In [1]:

```
#importing functions
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf #using it only for loading the MNIST dataset
from PIL import Image
import math
```

In [2]:

```
#loading the MNIST dataset from the tensorflow library
from tensorflow.examples.tutorials.mnist import input_data
data=input_data.read_data_sets("data/MNIST/",one_hot=True)
```

```
Extracting data/MNIST/train-images-idx3-ubyte.gz
Extracting data/MNIST/train-labels-idx1-ubyte.gz
Extracting data/MNIST/t10k-images-idx3-ubyte.gz
Extracting data/MNIST/t10k-labels-idx1-ubyte.gz
```

In [3]:

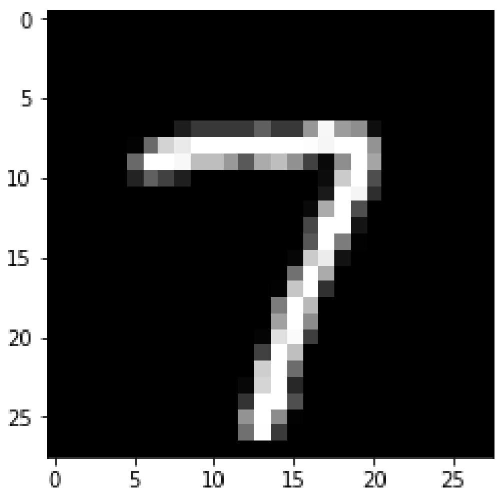
```
#assigning variables to the different dataset
train_data=data.train.images
train_labels = np.asarray(data.train.labels, dtype=np.int32)
eval_data = data.test.images
eval_labels = np.asarray(data.test.labels, dtype=np.int32)
```

In [4]:

```
#Reshape MNIST
X_train = train_data.reshape(train_data.shape[0], 28, 28, 1)
X_test = eval_data.reshape(eval_data.shape[0], 28, 28, 1)
```

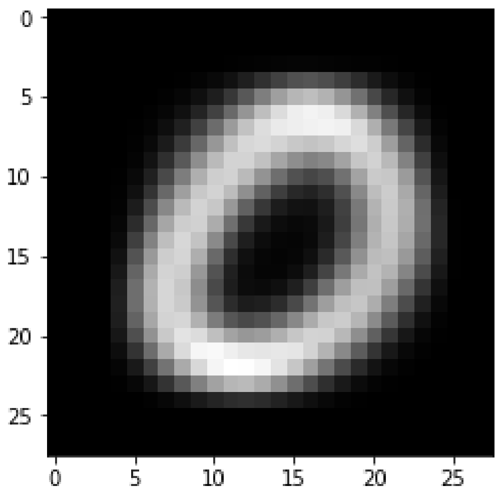
In [41]:

```
#Plotting a sample image
plt.imshow(X_train[100].reshape(28,28),cmap='gray')
plt.show()
```



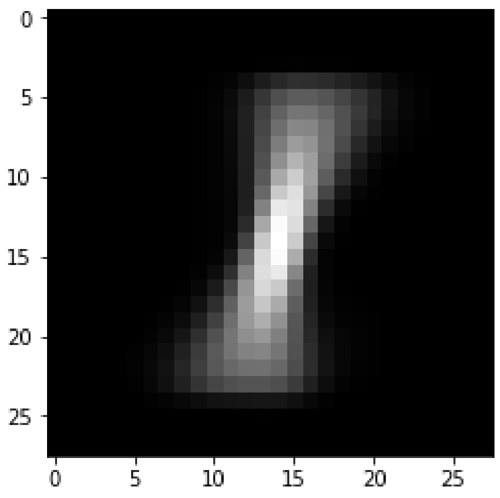
In [42]:

```
#Mean for Zero
X0=train_data[train_labels[:,0]==1]
m0 = np.mean(X0,axis=0)
plt.imshow(m0.reshape(28,28),cmap='gray')
plt.show()
```



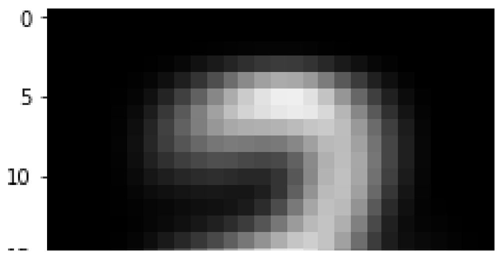
In [43]:

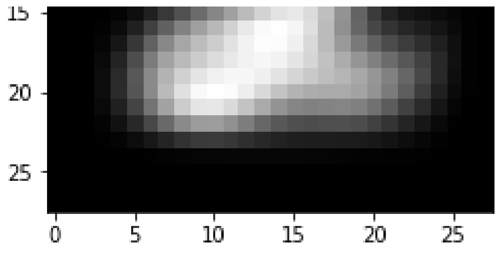
```
#Mean for One
X1=train_data[train_labels[:,1]==1]
m1 = np.mean(X1,axis=0)
plt.imshow(m1.reshape(28,28),cmap='gray')
plt.show()
```



In [44]:

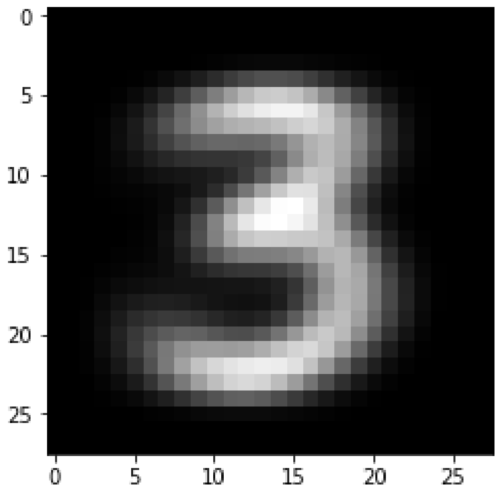
```
#Mean for Two
X2=train_data[train_labels[:,2]==1]
m2 = np.mean(X2,axis=0)
plt.imshow(m2.reshape(28,28),cmap='gray')
plt.show()
```





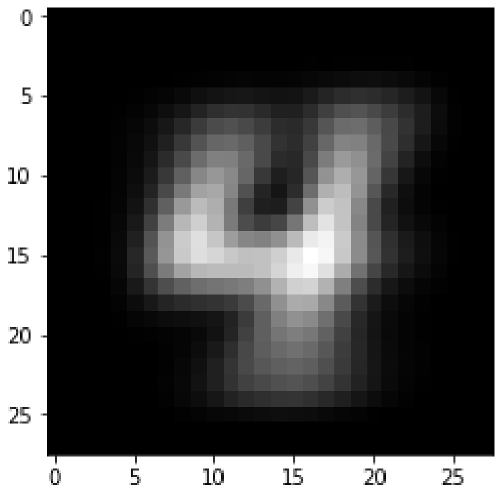
In [45]:

```
#Mean for Three
X3=train_data[train_labels[:,3]==1]
m3 = np.mean(X3,axis=0)
plt.imshow(m3.reshape(28,28),cmap='gray')
plt.show()
```



In [46]:

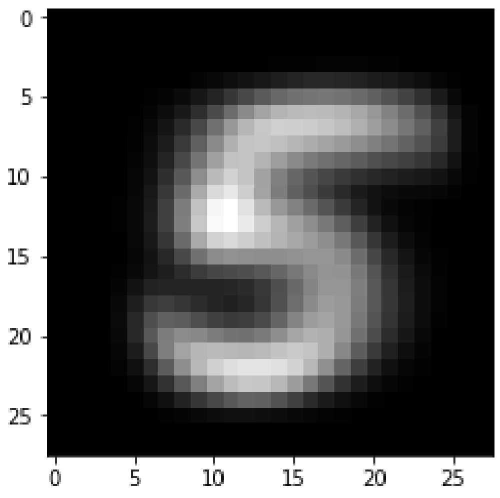
```
#Mean for Four
X4=train_data[train_labels[:,4]==1]
m4 = np.mean(X4,axis=0)
plt.imshow(m4.reshape(28,28),cmap='gray')
plt.show()
```



In [47]:

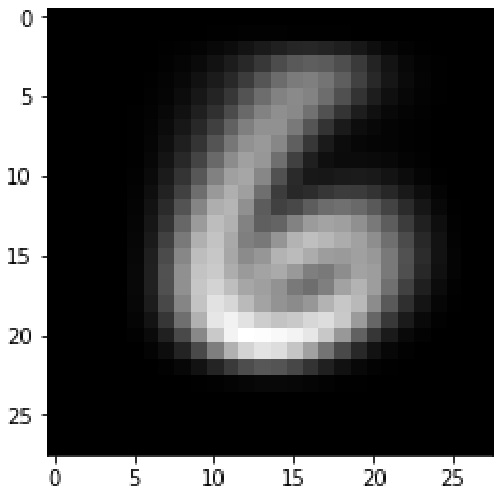
```
#Mean for Five
X5=train_data[train_labels[:,5]==1]
m5 = np.mean(X5,axis=0)
```

```
plt.imshow(m5.reshape(28,28),cmap='gray')
plt.show()
```



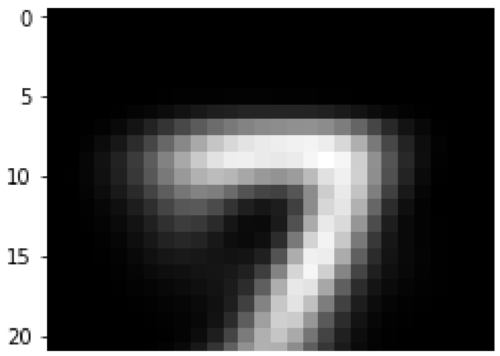
In [48]:

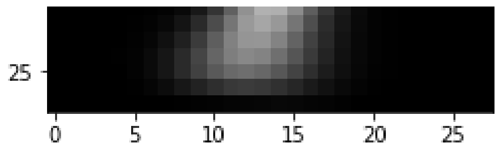
```
#Mean for Six
X6=train_data[train_labels[:,6]==1]
m6 = np.mean(X6,axis=0)
plt.imshow(m6.reshape(28,28),cmap='gray')
plt.show()
```



In [49]:

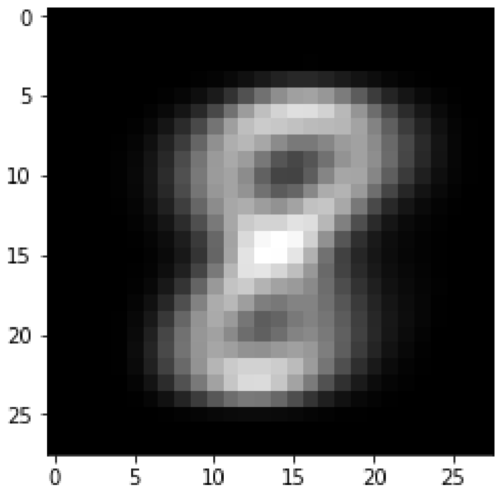
```
#Mean for Seven
X7=train_data[train_labels[:,7]==1]
m7 = np.mean(X7,axis=0)
plt.imshow(m7.reshape(28,28),cmap='gray')
plt.show()
```





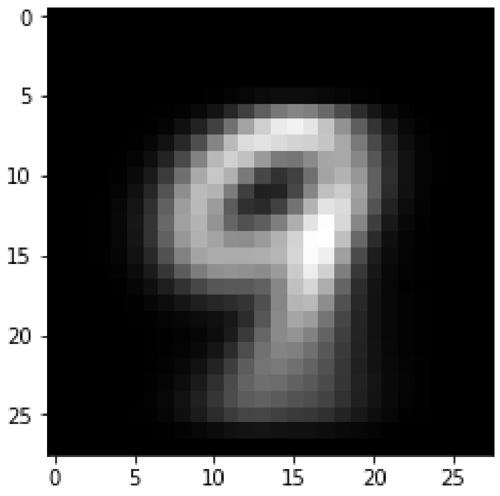
In [50]:

```
#Mean for Eight
X8=train_data[train_labels[:,8]==1]
m8 = np.mean(X8,axis=0)
plt.imshow(m8.reshape(28,28),cmap='gray')
plt.show()
```



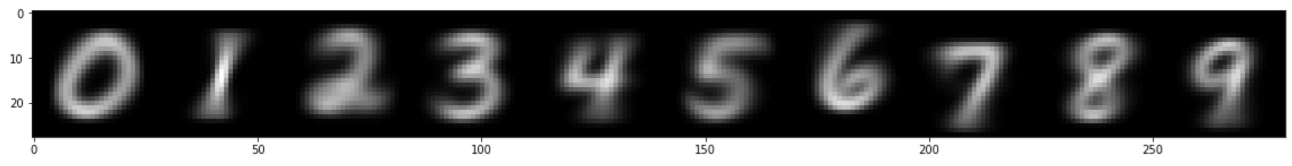
In [51]:

```
#Mean for Nine
X9=train_data[train_labels[:,9]==1]
m9 = np.mean(X9,axis=0)
plt.imshow(m9.reshape(28,28),cmap='gray')
plt.show()
```



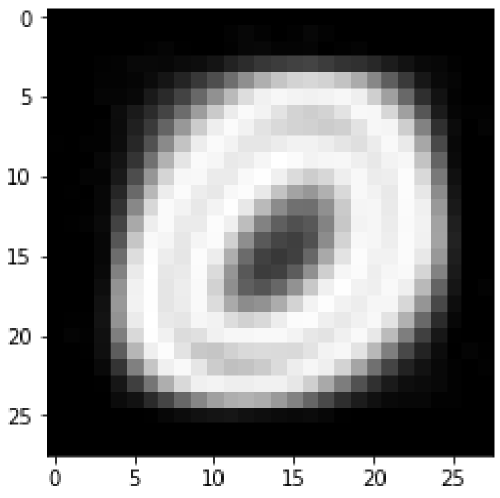
In [53]:

```
#Drawing a montage of all the mean digit images
plot_image = np.concatenate((m0.reshape(28,28),m1.reshape(28,28),m2.reshape
(28,28),
m3.reshape(28,28),m4.reshape(28,28),m5.reshape(28,28),m6.reshape(28,28),
m7.reshape(28,28),m8.reshape(28,28),m9.reshape(28,28)),axis=1)
plt.figure(figsize = (20,20))
plt.imshow(plot_image,cmap='gray');
plt.show()
```



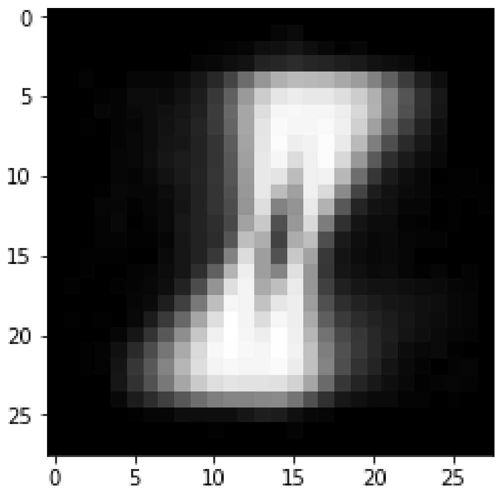
In [55]:

```
#Standard Deviation for Zero
X00=train_data[train_labels[:,0]==1]
m00 = np.std(X00,axis=0)
plt.imshow(m00.reshape(28,28),cmap='gray')
plt.show()
```



In [56]:

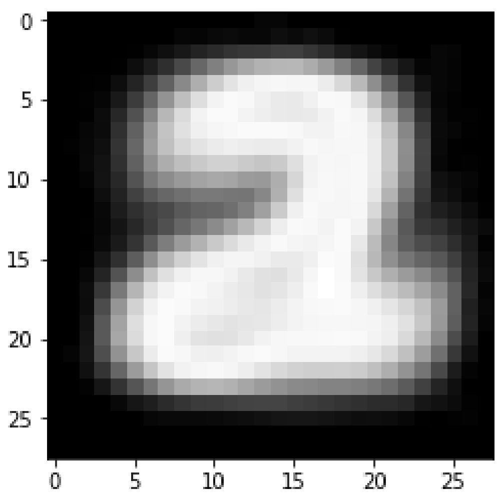
```
#Standard deviation for One
X11=train_data[train_labels[:,1]==1]
m11 = np.std(X11,axis=0)
plt.imshow(m11.reshape(28,28),cmap='gray')
plt.show()
```



In [57]:

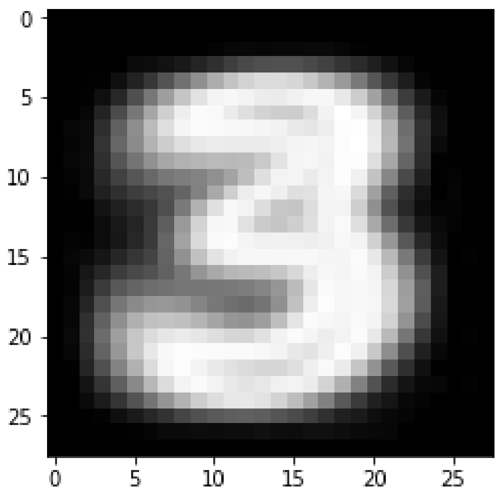
```
#Standard deviation for Two
X22=train_data[train_labels[:,2]==1]
```

```
m22 = np.std(X22,axis=0)
plt.imshow(m22.reshape(28,28),cmap='gray')
plt.show()
```



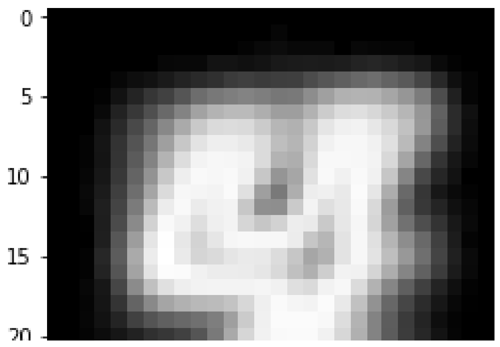
In [58]:

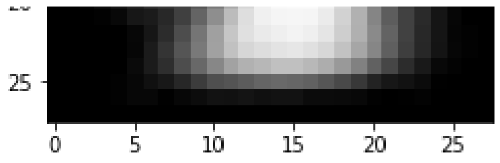
```
#Standard deviation for Three
X33=train_data[train_labels[:,3]==1]
m33 = np.std(X33,axis=0)
plt.imshow(m33.reshape(28,28),cmap='gray')
plt.show()
```



In [59]:

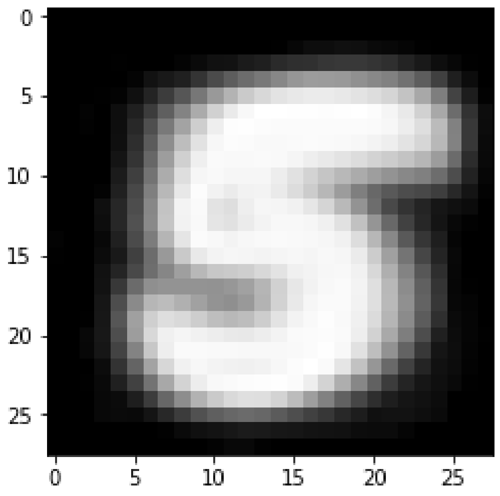
```
#Standard deviation for Four
X44=train_data[train_labels[:,4]==1]
m44 = np.std(X44,axis=0)
plt.imshow(m44.reshape(28,28),cmap='gray')
plt.show()
```





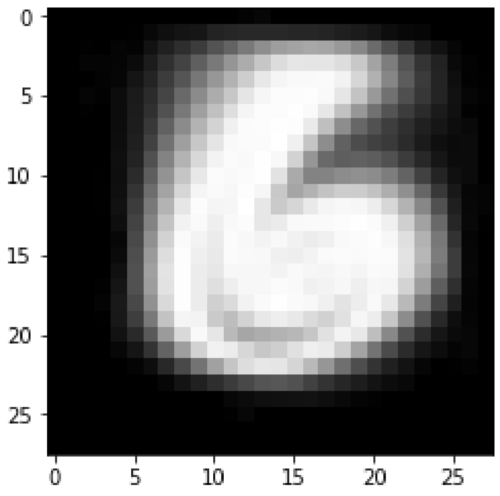
In [60]:

```
#Standard deviation for Five
X55=train_data[train_labels[:,5]==1]
m55 = np.std(X55,axis=0)
plt.imshow(m55.reshape(28,28),cmap='gray')
plt.show()
```



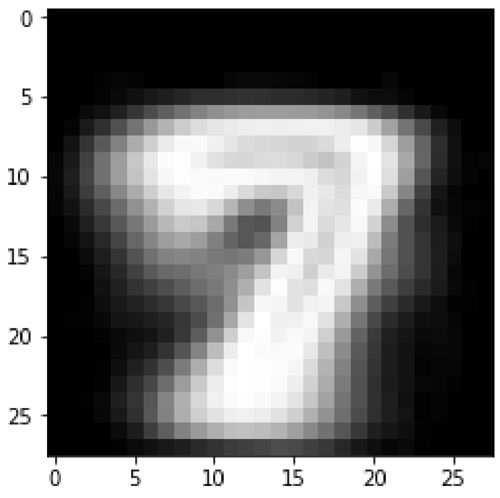
In [61]:

```
#Standard deviation for Six
X66=train_data[train_labels[:,6]==1]
m66 = np.std(X66,axis=0)
plt.imshow(m66.reshape(28,28),cmap='gray')
plt.show()
```



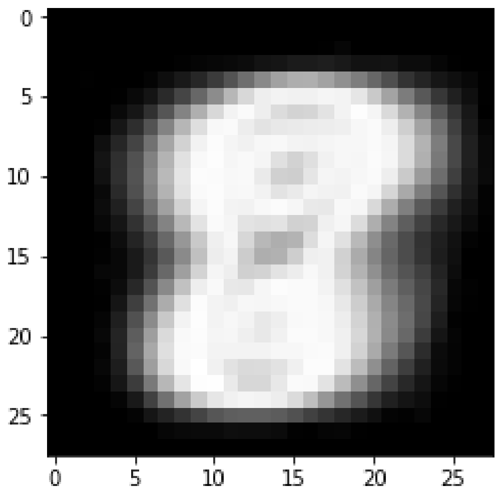
In [62]:

```
#Standard deviation for Seven
X77=train_data[train_labels[:,7]==1]
m77 = np.std(X77,axis=0)
plt.imshow(m77.reshape(28,28),cmap='gray')
plt.show()
```



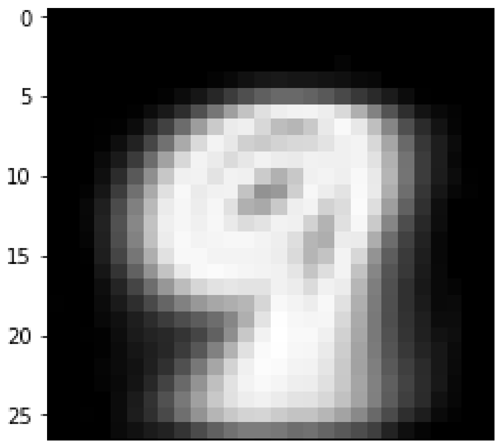
In [63]:

```
#Standard deviation for Eight
X88=train_data[train_labels[:,8]==1]
m88 = np.std(X88,axis=0)
plt.imshow(m88.reshape(28,28),cmap='gray')
plt.show()
```



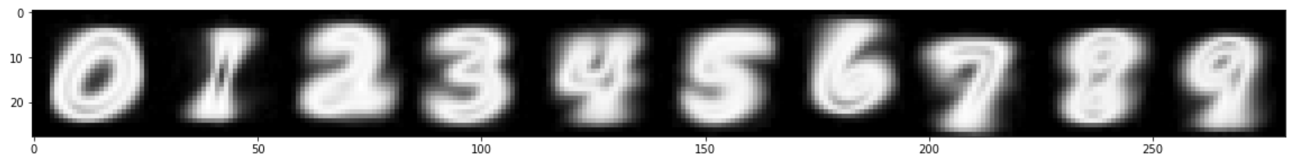
In [64]:

```
#Standard deviation for Nine
X99=train_data[train_labels[:,9]==1]
m99 = np.std(X99,axis=0)
plt.imshow(m99.reshape(28,28),cmap='gray')
plt.show()
```



In [65]:

```
#Drawing a montage of all the Standard deviation digit images
plot_image = np.concatenate((m00.reshape(28,28),m11.reshape(28,28),m22.resh
ape(28,28),
m33.reshape(28,28),m44.reshape(28,28),m55.reshape(28,28),m66.reshape(28,28)
,
m77.reshape(28,28),m88.reshape(28,28),m99.reshape(28,28)),axis=1)
plt.figure(figsize = (20,20))
plt.imshow(plot_image,cmap='gray');
plt.show()
```



In [30]:

```
#Calculating priors
prior0=X0.size/data.train.images.size
prior1=X1.size/data.train.images.size
prior2=X2.size/data.train.images.size
prior3=X3.size/data.train.images.size
prior4=X4.size/data.train.images.size
prior5=X5.size/data.train.images.size
prior6=X6.size/data.train.images.size
prior7=X7.size/data.train.images.size
prior8=X8.size/data.train.images.size
prior9=X9.size/data.train.images.size
```

In [31]:

```
#Calculating Covariances
C0=np.power(m00, 2)
C1=np.power(m11, 2)
C2=np.power(m22, 2)
C3=np.power(m33, 2)
C4=np.power(m44, 2)
C5=np.power(m55, 2)
C6=np.power(m66, 2)
C7=np.power(m77, 2)
C8=np.power(m88, 2)
C9=np.power(m99, 2)
```

In [32]:

```
#Creating an array with all the prior values
prior=np.array([prior0,prior1,prior2,prior3,prior4,prior5,prior6,prior7,prio
r8,prior9])
priors=[]
for i in range(len(prior)):
    priors.append(prior[i])

#Creating an array with all the mean values
mean=np.array([m0,m1,m2,m3,m4,m5,m6,m7,m8,m9])
means=[]
```

```
for i in range(len(mean)):
    means.append(mean[i])

#Creating an array with all the covariance values values
covMat= np.array([C0,C1,C2,C3,C4,C5,C6,C7,C8,C9])
covMatList=[]
for i in range(len(covMat)):
    covMatList.append(covMat[i])
```

In [33]:

```
#Creating a vector with label digits from the training data
train_mod=np.zeros(shape=(55000,1))
for i in range(len(train_labels)):
    for j in range(len(train_labels[i])):
        if train_labels[i][j]== 1:
            train_mod[i][0]=j


#Creating a vector with label digits from the testing data
eval_mod=np.zeros(shape=(10000,1))
for i in range(len(eval_labels)):
    for j in range(len(eval_labels[i])):
        if eval_labels[i][j]== 1:
            eval_mod[i][0]=j
```

In [34]:

```
#Manually defining the gaussian Function
from numpy.linalg import inv
from numpy.linalg import det

pi=3.14
k = 10
d = eval_data.shape[1]
count = 0
ypred=np.empty([eval_labels.shape[0],1])
for j in range(0,10000):
    x = np.matrix(eval_data[j]).T
    pmle = list()
    pri = list()
    # Xtest - a N x d matrix with each row correspond
    for i in range(0,k):

        mean = np.matrix(means[i]).T
        temp=1
        temp = np.exp(-0.5*temp)
        pmle.append(temp)




classes=[0,1,2,3,4,5,6,7,8,9]

#Function to get the data for individual digits
#Creating a matrix of digits
def get_examples_for_class(class_id):
    digits = []
    for i, digit in enumerate(train_data):
        if train_mod[i]==class_id:
            digits.append(digit)

    digits = np.matrix(digits)
    return digits
```

In [35]:

```
#fitting a multivariate_normal
post=[]

from scipy.stats import multivariate_normal
for class_id_all in classes:
    examples = get_examples_for_class(class_id_all)
    mean = np.array(examples.mean(0))[0]
    cov = np.cov(examples.T)
    p_x = multivariate_normal(mean=mean, cov=cov,allow_singular=True)
    post.append(p_x)
```

In [36]:

```
#Calculating to which class the data is nearest to
Y = []
for x in eval_data:
    bayes_probs = []
    for klass in classes:
        prob = [int(klass), (priors[int(klass)]) * post[int(klass)].pdf(x)]
        bayes_probs.append(prob)
    prediction = max(bayes_probs, key= lambda a: a[1])
    Y.append(prediction[0])
```

```
/Users/rajivranjan/anaconda3/lib/python3.6/site-
packages/scipy/stats/_multivariate.py:610: RuntimeWarning: overflow encount
ered in exp
  return np.exp(self.logpdf(x))
```

In [39]:

```
#Calculating the errors
Y_array=np.asarray(Y)
count=0
for i in range(len(eval_data)):
    if(eval_mod[i][0]!=Y_array[i]):
        count=count+1
#errors = (eval_mod != Y_array).sum()
total = eval_data.shape[0]
print("Error rate: %d/%d = %f" % ((count,total,(count/float(total)))))
```

```
Error rate: 2797/10000 = 0.279700
```

In [40]:

```
#accuracy percent
print("The accuracy rate: %f percent " % (((total-count)/float(total))*100)
)
```

```
The accuracy rate: 72.030000 percent
```