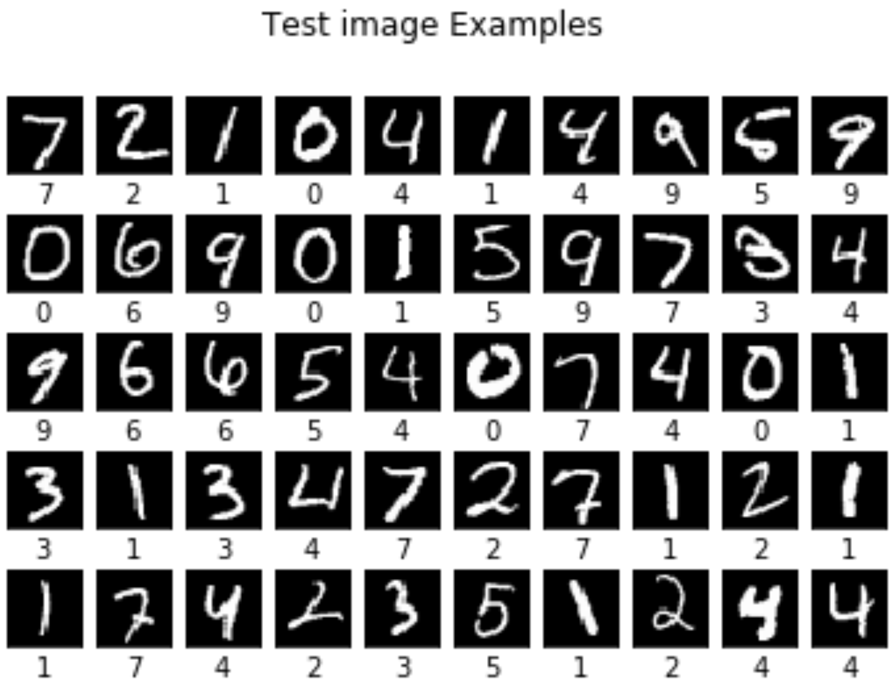
```
#1 import required modules
import time
import io
import matplotlib.pyplot as plt
import numpy as np
from scipy.io.arff import loadarff

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]:
# 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:]
```

## Bouns 1

```
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()
```



```
## 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)
```

```
In [7]:
#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: 16.35%
Test score with L1 penalty: 0.9023
```

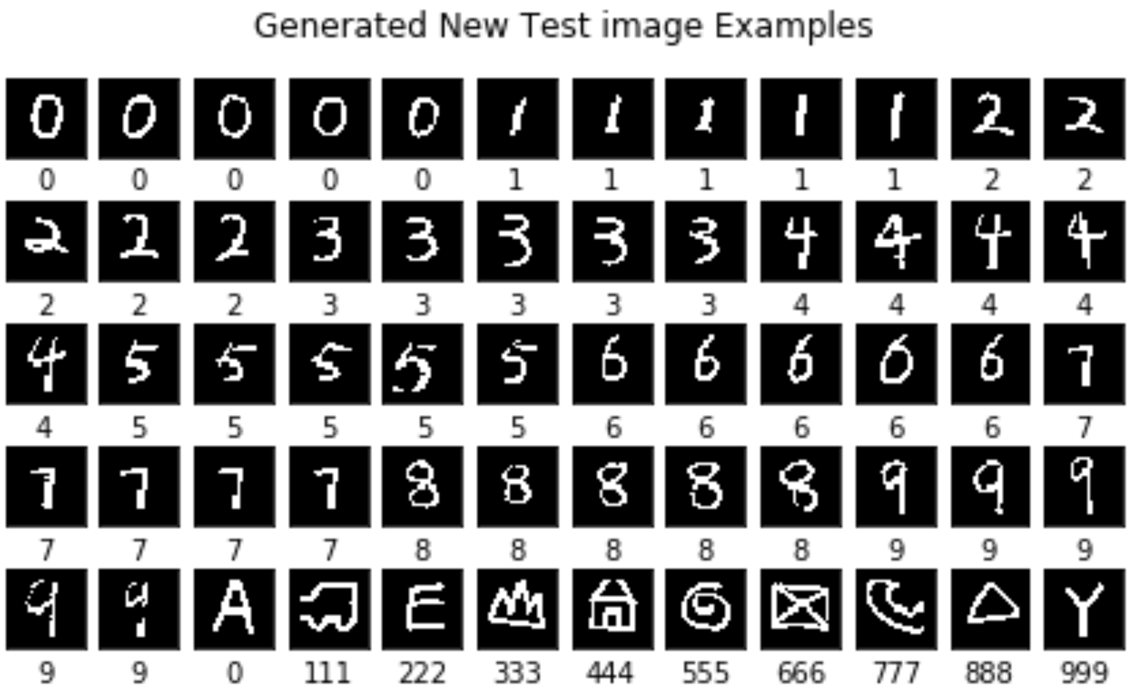
```python
""" Custom datatset loader
    based on https://github.com/utkuozbulak/pytorch-custom-dataset-examples
"""
import pandas as pd
import imageio

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 located
            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
```

```python
In [33]:
mydata = SimpleDataset( "./myDataSet/", "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)
```

```python
import warnings
```

```
warnings.filterwarnings('ignore')
#l1_plot = plt.subplot(12, 5, i + 1)
for i in range(60):
    l1_plot = plt.subplot(5, 12, i + 1)
    l1_plot.imshow(X[i], interpolation='nearest',
                   cmap=plt.cm.gray)
    l1_plot.set_xticks(())
    l1_plot.set_yticks(())
    l1_plot.set_xlabel('%i' % int(y[i]))
plt.suptitle('Generated New Test image Examples')
plt.show()
```



Note: above code are copied from my jupyter notebook.

## 5. Experiments with Logistic Regression classifier

Starting with the classifier described in the Lab tutorial evaluate the performance of the classifier for both the standard test dataset and also your own dataset.
Plot the images from the standard MNIST test set of the first 40 images with errors together with the correct and incorrect labels. Discuss these results.
Plot all your test images with truth and predicted labels. Discuss these results.

```
#reshaping the array into flattened 784 array as an input for prediciton by the log
istic regression classifier
X = X.reshape(X.shape[0], 784)
X = X / 255.
#data standardiation with the training set statistics is required for this clasifie
r
X = scaler.transform(X)

```

```python
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: 0.3000
y_predicted_values [9 9 9 9 9 1 1 1 1 1 8 8 9 2 2 3 3 3 3 3 8 9 8 8 8 8 8 9 8 8 8 8 8 6 8 1 1
 7 7 7 3 9 8 3 8 8 8 8 3 8 8 2 8 8 8 8 3 6 0 8]
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   0 111 222 333
 444 555 666 777 888 999]
```

## Discussion:

The penalty = 0.3, it is quite high and good result, the quality of my data set is high, because I use Mark pen to draw every number and character, the stroke is thick and clear, even if we scale the size of image, the outline if every number is still clear, and we use scanner to scan our image into the computer, so the background and forground is very clear even before we do threthold operation to divide the forground and background.

I also write some interesting characters such as “telephone” “mail” “house”…. At first I want to label them as its name using string datastructure, but I find in the tutorial code sample, the data structure of the label is stricted to integer, so I label them as “000” “111”…”999”, in python, “000” = “0”, so from the result, the label “000” is converted to “0” automatically.

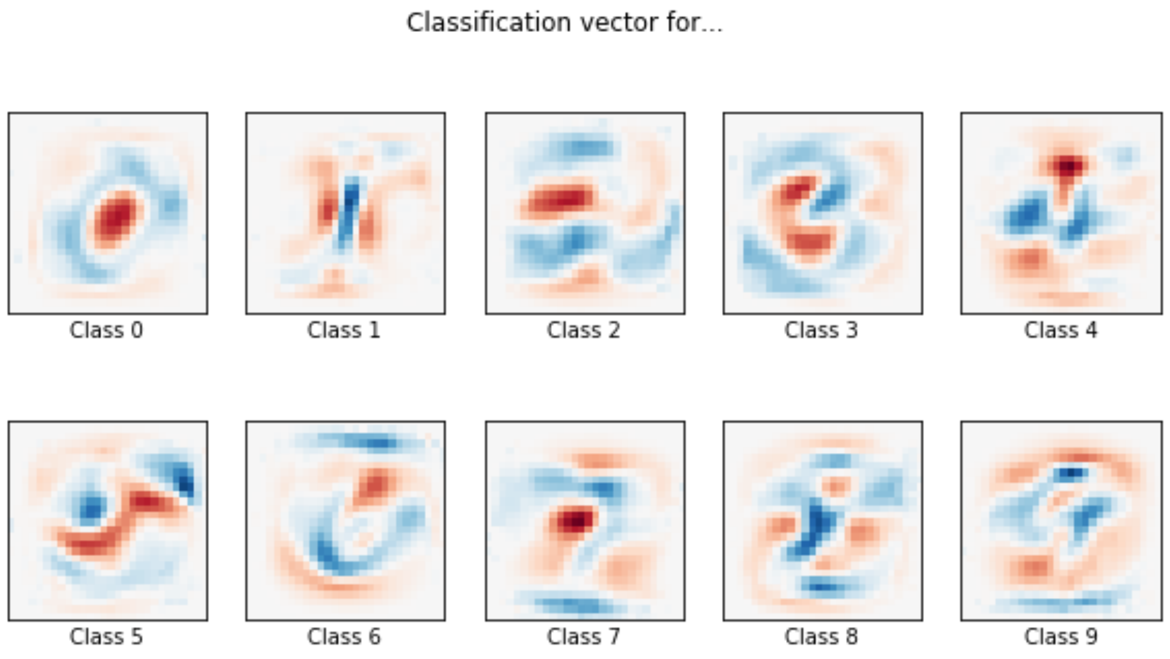
Then I put out the confusion matrix.

```python
## For analysis show also the confusion matrix

from sklearn.metrics import confusion_matrix
y_predict = clf.predict(X_test)
cfm = confusion_matrix(y_test, y_predict)
print (cfm)
```

```
[[ 957    0    1    1    0    8    9    2    2    0]
 [   0 1104    2    3    1    2    4    0   19    0]
 [  15   16  888   24   16    1   12   21   35    4]
 [   5    4   19  901    1   32    6   17   17    8]
 [   2    8    4    0  916    2   12    2    3   33]
 [  15    5    4   41   17  743   20   14   25    8]
 [  16    6    7    0   12   19  894    1    3    0]
 [   1   25   18    5   12    0    1  928    1   37]
 [  11   23   10   26   19   33   11   17  810   14]
 [  16    8    3   13   48    7    0   28    4  882]]
```

## Bonus 2

```python
# Bonus 2: Visualization of the weights
# This is only possible for simple classifiers

coef = clf.coef_.copy()
plt.figure(figsize=(10, 5))
scale = np.abs(coef).max()
for i in range(10):
    l1_plot = plt.subplot(2, 5, i + 1)
    l1_plot.imshow(coef[i].reshape(28, 28), interpolation='nearest',
                   cmap=plt.cm.RdBu, vmin=-scale, vmax=scale)
    l1_plot.set_xticks(())
    l1_plot.set_yticks(())
    l1_plot.set_xlabel('Class %i' % i)
plt.suptitle('Classification vector for...')
plt.show()
```



## Discussion:

The weight of visualization is quite straight forward, the orange color shows the contour of every forground, look at the orange colorin every class, class 0 looks like the number 0, class 1 looks like 1…. But for class 5, the orange color is more obscure, it is hard for us to imagine the contour of "5" from the orange color. Maybe "5" is more complicated in shape compared with other numbers.

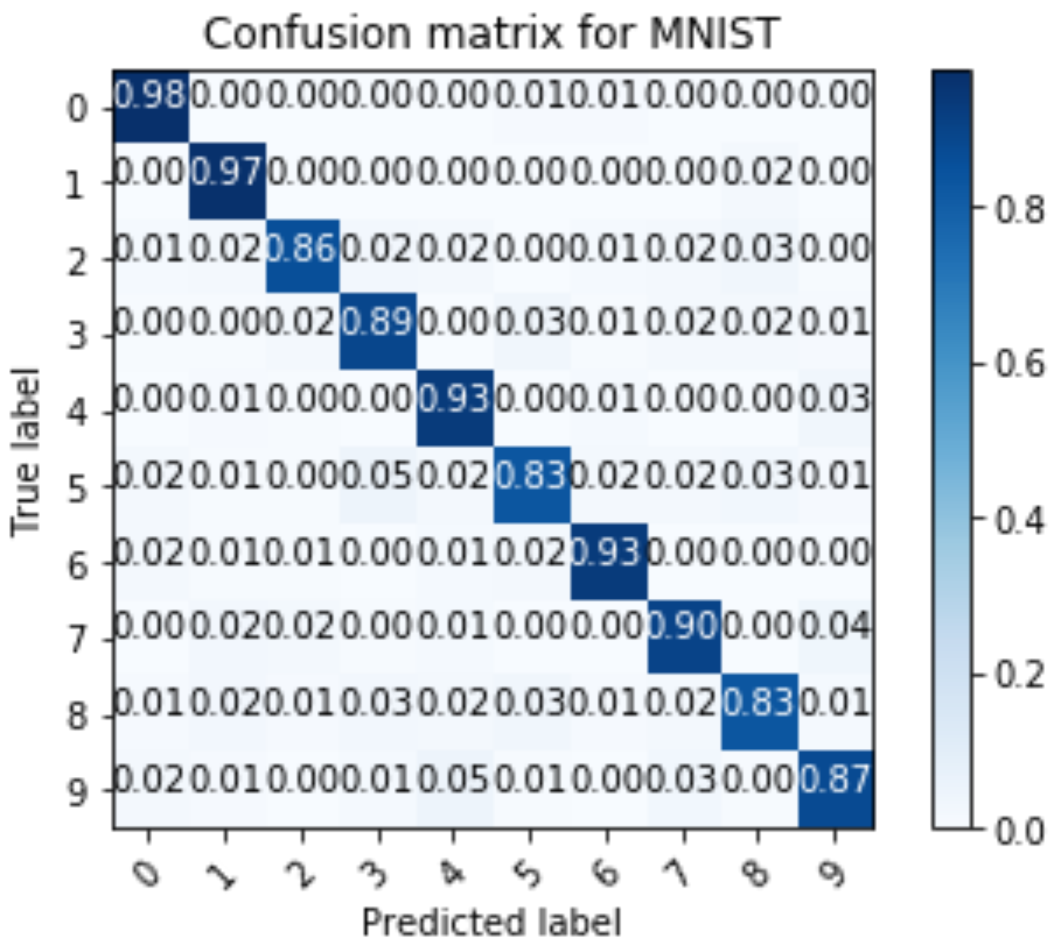
## Bouns 3

```python
### Bonus 3: comfusion matrix visualization tool
## A more elegant presentation for a confusion matrix

```

```
import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    #print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    #plt.tight_layout()
    plt.show()
plot_confusion_matrix(cfm, classes=range(10), normalize=True,
                      title='Confusion matrix for MNIST')
```

Finally, we plot the confusion matrix for MNIST as below:

Normalized confusion matrix



## Discussion:

The sample code in tutorial is not absolutely correct, I use the code from tutorial cannot draw the confusion matrix for MNIST, I make small change on the code, as below:
I add

```
normalize=True,
```

I comment this line and add another plot line.

```
#plt.tight_layout()
    plt.show()
```