Importing the MNIST digits dataset from keras

In [1]:
```
import tensorflow as tf
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load
_data()
```

Flattening the 28*28 pixel image to 784 vector

In [2]:
```
x_train=x_train.reshape(60000,784)
y_train=y_train.reshape(60000,1)
print(x_train.shape,y_train.shape)
```

```
(60000, 784) (60000, 1)
```

In [3]:
```
print(x_test.shape,y_test.shape)
```

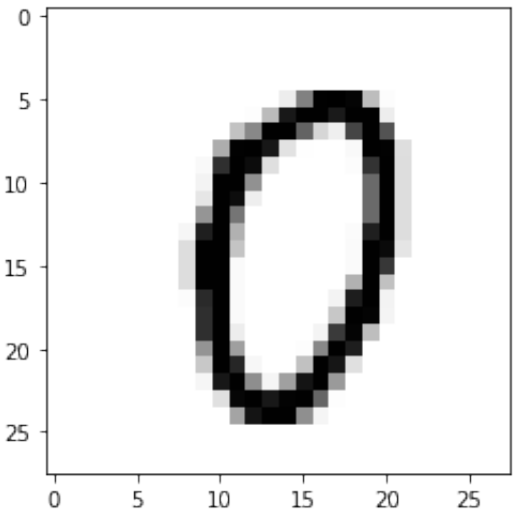
```
(10000, 28, 28) (10000,)
```

Displaying a sample training data

In [5]:
```
import matplotlib.pyplot as plt

image_index = 1000 # We may select anything up to 60,000
print(y_train[image_index]) # The label is 0
plt.imshow(x_train[image_index].reshape(28,28), cmap='Greys')
```

```
[0]
```

Out[5]:
```
<matplotlib.image.AxesImage at 0x638b66898>
```



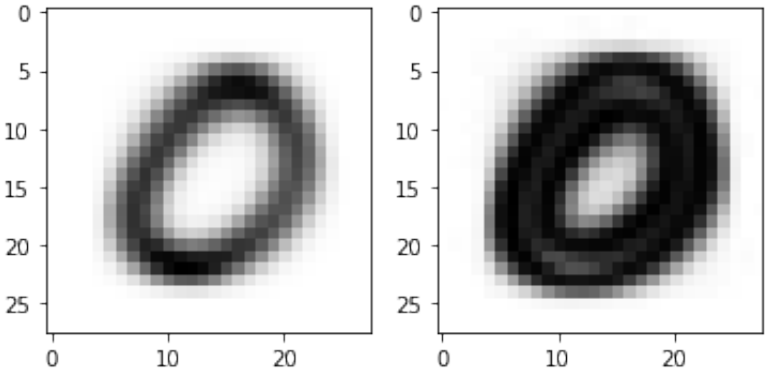
In [6]:
```
import numpy as np

dataset=np.concatenate((x_train,y_train),axis=1)
print(dataset.shape)
```

```
(60000, 785)
```

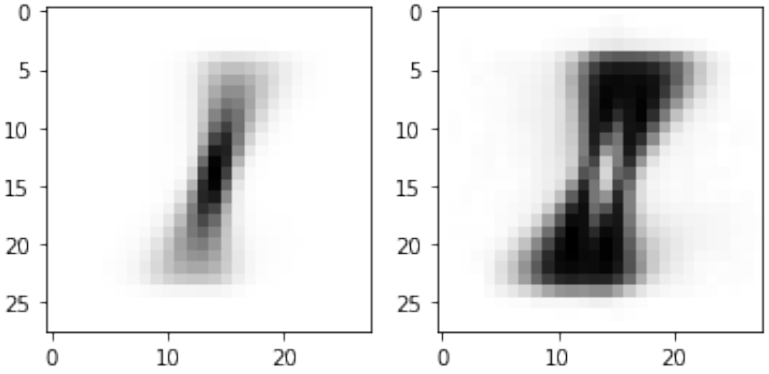
For each class (0 to 9) the training examples of that particular class is extracted.
Then the prior probability, mean, standard-deviation and covariance matrix is calculated for each class and displayed.

In [7]:
```
mean_records=[]
sd_records=[]
prior_prob=[]

cov_records=[]

for i in range(10):
    temp=[]
    for j in range(60000):
        if dataset[j,-1]==i:
            temp.append(list(dataset[j,:-1]))

    temp=np.array(temp)
    row,col=temp.shape
    prior_prob.append(row/60000)
    mean=np.mean(temp,axis=0)
    mean=mean.reshape(784,1)
    mean_records.append(mean)
    sd=np.std(temp,axis=0)
    sd_records.append(sd)

    forcov=temp.T
    cov=np.cov(forcov)
    np.fill_diagonal(cov, cov.diagonal() + 0.1)

    cov_records.append(cov)

    print('Mean and Standard Deviation for digit  ',i)

    f = plt.figure()
    f.add_subplot(1,2, 1)
    plt.imshow(mean.reshape(28,28), cmap='Greys')
    f.add_subplot(1,2, 2)
    plt.imshow(sd.reshape(28,28), cmap='Greys')
    plt.show(block=True)
```

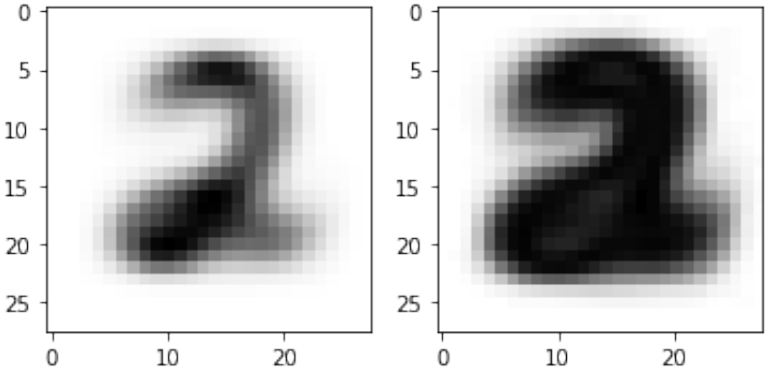
```
Mean and Standard Deviation for digit   0
```



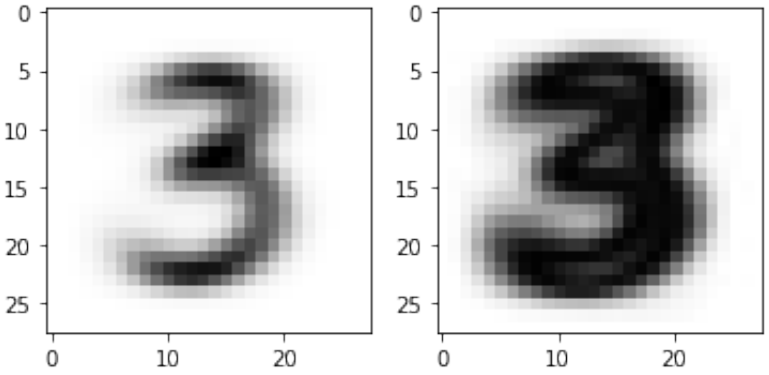
Mean and Standard Deviation for digit 1



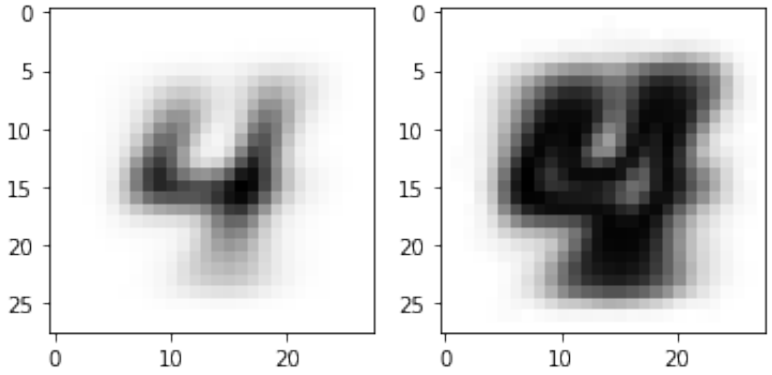
Mean and Standard Deviation for digit 2



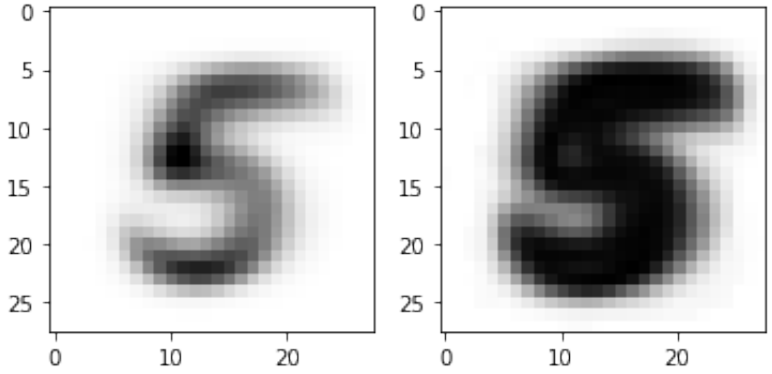
Mean and Standard Deviation for digit 3



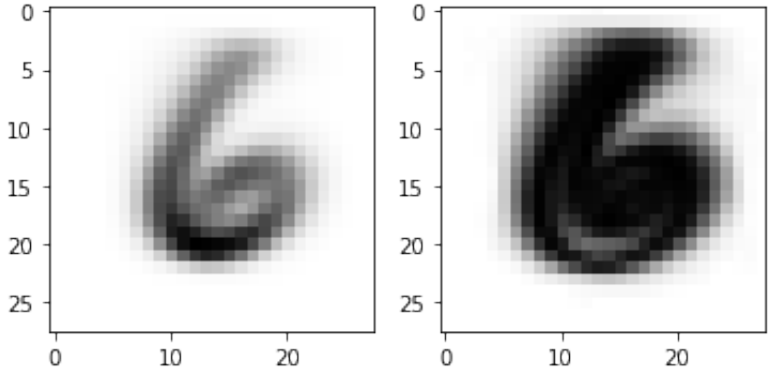
Mean and Standard Deviation for digit 4



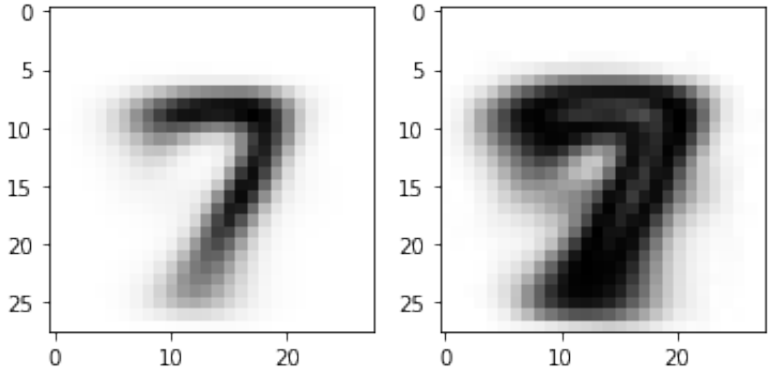
Mean and Standard Deviation for digit 5



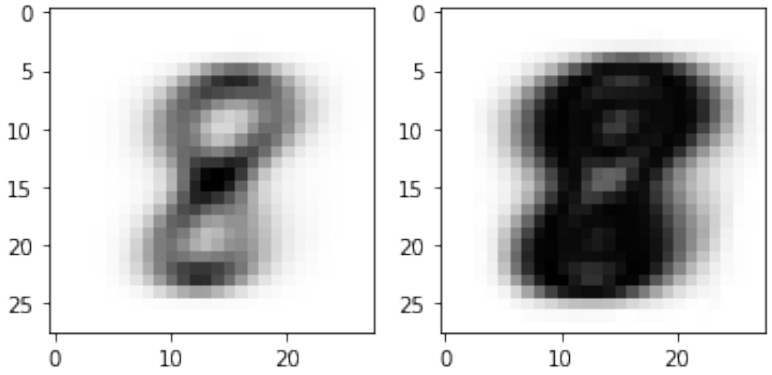
Mean and Standard Deviation for digit 6



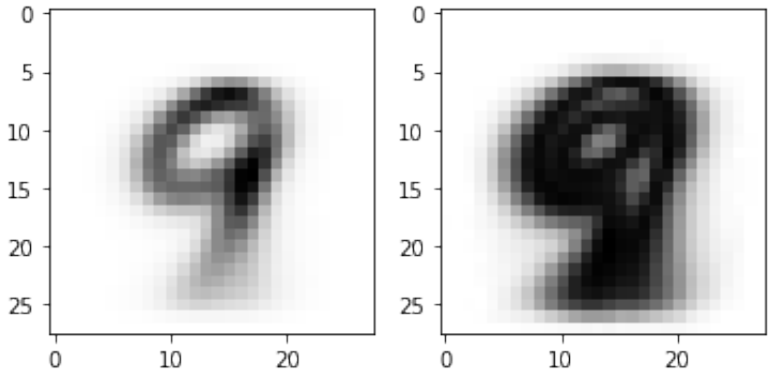
Mean and Standard Deviation for digit 7



Mean and Standard Deviation for digit 8



Mean and Standard Deviation for digit 9



In [8]:
```
for i in cov_records:
    print(np.linalg.slogdet(i))
```

```
(1.0, 2426.6108181541185)
(1.0, 1409.3621917329413)
(1.0, 2927.5039846998757)
(1.0, 2647.113928597218)
(1.0, 2652.07354138473)
(1.0, 2700.5552922981537)
(1.0, 2335.215422501885)
(1.0, 2444.8362584351344)
(1.0, 2520.4278616043)
(1.0, 2183.6917163313633)
```

In [9]:
```
(cov_records[0])
```

Out[9]:
```
array([[0.1, 0. , 0. , ..., 0. , 0. , 0. ],
       [0. , 0.1, 0. , ..., 0. , 0. , 0. ],
       [0. , 0. , 0.1, ..., 0. , 0. , 0. ],
       ...,
       [0. , 0. , 0. , ..., 0.1, 0. , 0. ],
       [0. , 0. , 0. , ..., 0. , 0.1, 0. ],
       [0. , 0. , 0. , ..., 0. , 0. , 0.1]])
```

In [10]:
```
cov_records[0].shape
```

Out[10]:
```
(784, 784)
```

In [11]:
```
mean_records[0].shape
```

Out[11]:
```
(784, 1)
```

In [12]:
```
for i in range(len(prior_prob)):
    print('Prior Probability of Class ',i,' = ',prior_prob[i])
```

```
Prior Probability of Class  0  =  0.09871666666666666
Prior Probability of Class  1  =  0.11236666666666667
Prior Probability of Class  2  =  0.0993
Prior Probability of Class  3  =  0.10218333333333333
Prior Probability of Class  4  =  0.09736666666666667
Prior Probability of Class  5  =  0.09035
Prior Probability of Class  6  =  0.09863333333333334
Prior Probability of Class  7  =  0.10441666666666667
Prior Probability of Class  8  =  0.09751666666666667
Prior Probability of Class  9  =  0.09915
```

We assume that the distribution of each class is Gaussian

$$\frac{1}{(2\pi)^{(d/2)}\det \sum^{1/2}}\exp(-1/2(x-\mu)^T \sum^{-1}(x-\mu))$$

where $\sum$ is the covariance matrix, μ is the mean of the respective class.

Now to calculate the discriminant function we use :

$$g_i(x) = \log(p|w_i) + \log(P(w_i))$$

$$g_i(x) = -\frac{1}{2}(x-\mu_i)^T \sum_i^{-1}(x-\mu_i) - \frac{d}{2}\log(2\pi) - \frac{1}{2}\log\det\sum_i + \log P(w_i)$$

$P(w_i)$ is the prior probability for each class.

Here we use the generalized quadratic discriminant function which is for different covariance matrices for each category or class.

$$g_i(x) = x^t W_i x + N_i^t x + B_{i0}$$

$$W_i = -1/2\sum_i^{-1}$$

$$N_i = \sum_i^{-1}\mu_i$$

$$B_{i0} = -1/2\mu_i^t\sum_i^{-1}\mu_i + \ln P(\omega_i) - 1/2\ln|\sum_i|$$

In [13]:
```
W_i=[]
N_i=[]
B_i0=[]

for i in range(10):
    sign,det=np.linalg.slogdet(cov_records[i])

    W_i.append(-1/2*np.linalg.inv(cov_records[i]))
    x=(np.matmul(np.linalg.inv(cov_records[i]),mean_records[i]))
    N_i.append(x)
    B_i0.append(-1/2*np.matmul(mean_records[i].T,x)-1/2*det+np.log(
prior_prob[i]))
```

In [14]:
```
def test(x):

    scores=[]

    for i in range(10):

        g_score=np.matmul(np.matmul(x.T,W_i[i]),x)+np.matmul(N_i[i]
.T,x)+B_i0[i]

        scores.append(g_score)

    return np.argmax(np.array(scores))
```

In [15]:
```
x_test=x_test.reshape(10000,784)
y_test=y_test.reshape(10000,)
print(x_test.shape,y_test.shape)
```

```
(10000, 784) (10000,)
```

Here accuracy is calculated using 0/1 Loss.
For any misclassification we assume the loss to be 1 and for every correctly classified data point the accuracy is increased by 1 and loss is 0.

In [16]:
```
hit=0
loss=0
for i in range(len(x_test)):
    pred=test(x_test[i].reshape(784,1))
    if pred==y_test[i]:
        hit+=1
    else:
        loss+=1

print("Accuracy % = ",hit/10000*100)
print("Loss % = ",loss/10000*100)
```

```
Accuracy % =  81.08999999999999
Loss % =  18.91
```

The loss is 18.91% which is much worse than all the other classification methods proposed in [http://yann.lecun.com/exdb/mnist/index.html (http://yann.lecun.com/exdb/mnist/index.html)](http://yann.lecun.com/exdb/mnist/index.html)

Quadratic discriminant analysis (QDA) is a variant of Linear Discriminant Analysis(LDA) that allows for non-linear separation of data.
This discriminant function analysis like LDA and QDA have a few drawbacks:

a) LDA and QDA assumes that the data are Gaussian. They are expected to work well if the class conditional densities of clusters are approximately normal. LDA on the other hand assumes that all classes share same covariance matrix.

b) LDA finds linear decision boundaries in a K-1 dimensional subspace. As such, it is not suited if there are higher order interactions between the independent variables.

c) Both LDA and QDA are well-suited for multi-class problems but should be used with care when the class distribution is imbalanced because the priors are estimated from the observed counts. Thus, observations will rarely be classified to infrequent classes.

In [ ]: