SPR Assignment 2

Jayanth S (201081003), Praveen Kumar N (201082001), Rishabh Roy (201082002)

March 17, 2021

1 Import the required libraries

[897]:

```
from mnist import MNIST
import os
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stat
from sklearn.metrics import confusion_matrix
import seaborn as sn
import pandas as pd
```

2 Problem 1: a) Gaussian Distribution

Estimated mean for i^{th} class is a vector with 784 mean feature values(pixel values) with j^{th} pixel value equal to:

$$\mu_{ij} = \sum_{l=1}^{m} \frac{x_{jl}}{m}, \tag{1}$$

where, m is the number of images of i^{th} particular class and
x_{jl} is the j^{th} pixel value of l^{th} image. $
$ The covaiance matrix for i^{th} class it is ,

$$\Sigma_i = \sum_{l=1}^{m} \frac{z_l}{m}, \tag{2}$$

where, $z_l = \frac{(x_l-\mu_i)*(x_l-\mu_i)^T}{n}$
Here x_l represents l^{th} image of i^{th} class and μ_i represents the mean of i^{th} class images.

2.1 MLE Estimation for Training Data with Class Conditional densities modeled as Multivariate Gaussian Distribution

[860]:

```
def mle_estimation_gaussian(train_images,train_labels):

    classes=[[] for i in range(np.max(train_labels)+1)]  #creates a list of 10␣
 ↪empty lists with each list referring to a class in our case (10 classes)
    for i in range(len(train_labels)):
          classes[train_labels[i]].append(train_images[i]) # the images with label k is added to the class k.


    mu= np.array([np.mean(classes[i],axis=0) for i in range(np.max(train_labels)+1)])  #generate the MLE estimated mean vectors for each class

    temp_var=[np.array(classes[i][:][:]) for i in range(np.max(train_labels)+1)]   #each item in the list contains all the  feature vectors of a class
    temp_var=[temp_var[i].T for i in range(np.max(train_labels)+1)]                    #take the transpose of each matrix present in the list
    #print(np.shape(mu))          #to know the shape of mean


    covariance=[((np.cov(temp_var[i]))/(np.shape(temp_var[i])[1]))+ 0.35*np.identity(len(temp_var[i])) for i in range(np.max(train_labels)+1)]     #find the MLE estimated covarinace matrix for each class
#     for i in range(10):
#         print(np.linalg.det(covariance[i]))
    #print(np.shape(covariance[1]))         #to know the shape of varainace


    #Below 3 lines is to calculate the prior probabilites of each class.
    prior_prob=np.zeros((np.max(train_labels)+1,1))
    for i in range(np.max(train_labels)+1):
        prior_prob[i]=len(classes[i])/(np.shape(train_images)[0])


# print("prior probabilities")
# print(prior_prob)            #to print the prior probabilities

#to plot the images with pixel values of a class (0-9) equal to the mean values of the corresponding classes
    plt.suptitle("Images constructed using mean values of pixels for each class")
    for i in range(np.max(train_labels)+1):
        plt.subplot(2,5,i+1)
        train_imag1=mu[i][:]
        train_imag1=np.array(train_imag1)
        train_imag1=train_imag1.reshape(int(np.sqrt(len(train_imag1))),int(np.sqrt(len(train_imag1))))
        plt.imshow(train_imag1)
        plt.axis('off')
        plt.title("Class "+str(i))
```

```
#      plt.show()
    return mu,covariance,prior_prob
```

2.2 Bayes Classfier for Test Data with Class Conditional densities modeled as Multivariate Gaussian Distribution

[861]:
```
def bayes_clasfr_gaussian(test_images,test_labels,mu,covariance,prior_prob):

  mis_clf=0

  predicted_label=np.zeros((len(test_images)))    #to store the predicted label␣
 ↪for each image
  #print(np.shape(predicted_label))

  for i in range(len(test_labels)):
    temp_image=np.array(test_images[i][:])   #load the feature vector of␣
 ↪particular image that needs to be classified
    pst_prob=np.zeros((np.max(train_labels)+1))           #to store posterior␣
 ↪probabilites given the feature vector

    for j in range(np.max(test_labels)+1):
        covariance_temp=covariance[j][:][:]   # load the co-variance matrix of␣
 ↪jth class
        mu_temp=np.array(mu[j,:])             #load the mean vector of jth class
        x=(temp_image-mu_temp).T@np.linalg.
 ↪inv(covariance_temp)@(temp_image-mu_temp)
        #print(np.log(np.linalg.det(covariance_temp)))
        pst_prob[j]=-0.5*len(covariance_temp)*np.log(2*np.pi)-0.5*np.log(np.
 ↪linalg.det(covariance_temp))-0.5*x + np.log(prior_prob[j])    #find the PDF of␣
 ↪the feature vector that follows the given distribution

    predicted_label[i]=np.argmax(pst_prob)                                #find␣
 ↪the index at which the posterior probability is maximum

    if predicted_label[i]!=test_labels[i]:
        mis_clf+=1
        #print(i,mis_clf,predicted_label[i],test_labels[i])                 #to␣
 ↪print the predicted class and ture class of a mis-classified image

  #print(np.shape(predicted_label))
  test_labels=np.array(test_labels)
  #to create the confusion matrix
  conf_matrix=confusion_matrix(test_labels,predicted_label)       #to create the␣
 ↪confusion matrix

  return conf_matrix,mis_clf
```

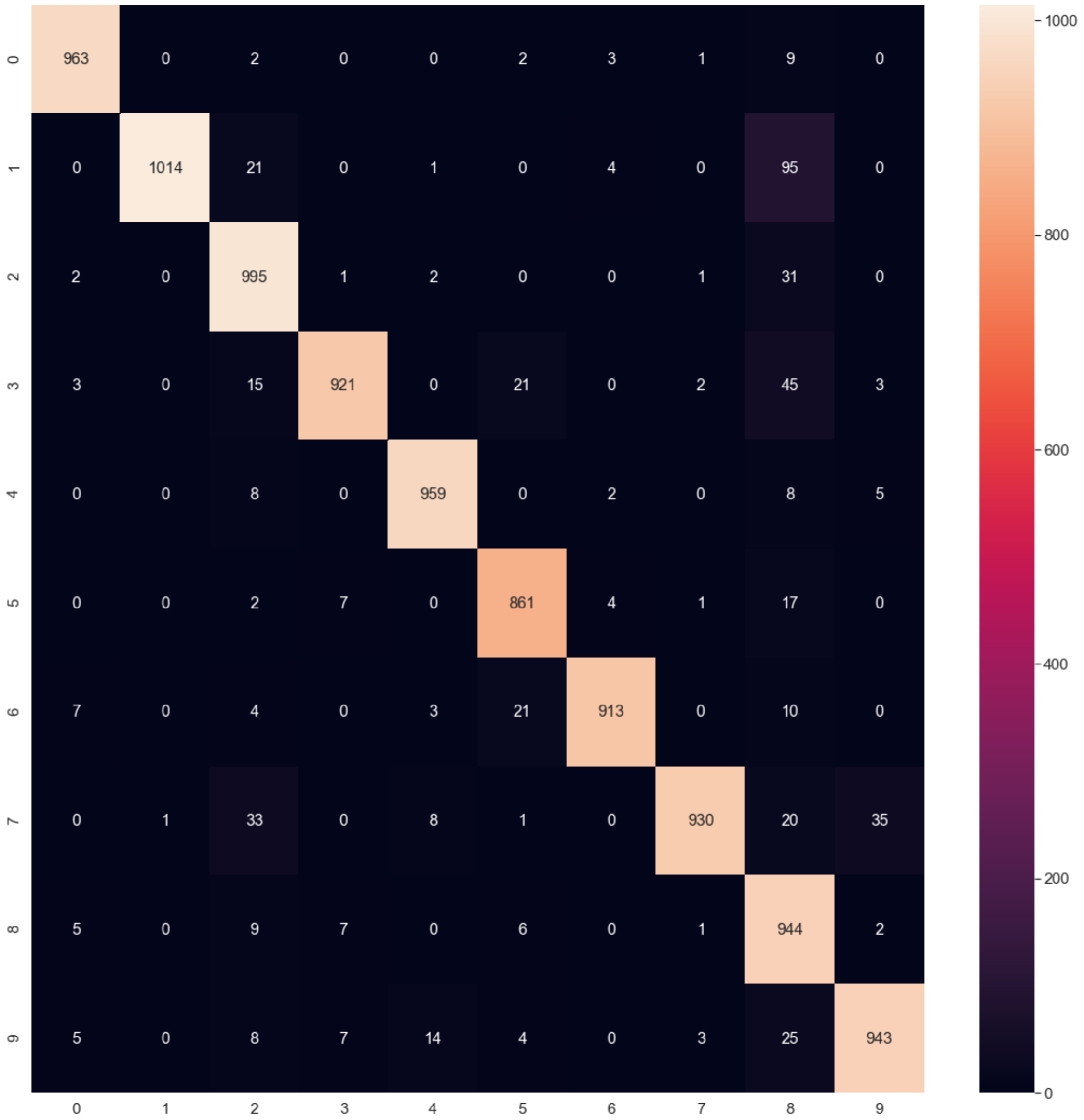
3 The below cell is to plot Confusion Matrix

[862]:
```
def plot_cnf_matrix(conf_matrix):
    df_cm = pd.DataFrame(conf_matrix, range(np.shape(conf_matrix)[0]), range(np.
 ↪shape(conf_matrix)[1]))
    plt.figure(figsize=(20,20))
    sn.set(font_scale=1.4) # for label size
    sn.heatmap(df_cm, annot=True, annot_kws={"size":16},fmt='d') # font size
    plt.show()
```

4 Training the Model and Testing the Model: :

[869]:
```
mndata=MNIST(r"Dataset_spr_2")  #Extract MNIST data from the specified folder.␣
 ↪mndata has images of numbers 0-9 with their respective labels

train_images, train_labels=mndata.load_training()      #loads training images␣
 ↪and their labels
#print(np.shape(train_images[1]))          #(60,0000,784) there are 60,000 images␣
 ↪with each of size 28x28
#print(np.shape(train_labels))          #(60,000,) labels
test_images, test_labels=mndata.load_testing()     #loads testing images and␣
 ↪their labels
test_images=np.array(test_images)
#print(np.shape(test_images))

mu,covariance,prior_prob = mle_estimation_gaussian(train_images,train_labels)

print(f"Total no. of images uses for training : {str(len(train_images))}")
print(f"No. of pixel values for each image : {str(np.shape(train_images)[1])}")
print(f"No. of classes is : {str(np.max(train_labels)+1)}")
print(f"Classifier : Bayes Classifier and Class Conditional densities for␣
 ↪training Data : Mulitvariate Gaussian Distribution")
for i in range(np.max(train_labels)+1):
    print("Prior Probability of class " + str(i) + " is "  +␣
 ↪str(round(prior_prob[i,0],3)))

conf_matrix,mis_clf =␣
 ↪bayes_clasfr_gaussian(test_images,test_labels,mu,covariance,prior_prob)

plot_cnf_matrix(conf_matrix)

perc_misclf= mis_clf/len(test_labels)

print(f"The % of misclassification is {str(perc_misclf*100)}% for MLE estimation␣
 ↪assuming the data is sampled from Multivariate Gaussian Distribution")
```

```
Total no. of images uses for training : 60000
No. of pixel values for each image : 784
No. of classes is : 10
Classifier : Bayes Classifier and Class Conditional densities for training Data
: Mulitvariate Gaussian Distribution
Prior Probability of class 0 is 0.099
Prior Probability of class 1 is 0.112
Prior Probability of class 2 is 0.099
Prior Probability of class 3 is 0.102
Prior Probability of class 4 is 0.097
Prior Probability of class 5 is 0.09
Prior Probability of class 6 is 0.099
Prior Probability of class 7 is 0.104
Prior Probability of class 8 is 0.098
Prior Probability of class 9 is 0.099
```



	0	1	2	3	4	5	6	7	8	9
0	963	0	2	0	0	2	3	1	9	0
1	0	1014	21	0	1	0	4	0	95	0
2	2	0	995	1	2	0	0	1	31	0
3	3	0	15	921	0	21	0	2	45	3
4	0	0	8	0	959	0	2	0	8	5
5	0	0	2	7	0	861	4	1	17	0
6	7	0	4	0	3	21	913	0	10	0
7	0	1	33	0	8	1	0	930	20	35
8	5	0	9	7	0	6	0	1	944	2
9	5	0	8	7	14	4	0	3	25	943

```
The % of misclassification is 5.57% for MLE estimation assuming the data is
sampled from Multivariate Gaussian Distribution
```

[894]:

```
def bayes_clasfr_gaussian_combine(test_images,test_labels,mu,covariance,prior_prob,combined_class

    mis_clf=0

    predicted_label=np.zeros((len(test_images)))     #to store the predicted label for each image
    #print(np.shape(predicted_label)
    mod_test_label=range(np.max(test_labels)+1)  #hard_coded the labels as the generalised code didnot work
    cmb_class=''
    for i in range(len(combined_class)):
        cmb_class+=str(combined_class[i])

    cmb_class=int(cmb_class)

    mod_test_label=[i if i!=combined_class[i%2] else cmb_class for i in mod_test_label]

    #print(np.shape(mod_test_label))

#     mod_test_label=np.append(mod_test_label,cmb_class)
    #print(np.shape(mod_test_label))

    for i in range(len(test_labels)):
        if test_labels[i] in combined_class:
            test_labels[i]=cmb_class

    for i in range(len(test_labels)):
        temp_image=np.array(test_images[i][:])   #load the feature vector of particular image that needs to be classified
        pst_prob=np.zeros((np.max(train_labels)+1-len(combined_class)+1))         #to store posterior probabilites given the feature vector

        for j in range(len(mu)):
            covariance_temp=covariance[j][:][:]   # load the co-variance matrix of jth class
            #print(np.shape(covariance))
            mu_temp=np.array(mu[j][:])            #load the mean vector of jth class
            x=(temp_image-mu_temp).T@np.linalg.inv(covariance_temp)@(temp_image-mu_temp)
            #print(np.log(np.linalg.det(covariance_temp)))
            pst_prob[j]=-0.5*len(covariance_temp)*np.log(2*np.pi)-0.5*np.log(np.linalg.det(covariance_temp))-0.5*x + np.log(prior_prob[j])   #find the PDF of the feature vector that follows the given distribution

        predicted_label[i]=mod_test_label[np.argmax(pst_prob)]                   #find the index at which the posterior probability is maximum

        if predicted_label[i]!=test_labels[i]:
            mis_clf+=1
            #print(i,mis_clf,predicted_label[i],test_labels[i])     #to print the predicted class and ture class of a mis-classified image
    #print(np.shape(predicted_label))
    test_labels=np.array(test_labels)
    #to create the confusion matrix
    conf_matrix=confusion_matrix(test_labels,predicted_label)        #to create the confusion matrix

    return conf_matrix,mis_clf,mod_test_label
```

[537]:

```
def plot_cnf_matrix_combine(conf_matrix,mod_test_label):
    df_cm = pd.DataFrame(conf_matrix, mod_test_label, mod_test_label)
    plt.figure(figsize=(20,20))
    sn.set(font_scale=1.4) # for label size
    sn.heatmap(df_cm, annot=True, annot_kws={"size":16},fmt='d') # font size
    plt.show()
```

4.1 As misclassification of 1 and 3 (trueclass) as 8 (predicted class) is more.

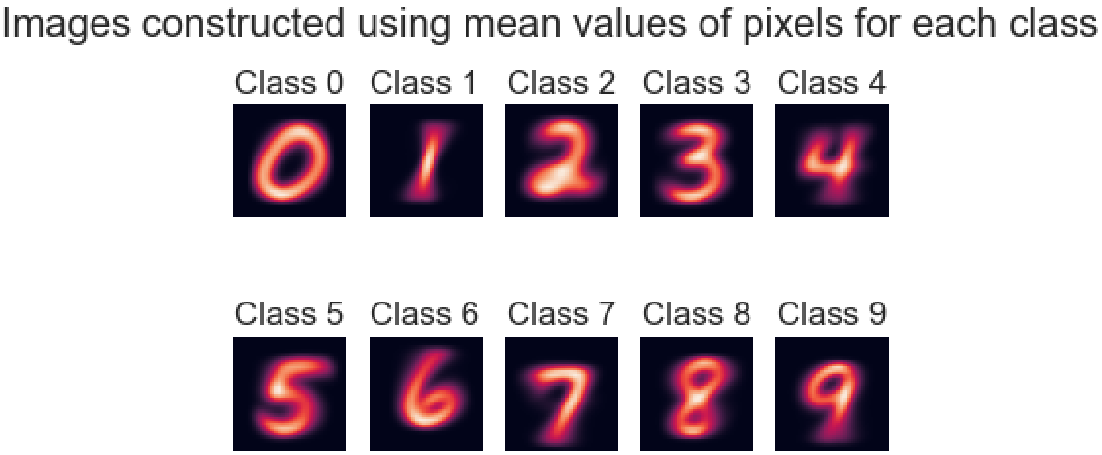
We combine the class 1, class 3 and class 8. The correspoding mean and varinaces of this combined class is $\frac{\mu_1+\mu_3+\mu_8}{3}$ and variance is $\frac{\Sigma_1+\Sigma_3+\Sigma_8}{3}$

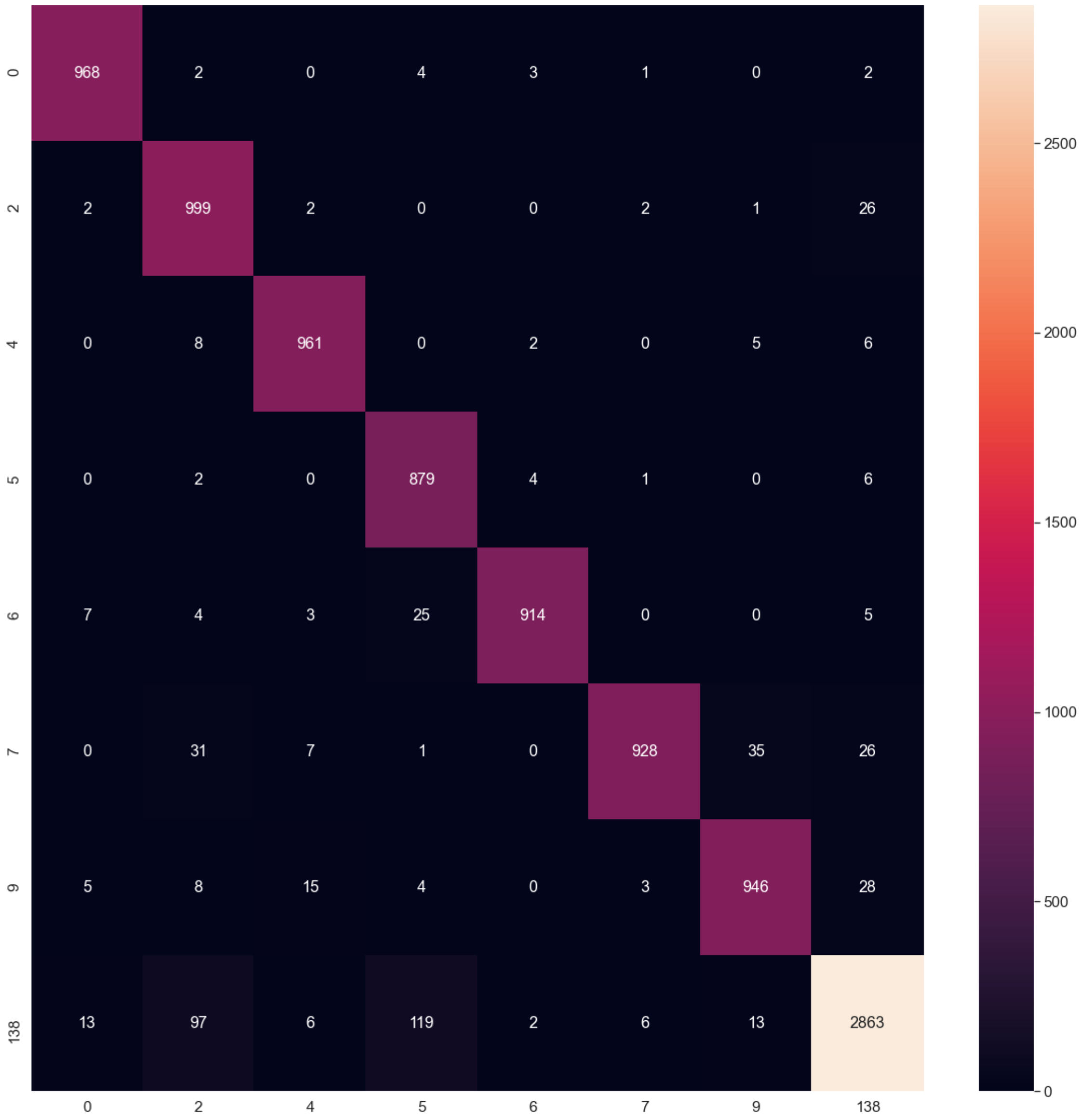
[888]:

```
def choose_dataset(mndata,combined_class):
    train_images, train_labels=mndata.load_training()      #loads training images and their labels
    #print(np.shape(train_images[1]))         #(60,0000,784) there are 60,000 images with each of size 28x28
    #print(np.shape(train_labels))          #(60,000,) labels
    test_images, test_labels=mndata.load_testing()      #loads testing images and their labels
    test_images=np.array(test_images)
    #print(np.shape(test_images))

    mu,covariance,prior_prob = mle_estimation_gaussian(train_images,train_labels)

    #to remove the mean and variances of classed to be combined.
    mask=np.ones(len(mu),dtype='bool')
    mask[combined_class]=False
    mod_mu=mu[mask]
    covar=np.array(covariance)
    mod_covariance=covar[mask]
    mod_prior_prob=prior_prob[mask]

    #to find the mean and variance of combined class
    mask=np.zeros(len(mu),dtype='bool')
    mask[combined_class]=True
    mu138=mu[mask]
    mu138=np.mean(mu138,axis=0)
    mu138=np.reshape(mu138,(1,784))

    mod_mu=np.append(mod_mu,mu138,axis=0)   # append the new mean of the␣
↪combined class


    cov138=covar[mask]
    cov138=np.sum(cov138,axis=0)/len(combined_class)
    cov138=np.reshape(cov138,(1,784,784))

    mod_covariance=np.append(mod_covariance,cov138,axis=0)    #append the new␣
↪varainace of the combined class

    p138=prior_prob[mask]
    p138=np.mean(p138)
    mod_prior_prob=np.append(mod_prior_prob,p138)    #append the new prior␣
↪probability of the combined class

    # print(np.shape(mod_covariance))
  ␣
↪conf_matrix,mis_clf,mod_test_label=bayes_clasfr_gaussian_combine(test_images,test_labels,mod_

    plot_cnf_matrix_combine(conf_matrix,mod_test_label)

    perc_misclf= mis_clf/len(test_labels)

    print(f"The % of misclassification is {perc_misclf*100}% for MLE estimation␣
↪assuming the data is sampled from Multivariate Gaussian Distribution with␣
↪classes 1,3,8 combined")
```

[858]:

```
mndata=MNIST(r"Dataset_spr_2")  #Extract MNIST data from the specified folder.␣
↪mndata has images of numbers 0-9 with their respective labels

combined_class=[1,3,8]

choose_dataset(mndata,combined_class)
```


Images constructed using mean values of pixels for each class
Class 0
Class 1
Class 2
Class 3
Class 4
Class 5
Class 6
Class 7
Class 8
Class 9



The % of misclassification is 5.42% for MLE estimation assuming the data is sampled from Multivariate Gaussian Distribution with classes 1,3,8 combined

4.2 b) Exponential Distribution

Estimate of paramter λ for i^{th} class is a vector with 784 values with j^{th} value equal to:

$$\lambda_{ij} = \sum_{l=1}^{m} \frac{m}{x_{jl}}, \tag{3}$$

where, m is the number of images of i^{th} particular class and
x_{jl} is the j^{th} pixel value of l^{th} image.

4.3 MLE Estimation for Training Data with Class Conditional densities modeled as Multivariate Exponential Distribution

[864]:

```
def mle_estimation_exp(train_images,train_labels,lamda_smoothing):

    classes=[[] for i in range(np.max(train_labels)+1)]  #creates a list of 10␣
 ↪empty lists with each list referring to a class in our case (10 classes)

    for i in range(len(train_labels)):
        classes[train_labels[i]].append(train_images[i]) # the images with label␣
 ↪k is added to the class k.




    lamda= np.array([1/((np.mean(classes[i],axis=0))) for i in range(np.
 ↪max(train_labels)+1)])  #generate the MLE estimate of parameter lamda vector␣
 ↪for each class

    lamda[lamda==np.inf]=lamda_smoothing  # np.unique(lamda)[-2]
    prior_prob=np.zeros((np.max(train_labels)+1,1))
    for i in range((np.max(train_labels)+1)):
        prior_prob[i]=len(classes[i])/(np.shape(train_images)[0])

#     print("prior probabilities")
#     print(prior_prob)
#     #to plot the images with pixel values of a class (0-9) equal to the mean␣
 ↪values of the corresponding classes
    plt.suptitle("Images constructed using mean values of pixels for each class")
    for i in range(np.max(train_labels)+1):
        plt.subplot(2,5,i+1)
        train_imag1=lamda[i][:]
        train_imag1=np.array(1/(train_imag1))
        train_imag1=train_imag1.reshape(int(np.sqrt(len(train_imag1))),int(np.
 ↪sqrt(len(train_imag1))))
        plt.imshow(train_imag1)
        plt.axis('off')
        plt.title("Class "+str(i))

    plt.show()

    return lamda,prior_prob
```

4.4 Bayes Classfier for Test Data with Class Conditional densities modeled as Multivariate Gaussian Distribution

[865]:

```python
def bayes_clasfr(test_images,test_labels,lamda,prior_prob):
    mis_clf=0
    predicted_label=[]

    for i in range(len(test_labels)):
        dmy=np.array(test_images[i][:])   #load the feature vector of particular image that needs to be classified
        #dmy=np.reshape(dmy,(1,784))
        pst_prob=np.zeros((np.max(test_labels)+1))          #to store posterior probabilites given the feature vector
        for j in range(np.max(test_labels)+1):

            lamda_temp=np.array(lamda[j,:])           #load the mean vector of jth class
            pst_prob[j]= np.sum(np.log(lamda_temp))+np.sum(-lamda_temp*dmy)+np.log(prior_prob[j])#np.sum(np.log(lamda_temp))-np.sum(lamda*dmy) + np.log(prior_prob[j])   #find the PDF of the feature vector that follows the given distribution
        predicted_label.append(np.argmax(pst_prob))                                  #find the index at which the posterior probability is maximum

        if predicted_label[i]!=test_labels[i]:
            mis_clf+=1
            #print(i,mis_clf,predicted_label[i],test_labels[i])                  #to print the predicted class and ture class of a mis-classified image

    #print(np.shape(predicted_label))
    test_labels=np.array(test_labels)
    predicted_label=np.array(predicted_label)

    #print(np.shape(predicted_label))

    conf_matrix=confusion_matrix(test_labels,predicted_label)      #to create the confusion matrix
    #print(mis_clf)
    return conf_matrix,mis_clf
```

4.5 Training and Testing The Model:

[879]:

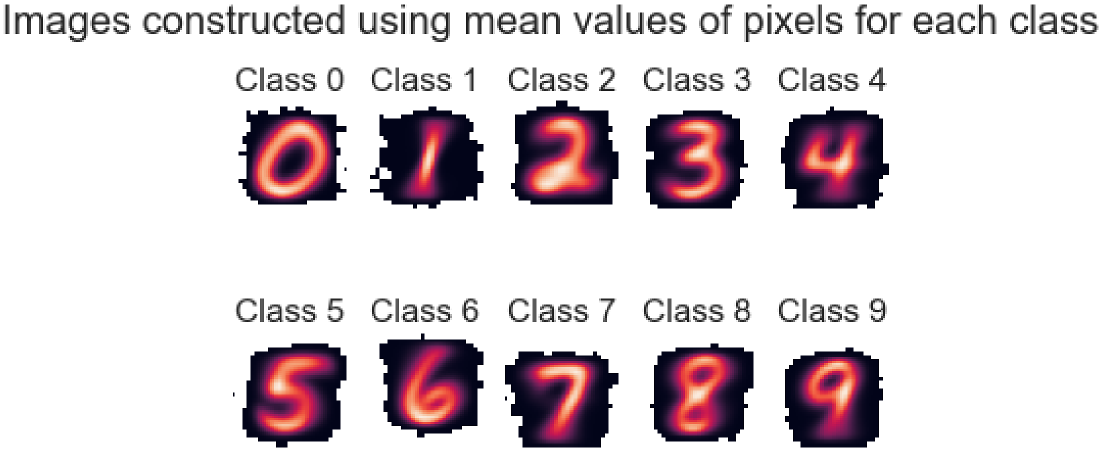
```python
mndata=MNIST(r"Dataset_spr_2")  #Extract MNIST data from the specified folder. mndata has images of numbers 0-9 with their respective labels
```

```
train_images, train_labels=mndata.load_training()      #loads training images␣
 ↪and their labels
test_images, test_labels=mndata.load_testing()     #loads testing images and␣
 ↪their labels
test_images=np.array(test_images)

print(f"Total no. of images uses for training : {str(len(train_images))}")
print(f"No. of pixel values for each image : {str(np.shape(train_images)[1])}")
print(f"No. of classes is : {str(np.max(train_labels)+1)}")

#print(f"Classifier : Bayes Classifier and Class Conditional densities for␣
 ↪training Data is Mulitvariate Exponential Distribution")
#the below values of lamda are for replacing the lamda value with one particular␣
 ↪value in a iteration when it is equal to infinity

lamda_smoothing=[0,0.1,0.18,0.3,0.5,0.8,1]
mis_clf=np.zeros((len(lamda_smoothing),1))
perc_misclf=np.zeros((len(lamda_smoothing),1))

for i in range(len(lamda_smoothing)):
    lamda,prior_prob =␣
 ↪mle_estimation_exp(train_images,train_labels,lamda_smoothing[i])
    print(f"Classifier : Bayes Classifier and Class Conditional densities for␣
 ↪training Data is Mulitvariate Exponential Distribution")
    for j in range(np.max(train_labels)+1):
        print("Prior Probability of class " + str(i) + " is "  +␣
 ↪str(prior_prob[j,0]))

    conf_matrix,mis_clf[i] =␣
 ↪bayes_clasfr(test_images,test_labels,lamda,prior_prob)
    plot_cnf_matrix(conf_matrix)
    perc_misclf[i]= mis_clf[i]/len(test_labels)
    print(f"The % of misclassification is {str(perc_misclf[i,0]*100)}% when the␣
 ↪noise added to the lambda is {lamda_smoothing[i]}")


print(f"The % of misclassification is less for noise value␣
 ↪{lamda_smoothing[int(np.argmin(perc_misclf))]} added to the lamda when lambda␣
 ↪goes to infinity")
print(f"The corresponding % misclassification is {np.min(perc_misclf)*100}")
```

```
Total no. of images uses for training : 60000
No. of pixel values for each image : 784
No. of classes is : 10
```

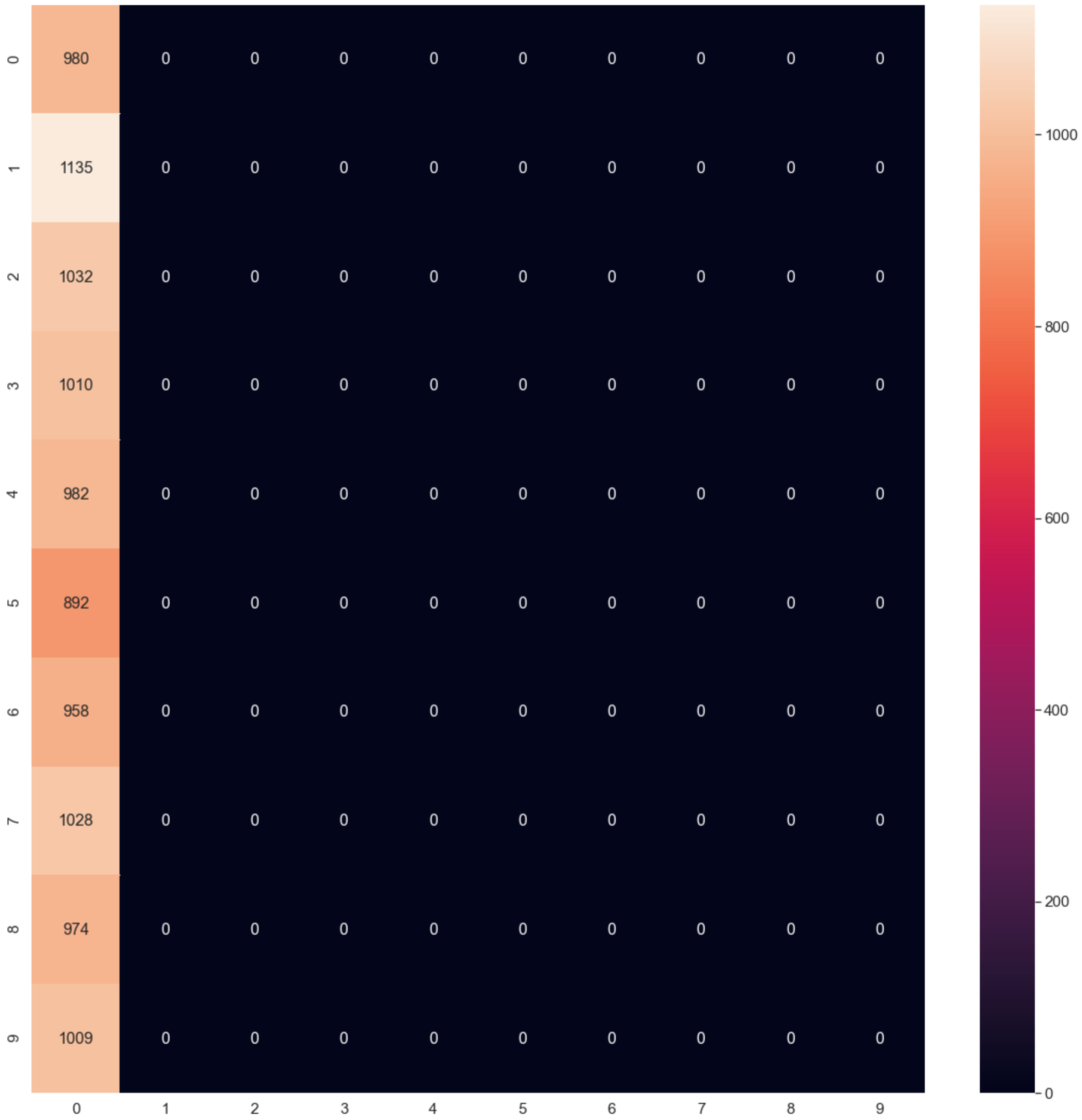
```
<ipython-input-864-c7f80f3173d1>:10: RuntimeWarning: divide by zero encountered
```

```
in true_divide
  lamda= np.array([1/((np.mean(classes[i],axis=0))) for i in
range(np.max(train_labels)+1)])  #generate the MLE estimate of parameter lamda
vector for each class
<ipython-input-864-c7f80f3173d1>:24: RuntimeWarning: divide by zero encountered
in true_divide
  train_imag1=np.array(1/(train_imag1))
```

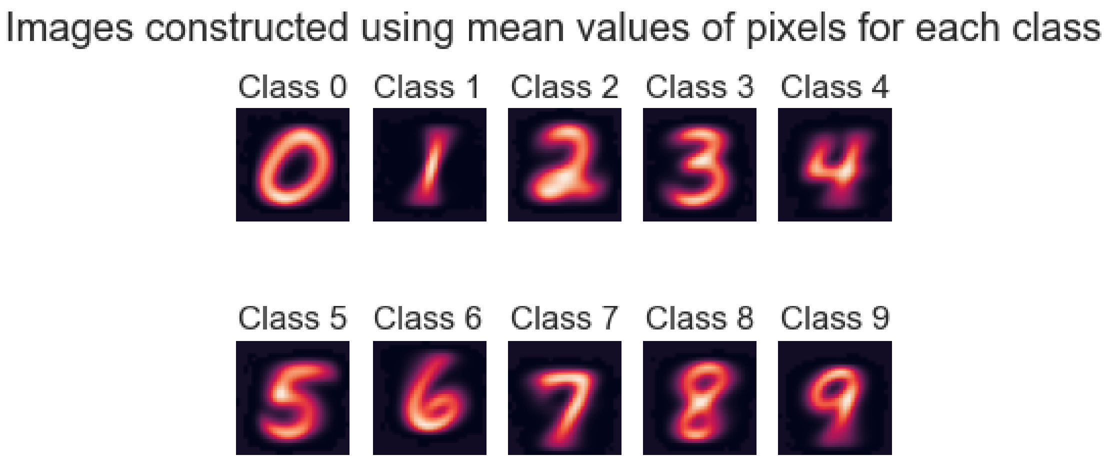


```
Classifier : Bayes Classifier and Class Conditional densities for training Data
is Mulitvariate Exponential Distribution
Prior Probability of class 0 is 0.09871666666666666
Prior Probability of class 0 is 0.11236666666666667
Prior Probability of class 0 is 0.0993
Prior Probability of class 0 is 0.10218333333333333
Prior Probability of class 0 is 0.09736666666666667
Prior Probability of class 0 is 0.09035
Prior Probability of class 0 is 0.09863333333333334
Prior Probability of class 0 is 0.10441666666666667
Prior Probability of class 0 is 0.09751666666666667
Prior Probability of class 0 is 0.09915
```

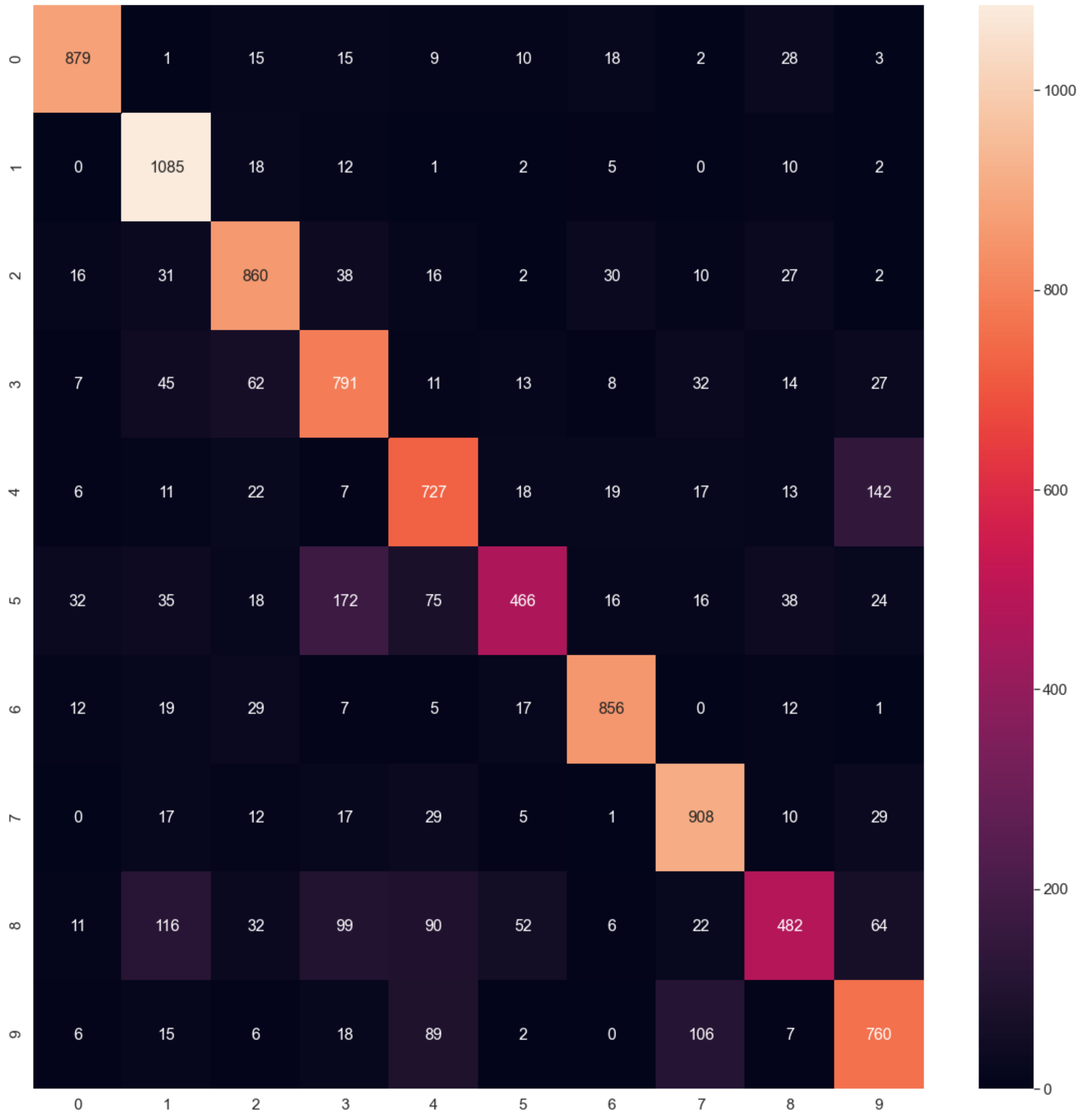
```
<ipython-input-865-a702fd3e510d>:12: RuntimeWarning: divide by zero encountered
in log
  pst_prob[j]= np.sum(np.log(lamda_temp))+np.sum(-lamda_temp*dmy)+np.log(prior_p
rob[j])#np.sum(np.log(lamda_temp))-np.sum(lamda*dmy) + np.log(prior_prob[j])
#find the PDF of the feature vector that follows the given distribution
```



```
The % of misclassification is 90.2% when the noise added to the lambda is 0
```

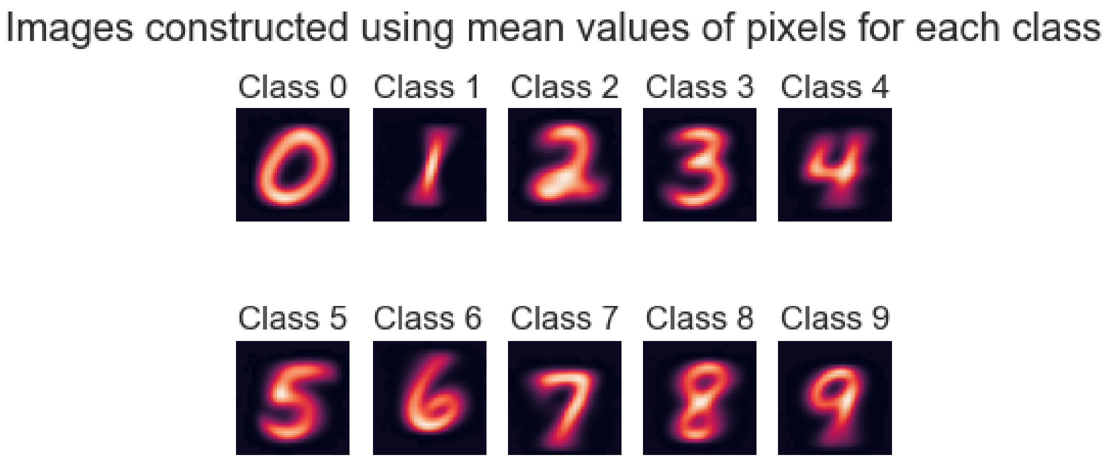


```
Classifier : Bayes Classifier and Class Conditional densities for training Data
is Mulitvariate Exponential Distribution
Prior Probability of class 1 is 0.09871666666666666
Prior Probability of class 1 is 0.11236666666666667
Prior Probability of class 1 is 0.0993
Prior Probability of class 1 is 0.10218333333333333
Prior Probability of class 1 is 0.09736666666666667
Prior Probability of class 1 is 0.09035
Prior Probability of class 1 is 0.09863333333333334
Prior Probability of class 1 is 0.10441666666666667
Prior Probability of class 1 is 0.09751666666666667
Prior Probability of class 1 is 0.09915
```

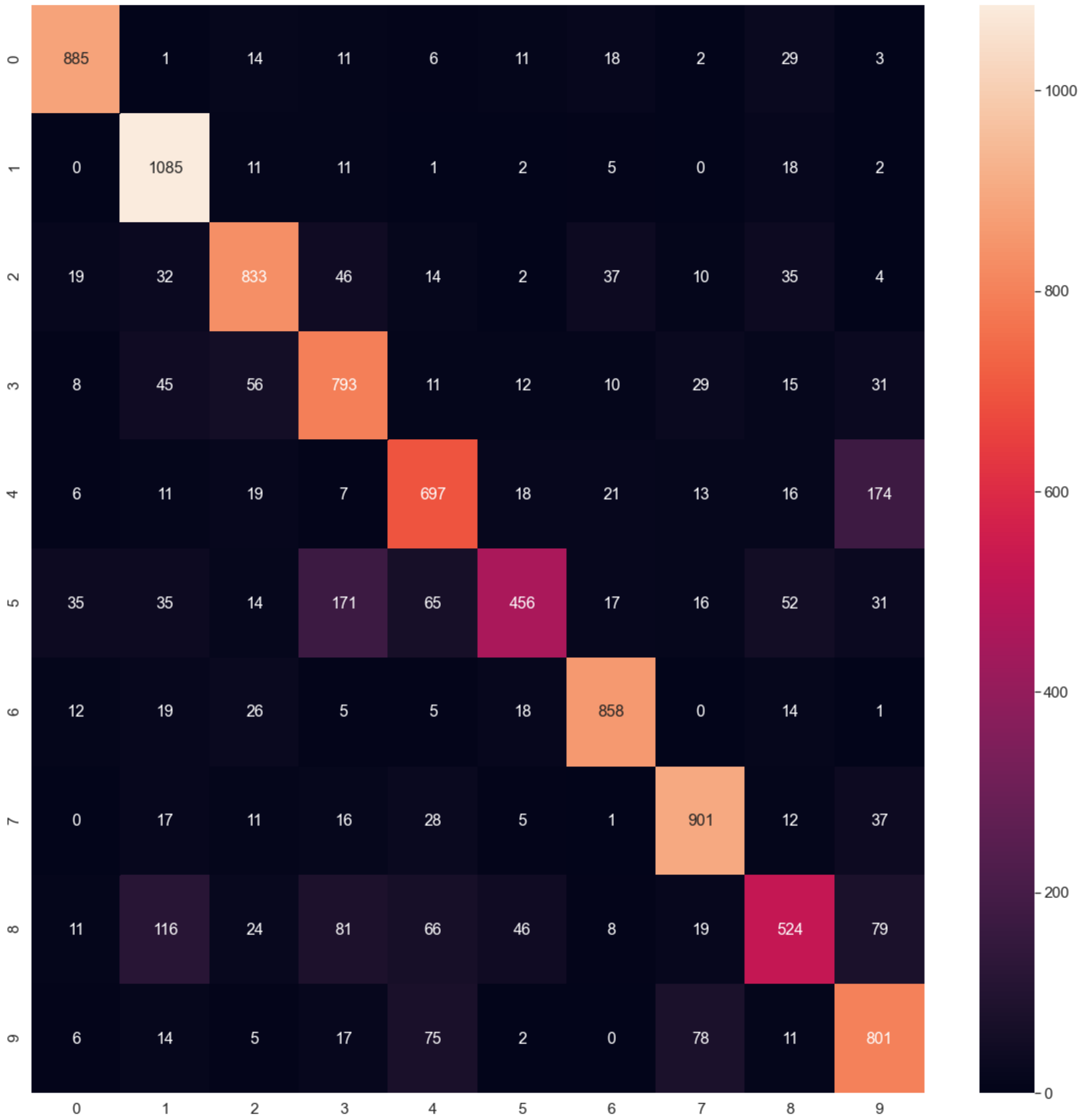


	0	1	2	3	4	5	6	7	8	9
0	879	1	15	15	9	10	18	2	28	3
1	0	1085	18	12	1	2	5	0	10	2
2	16	31	860	38	16	2	30	10	27	2
3	7	45	62	791	11	13	8	32	14	27
4	6	11	22	7	727	18	19	17	13	142
5	32	35	18	172	75	466	16	16	38	24
6	12	19	29	7	5	17	856	0	12	1
7	0	17	12	17	29	5	1	908	10	29
8	11	116	32	99	90	52	6	22	482	64
9	6	15	6	18	89	2	0	106	7	760

The % of misclassification is 21.86% when the noise added to the lambda is 0.1

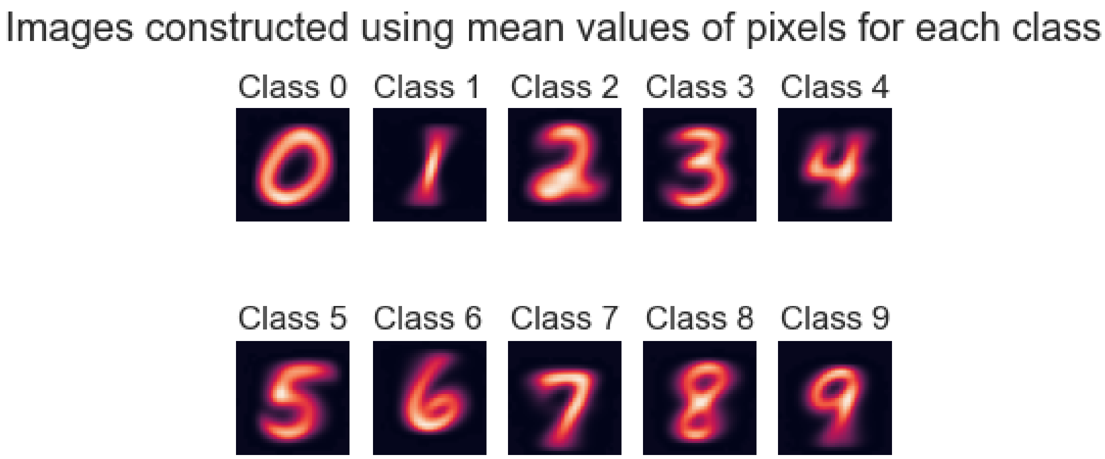


```
Classifier : Bayes Classifier and Class Conditional densities for training Data
is Mulitvariate Exponential Distribution
Prior Probability of class 2 is 0.09871666666666666
Prior Probability of class 2 is 0.11236666666666667
Prior Probability of class 2 is 0.0993
Prior Probability of class 2 is 0.10218333333333333
Prior Probability of class 2 is 0.09736666666666667
Prior Probability of class 2 is 0.09035
Prior Probability of class 2 is 0.09863333333333334
Prior Probability of class 2 is 0.10441666666666667
Prior Probability of class 2 is 0.09751666666666667
Prior Probability of class 2 is 0.09915
```

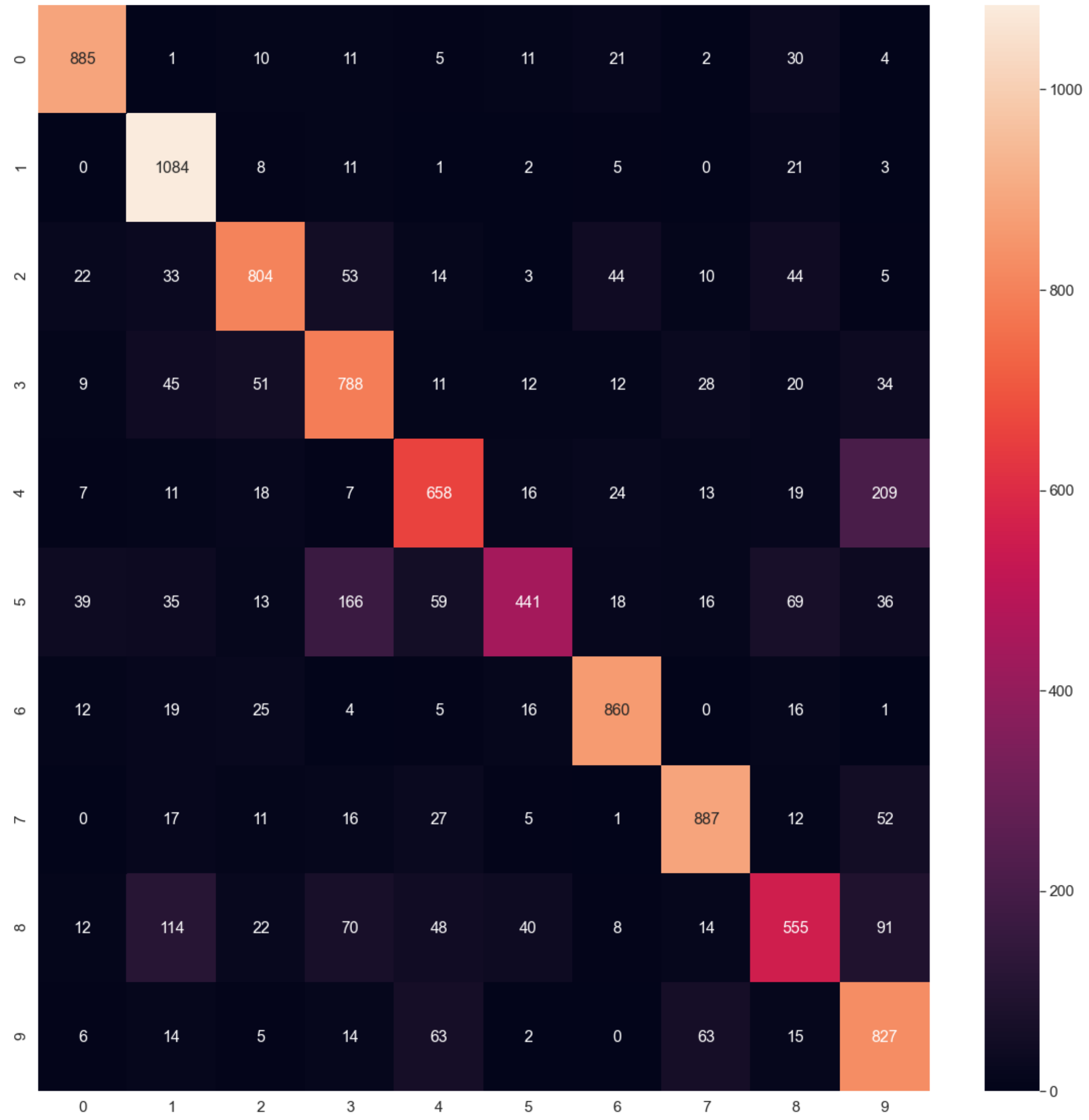


	0	1	2	3	4	5	6	7	8	9
0	885	1	14	11	6	11	18	2	29	3
1	0	1085	11	11	1	2	5	0	18	2
2	19	32	833	46	14	2	37	10	35	4
3	8	45	56	793	11	12	10	29	15	31
4	6	11	19	7	697	18	21	13	16	174
5	35	35	14	171	65	456	17	16	52	31
6	12	19	26	5	5	18	858	0	14	1
7	0	17	11	16	28	5	1	901	12	37
8	11	116	24	81	66	46	8	19	524	79
9	6	14	5	17	75	2	0	78	11	801

The % of misclassification is 21.67% when the noise added to the lambda is 0.18

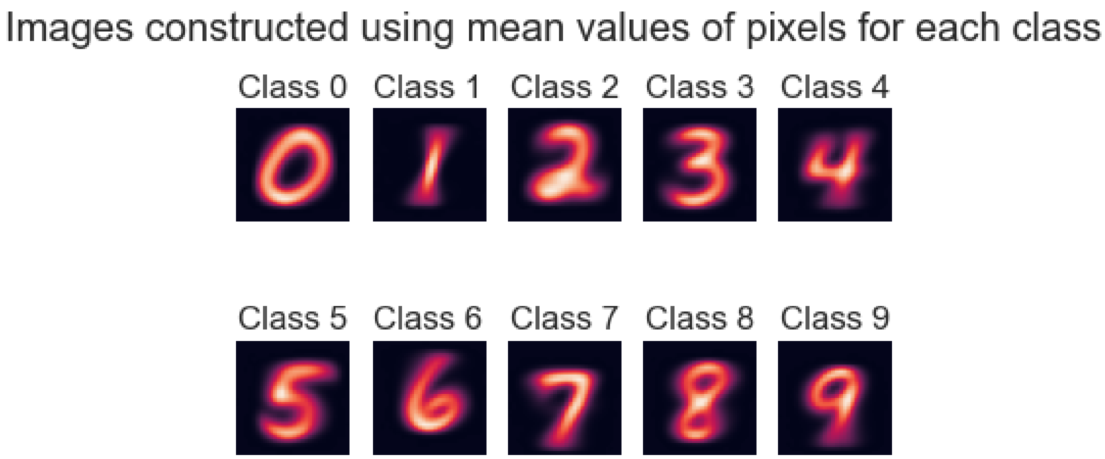


```
Classifier : Bayes Classifier and Class Conditional densities for training Data
is Mulitvariate Exponential Distribution
Prior Probability of class 3 is 0.09871666666666666
Prior Probability of class 3 is 0.11236666666666667
Prior Probability of class 3 is 0.0993
Prior Probability of class 3 is 0.10218333333333333
Prior Probability of class 3 is 0.09736666666666667
Prior Probability of class 3 is 0.09035
Prior Probability of class 3 is 0.09863333333333334
Prior Probability of class 3 is 0.10441666666666667
Prior Probability of class 3 is 0.09751666666666667
Prior Probability of class 3 is 0.09915
```

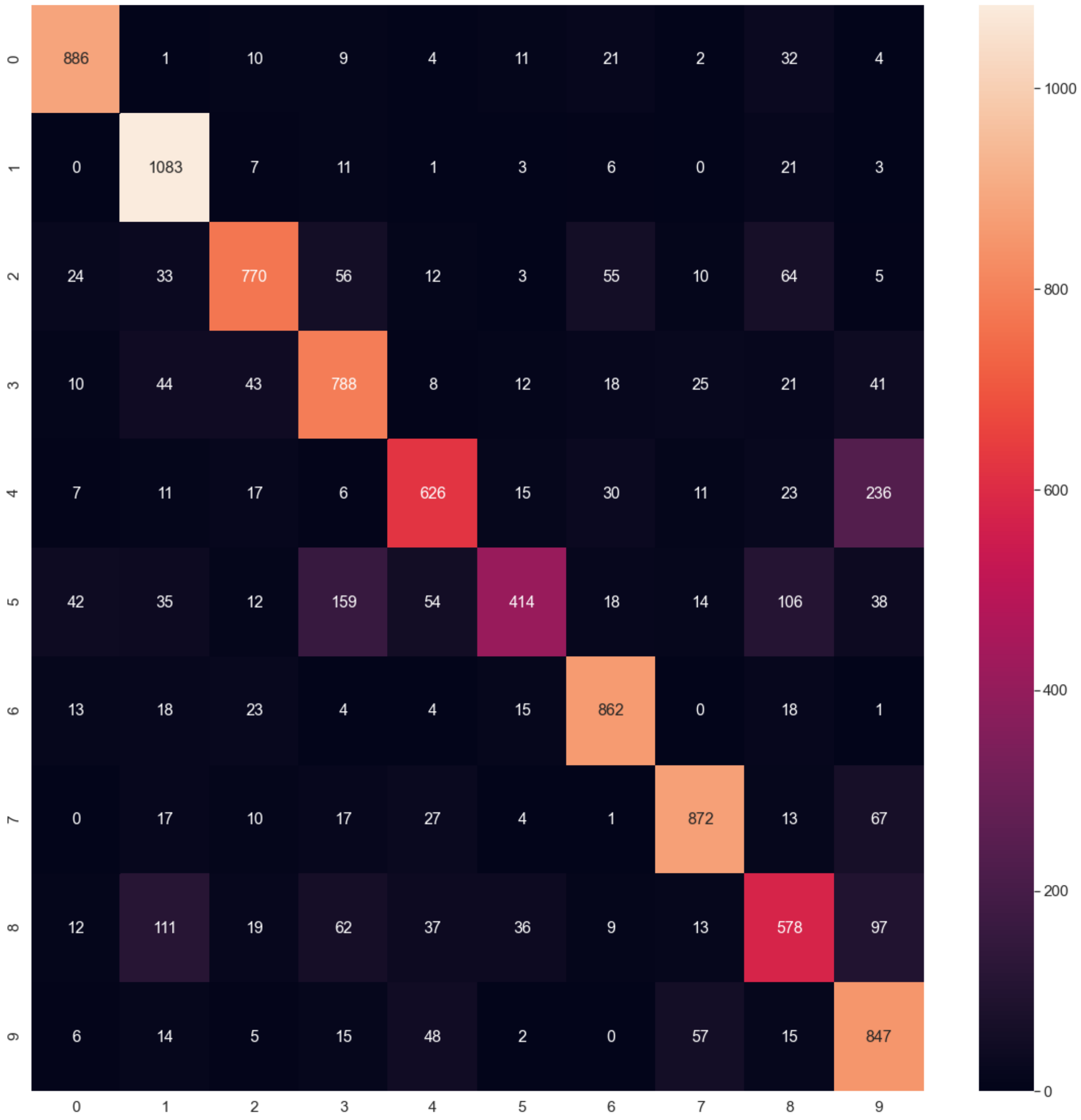


	0	1	2	3	4	5	6	7	8	9
0	885	1	10	11	5	11	21	2	30	4
1	0	1084	8	11	1	2	5	0	21	3
2	22	33	804	53	14	3	44	10	44	5
3	9	45	51	788	11	12	12	28	20	34
4	7	11	18	7	658	16	24	13	19	209
5	39	35	13	166	59	441	18	16	69	36
6	12	19	25	4	5	16	860	0	16	1
7	0	17	11	16	27	5	1	887	12	52
8	12	114	22	70	48	40	8	14	555	91
9	6	14	5	14	63	2	0	63	15	827

```
The % of misclassification is 22.11% when the noise added to the lambda is 0.3
```

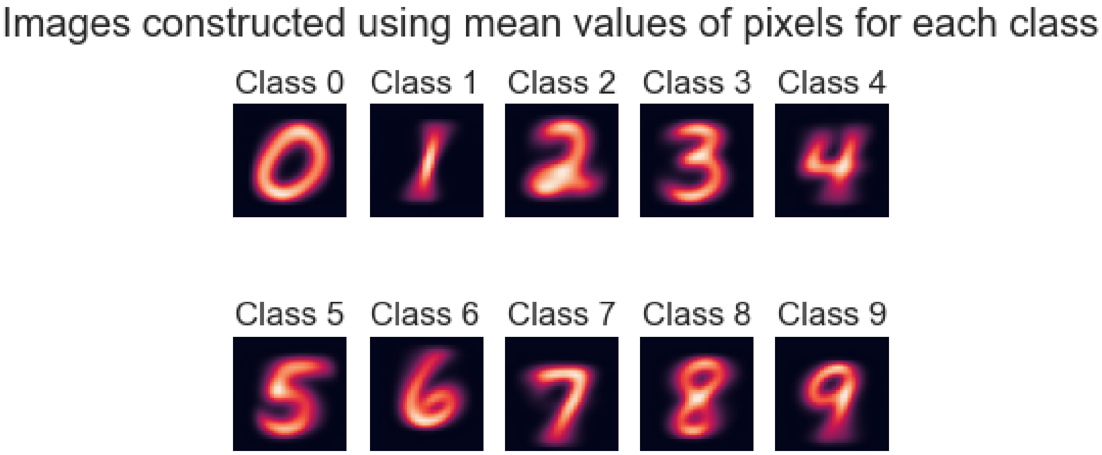


```
Classifier : Bayes Classifier and Class Conditional densities for training Data
is Mulitvariate Exponential Distribution
Prior Probability of class 4 is 0.09871666666666666
Prior Probability of class 4 is 0.11236666666666667
Prior Probability of class 4 is 0.0993
Prior Probability of class 4 is 0.10218333333333333
Prior Probability of class 4 is 0.09736666666666667
Prior Probability of class 4 is 0.09035
Prior Probability of class 4 is 0.09863333333333334
Prior Probability of class 4 is 0.10441666666666667
Prior Probability of class 4 is 0.09751666666666667
Prior Probability of class 4 is 0.09915
```

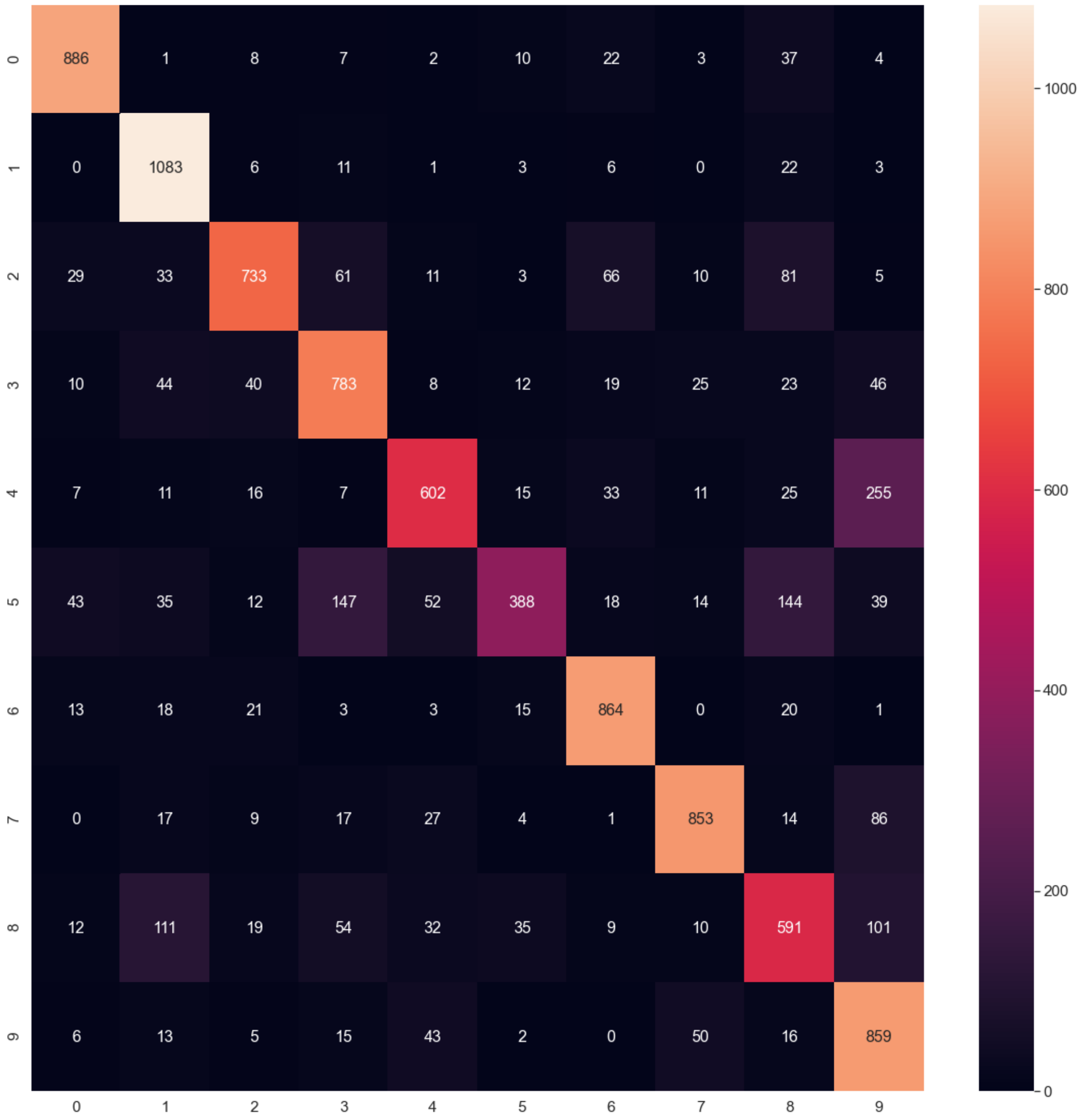


	0	1	2	3	4	5	6	7	8	9
0	886	1	10	9	4	11	21	2	32	4
1	0	1083	7	11	1	3	6	0	21	3
2	24	33	770	56	12	3	55	10	64	5
3	10	44	43	788	8	12	18	25	21	41
4	7	11	17	6	626	15	30	11	23	236
5	42	35	12	159	54	414	18	14	106	38
6	13	18	23	4	4	15	862	0	18	1
7	0	17	10	17	27	4	1	872	13	67
8	12	111	19	62	37	36	9	13	578	97
9	6	14	5	15	48	2	0	57	15	847

The % of misclassification is 22.74% when the noise added to the lambda is 0.5

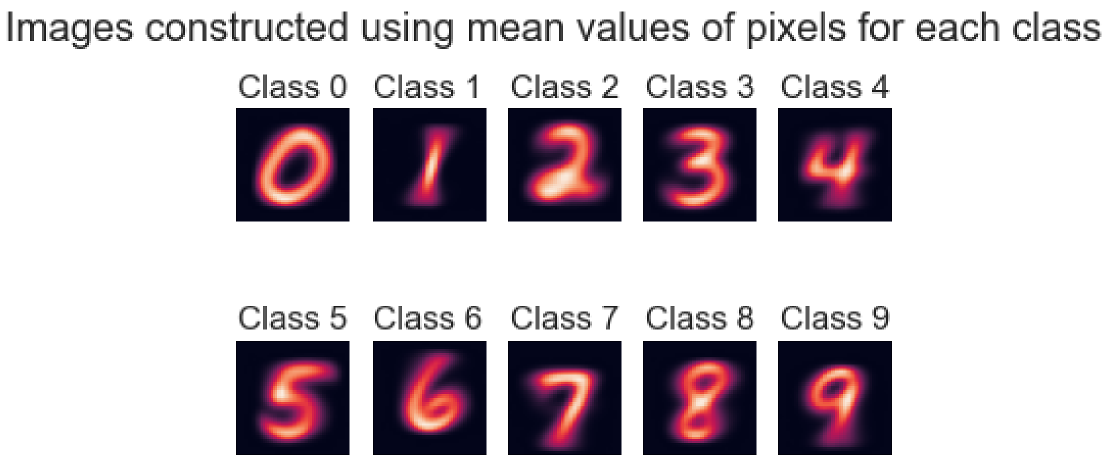


```
Classifier : Bayes Classifier and Class Conditional densities for training Data
is Mulitvariate Exponential Distribution
Prior Probability of class 5 is 0.09871666666666666
Prior Probability of class 5 is 0.11236666666666667
Prior Probability of class 5 is 0.0993
Prior Probability of class 5 is 0.10218333333333333
Prior Probability of class 5 is 0.09736666666666667
Prior Probability of class 5 is 0.09035
Prior Probability of class 5 is 0.09863333333333334
Prior Probability of class 5 is 0.10441666666666667
Prior Probability of class 5 is 0.09751666666666667
Prior Probability of class 5 is 0.09915
```

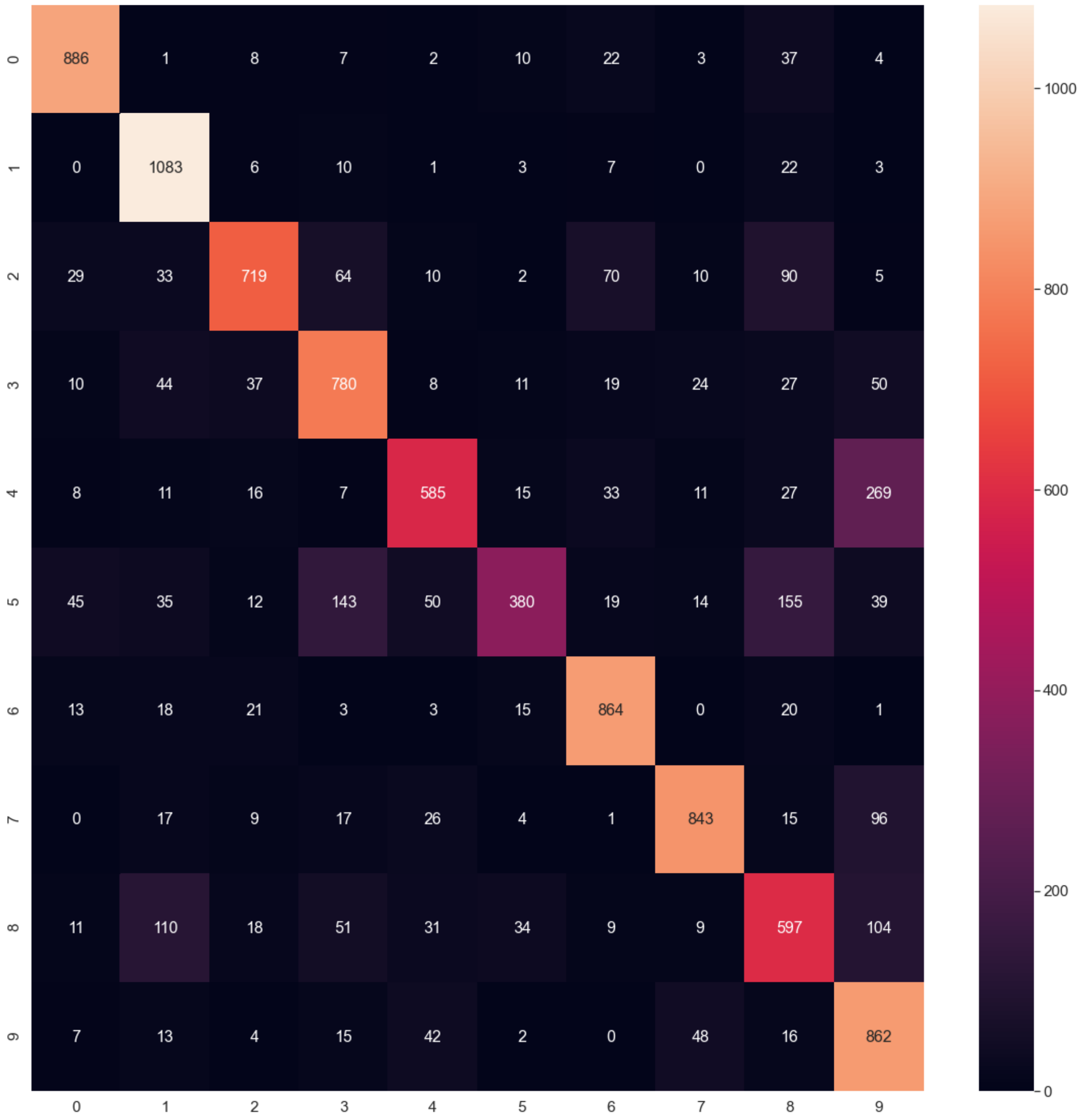


	0	1	2	3	4	5	6	7	8	9
0	886	1	8	7	2	10	22	3	37	4
1	0	1083	6	11	1	3	6	0	22	3
2	29	33	733	61	11	3	66	10	81	5
3	10	44	40	783	8	12	19	25	23	46
4	7	11	16	7	602	15	33	11	25	255
5	43	35	12	147	52	388	18	14	144	39
6	13	18	21	3	3	15	864	0	20	1
7	0	17	9	17	27	4	1	853	14	86
8	12	111	19	54	32	35	9	10	591	101
9	6	13	5	15	43	2	0	50	16	859

```
The % of misclassification is 23.580000000000002% when the noise added to the
lambda is 0.8
```



```
Classifier : Bayes Classifier and Class Conditional densities for training Data
is Mulitvariate Exponential Distribution
Prior Probability of class 6 is 0.09871666666666666
Prior Probability of class 6 is 0.11236666666666667
Prior Probability of class 6 is 0.0993
Prior Probability of class 6 is 0.10218333333333333
Prior Probability of class 6 is 0.09736666666666667
Prior Probability of class 6 is 0.09035
Prior Probability of class 6 is 0.09863333333333334
Prior Probability of class 6 is 0.10441666666666667
Prior Probability of class 6 is 0.09751666666666667
Prior Probability of class 6 is 0.09915
```



```
The % of misclassification is 24.01% when the noise added to the lambda is 1
The % of misclassification is less for noise value 0.18 added to the lamda when
lambda goes to infinity
The corresponding % misclassification is 21.67
```

4.6 Problem 3:

4.7 We have considered MNIST Fashion Dataset and used Multivariate Gaussian and Multivariate Exponential Distribution for modeling Class Conditional Densities

Training Data consists of 60,000 small square 28×28 pixel grayscale images of fashion items of 10 types.Testing Data consists of 10,000 small square 28×28 pixel grayscale images of fashion items of 10 types. The mapping of all 0-9 integers to class labels is listed below.

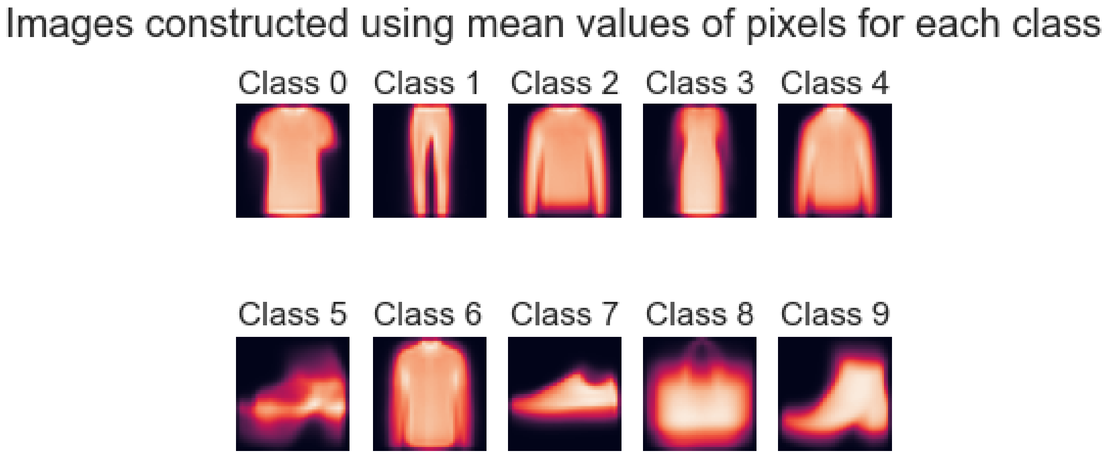
0 : T-shirt/top
1 : Trouser
2 : Pullover
3 : Dress
4 : Coat
5 : Sandal
6 : Shirt
7 : Sneaker
8 : Bag
9 : Ankle boot

Trianing and Testing with Class conditional Densities modeled as Multivariate Gaussian Distribution

[870]:
```
mndata=MNIST(r"MNIST_fashion") #Extract MNIST data from the specified folder.␣
 ↪mndata has images of numbers 0-9 with their respective labels

train_images, train_labels=mndata.load_training()      #loads training images␣
 ↪and their labels
#print(np.shape(train_images[1]))         #(60,0000,784) there are 60,000 images␣
 ↪with each of size 28x28
#print(np.shape(train_labels))          #(60,000,) labels
test_images, test_labels=mndata.load_testing()     #loads testing images and␣
 ↪their labels
test_images=np.array(test_images)
#print(np.shape(test_images))

mu,covariance,prior_prob = mle_estimation_gaussian(train_images,train_labels)

print(f"Total no. of images uses for training : {str(len(train_images))}")
print(f"No. of pixel values for each image : {str(np.shape(train_images)[1])}")
print(f"No. of classes is : {str(np.max(train_labels)+1)}")
print(f"Classifier : Bayes Classifier and Class Conditional densities for␣
 ↪training Data : Mulitvariate Gaussian Distribution")
for i in range(np.max(train_labels)+1):
    print("Prior Probability of class " + str(i) + " is "  +␣
 ↪str(prior_prob[i,0]))

conf_matrix,mis_clf =␣
 ↪bayes_clasfr_gaussian(test_images,test_labels,mu,covariance,prior_prob)

plot_cnf_matrix(conf_matrix)

perc_misclf= mis_clf/len(test_labels)
```

```python
print(f"The % of misclassification is {str(perc_misclf*100)}% for MLE estimation␣
 ↪assuming the data is sampled from Multivariate Gaussian Distribution")
```

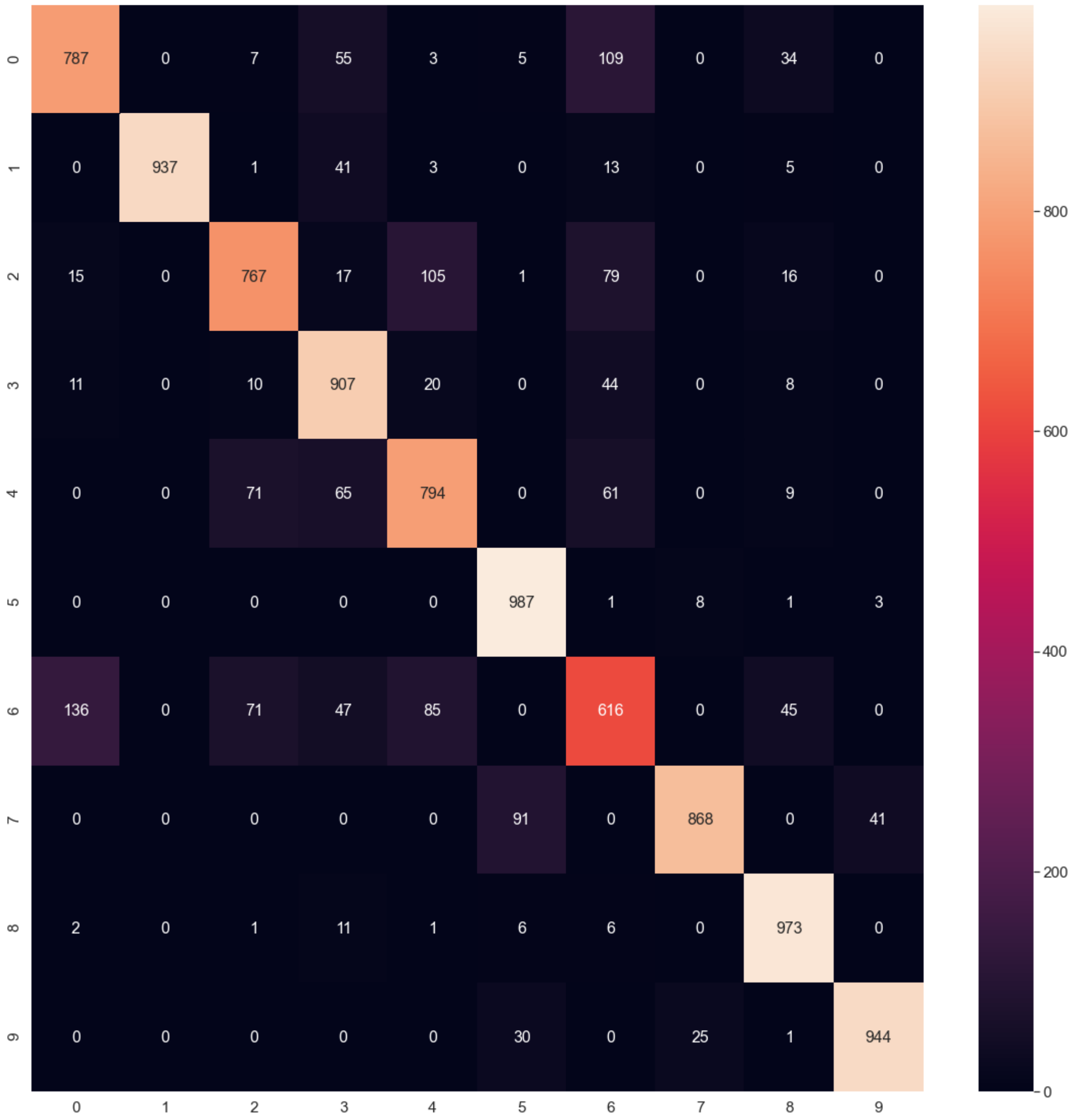
```
Total no. of images uses for training : 60000
No. of pixel values for each image : 784
No. of classes is : 10
Classifier : Bayes Classifier and Class Conditional densities for training Data
: Mulitvariate Gaussian Distribution
Prior Probability of class 0 is 0.1
Prior Probability of class 1 is 0.1
Prior Probability of class 2 is 0.1
Prior Probability of class 3 is 0.1
Prior Probability of class 4 is 0.1
Prior Probability of class 5 is 0.1
Prior Probability of class 6 is 0.1
Prior Probability of class 7 is 0.1
Prior Probability of class 8 is 0.1
Prior Probability of class 9 is 0.1
```



Images constructed using mean values of pixels for each class

Class 0 Class 1 Class 2 Class 3 Class 4

Class 5 Class 6 Class 7 Class 8 Class 9



	0	1	2	3	4	5	6	7	8	9
0	787	0	7	55	3	5	109	0	34	0
1	0	937	1	41	3	0	13	0	5	0
2	15	0	767	17	105	1	79	0	16	0
3	11	0	10	907	20	0	44	0	8	0
4	0	0	71	65	794	0	61	0	9	0
5	0	0	0	0	0	987	1	8	1	3
6	136	0	71	47	85	0	616	0	45	0
7	0	0	0	0	0	91	0	868	0	41
8	2	0	1	11	1	6	6	0	973	0
9	0	0	0	0	0	30	0	25	1	944

The % of misclassification is 14.2% for MLE estimation assuming the data is sampled from Multivariate Gaussian Distribution

Trianing and Testing with Class conditional Densities modeled as Multivariate Exponentail Distribution

[878]:

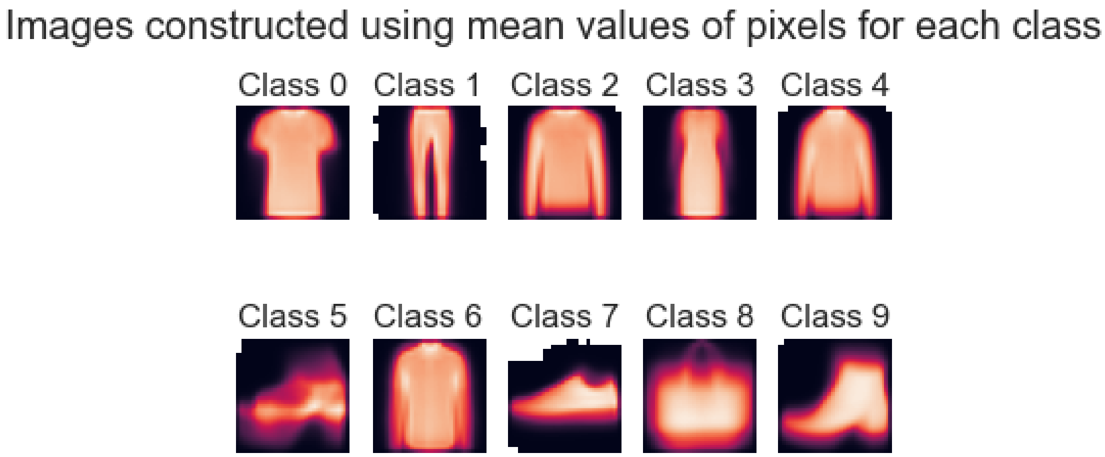
```
mndata=MNIST(r"MNIST_fashion")  #Extract MNIST data from the specified folder. mndata has images of numbers 0-9 with their respective labels

train_images, train_labels=mndata.load_training()      #loads training images and their labels
```

```
test_images, test_labels=mndata.load_testing()     #loads testing images and␣
 ↪their labels
test_images=np.array(test_images)

#the below values of lamda are for replacing the lamda value with one particular␣
 ↪value in a iteration when it is equal to infinity
lamda_smoothing=[0,0.1,0.18,0.3,0.5,0.8,1]
mis_clf=np.zeros((len(lamda_smoothing),1))
perc_misclf=np.zeros((len(lamda_smoothing),1))

print(f"Total no. of images uses for training : {str(len(train_images))}")
print(f"No. of pixel values for each image : {str(np.shape(train_images)[1])}")
print(f"No. of classes is : {str(np.max(train_labels)+1)}")

#print(f"Classifier : Bayes Classifier and Class Conditional densities for␣
 ↪training Data is Mulitvariate Exponential Distribution")
for i in range(len(lamda_smoothing)):
    lamda,prior_prob =␣
 ↪mle_estimation_exp(train_images,train_labels,lamda_smoothing[i])
    print(f"Classifier : Bayes Classifier and Class Conditional densities for␣
 ↪training Data is Mulitvariate Exponential Distribution")
    for j in range(np.max(train_labels)+1):
        print("Prior Probability of class " + str(j) + " is "  +␣
 ↪str(prior_prob[j,0]))

    conf_matrix,mis_clf[i] =␣
 ↪bayes_clasfr(test_images,test_labels,lamda,prior_prob)
    plot_cnf_matrix(conf_matrix)
    perc_misclf[i]= mis_clf[i]/len(test_labels)
    print(f"The % of misclassification is {perc_misclf[i]*100}% when the noise␣
 ↪added to the lambda is {lamda_smoothing[i]}")

print(f"The % of misclassification is less for noise value␣
 ↪{lamda_smoothing[int(np.argmin(perc_misclf))]} added to the lamda when lambda␣
 ↪goes to infinity")
print(f"The corresponding % misclassification is {np.min(perc_misclf)*100}")
```

```
Total no. of images uses for training : 60000
No. of pixel values for each image : 784
No. of classes is : 10
```

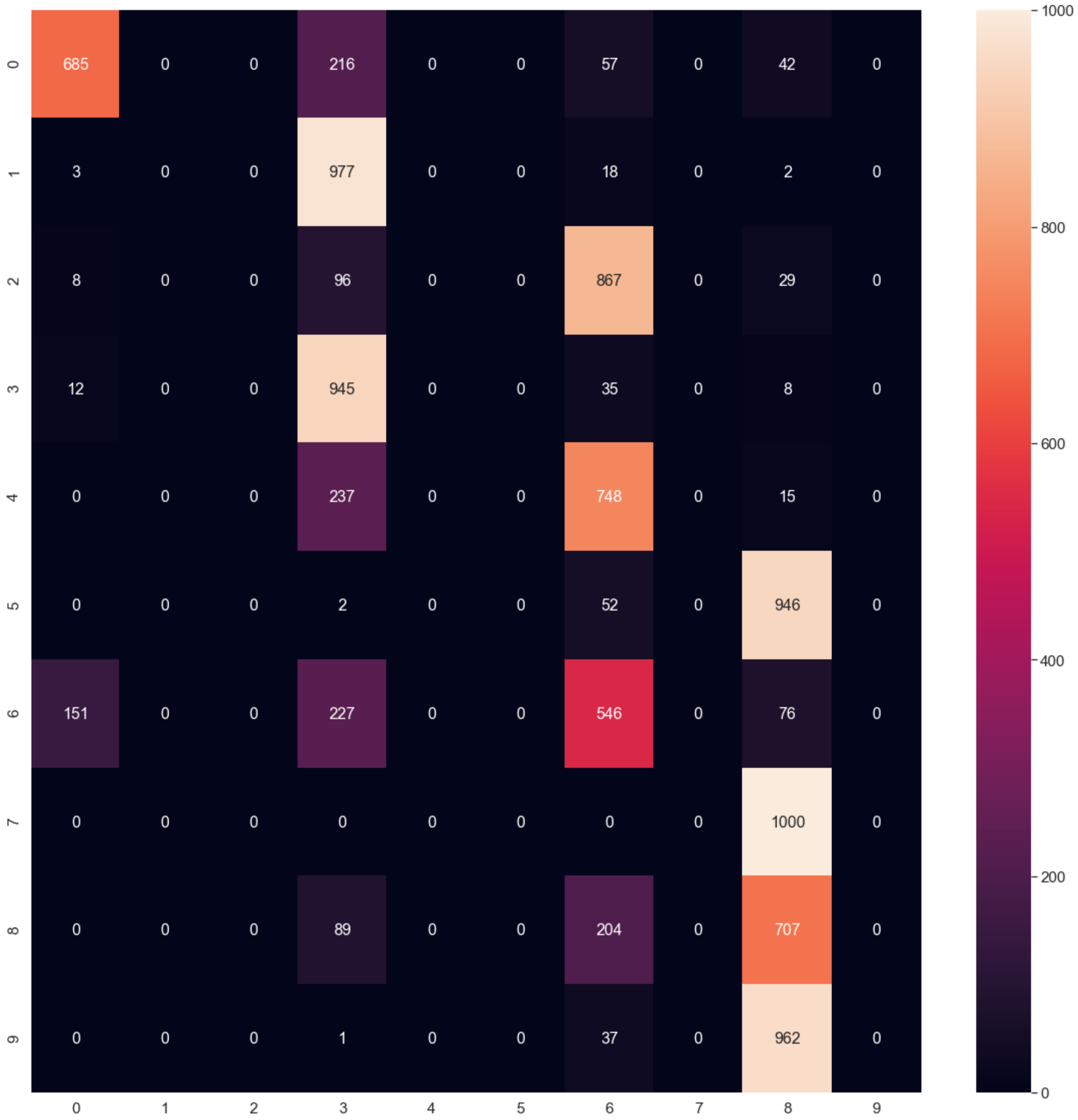
```
<ipython-input-864-c7f80f3173d1>:10: RuntimeWarning: divide by zero encountered
in true_divide
  lamda= np.array([1/((np.mean(classes[i],axis=0))) for i in
range(np.max(train_labels)+1)])  #generate the MLE estimate of parameter lamda
vector for each class
```

```
<ipython-input-864-c7f80f3173d1>:24: RuntimeWarning: divide by zero encountered
in true_divide
  train_imag1=np.array(1/(train_imag1))
```



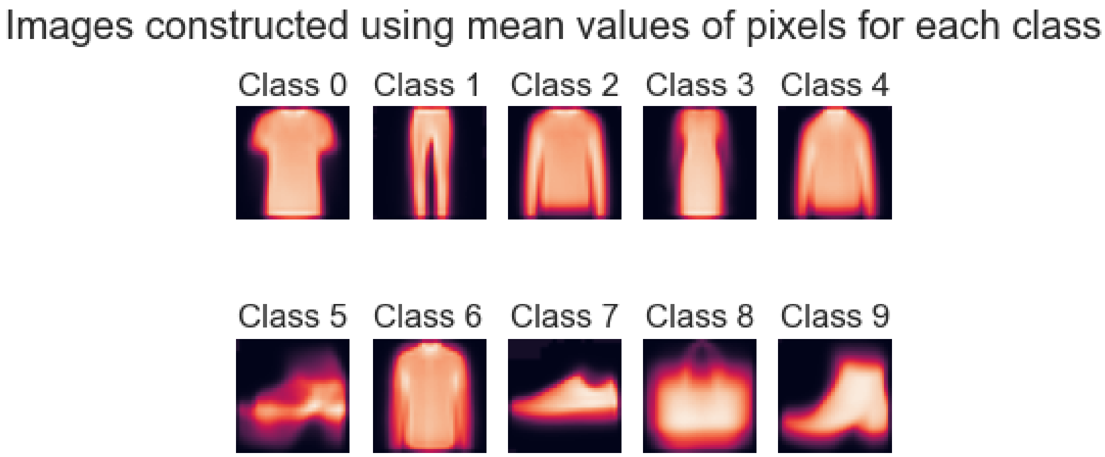
```
Classifier : Bayes Classifier and Class Conditional densities for training Data
is Mulitvariate Exponential Distribution
Prior Probability of class 0 is 0.1
Prior Probability of class 1 is 0.1
Prior Probability of class 2 is 0.1
Prior Probability of class 3 is 0.1
Prior Probability of class 4 is 0.1
Prior Probability of class 5 is 0.1
Prior Probability of class 6 is 0.1
Prior Probability of class 7 is 0.1
Prior Probability of class 8 is 0.1
Prior Probability of class 9 is 0.1
```

```
<ipython-input-865-a702fd3e510d>:12: RuntimeWarning: divide by zero encountered
in log
  pst_prob[j]= np.sum(np.log(lamda_temp))+np.sum(-lamda_temp*dmy)+np.log(prior_p
rob[j])#np.sum(np.log(lamda_temp))-np.sum(lamda*dmy) + np.log(prior_prob[j])
#find the PDF of the feature vector that follows the given distribution
```

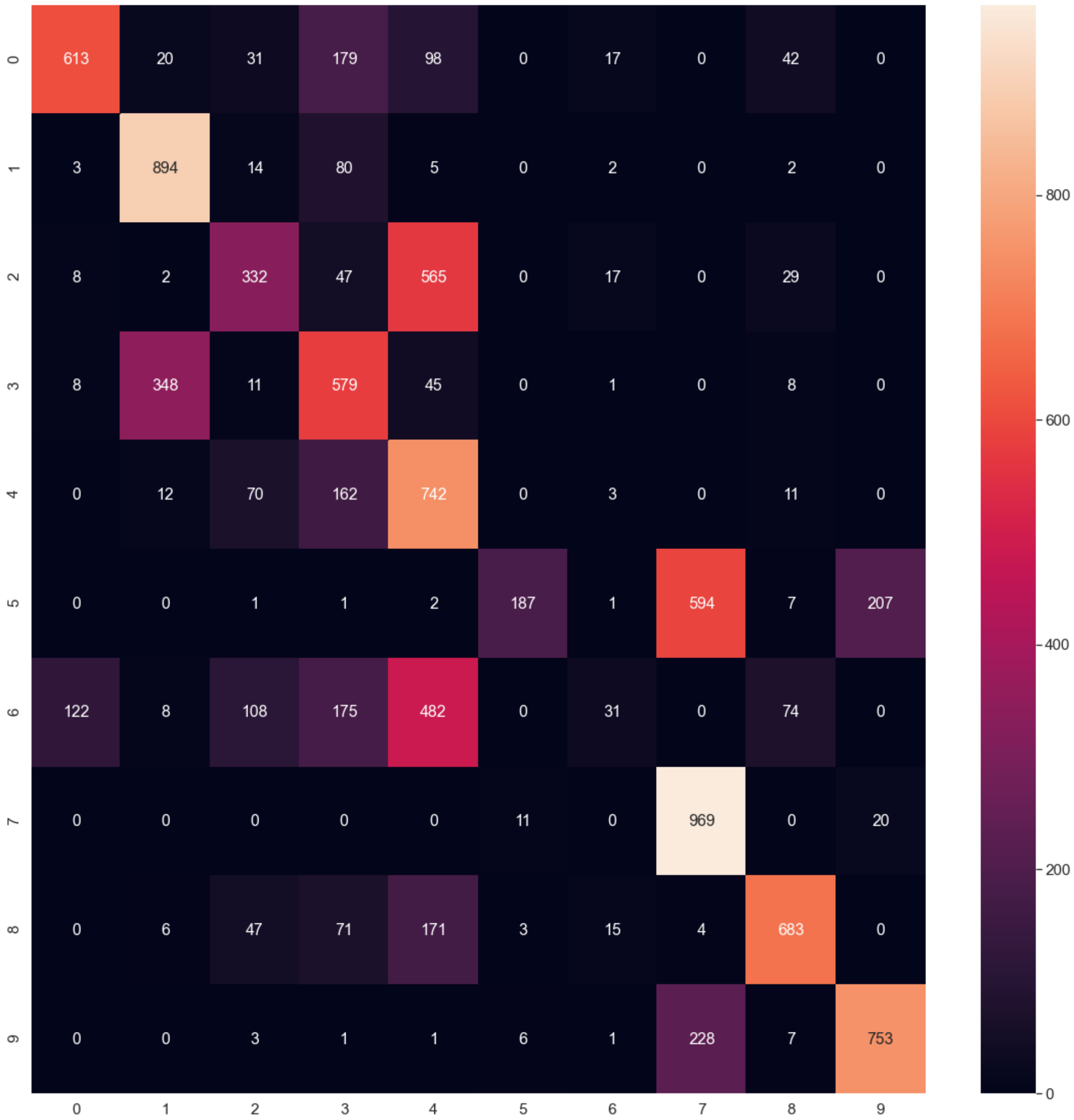


	0	1	2	3	4	5	6	7	8	9
0	685	0	0	216	0	0	57	0	42	0
1	3	0	0	977	0	0	18	0	2	0
2	8	0	0	96	0	0	867	0	29	0
3	12	0	0	945	0	0	35	0	8	0
4	0	0	0	237	0	0	748	0	15	0
5	0	0	0	2	0	0	52	0	946	0
6	151	0	0	227	0	0	546	0	76	0
7	0	0	0	0	0	0	0	0	1000	0
8	0	0	0	89	0	0	204	0	707	0
9	0	0	0	1	0	0	37	0	962	0

```
The % of misclassification is [71.17]% when the noise added to the lambda is 0
```

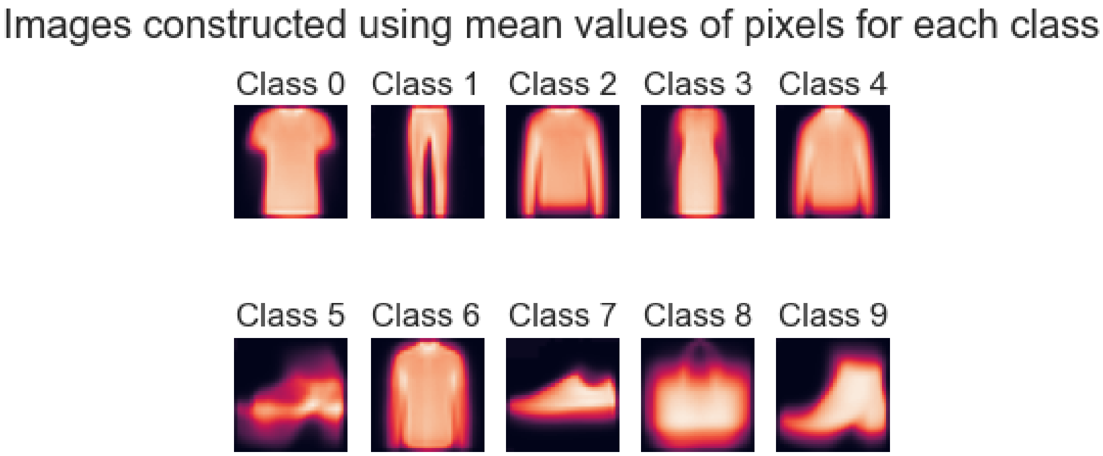


```
Classifier : Bayes Classifier and Class Conditional densities for training Data
is Mulitvariate Exponential Distribution
Prior Probability of class 0 is 0.1
Prior Probability of class 1 is 0.1
Prior Probability of class 2 is 0.1
Prior Probability of class 3 is 0.1
Prior Probability of class 4 is 0.1
Prior Probability of class 5 is 0.1
Prior Probability of class 6 is 0.1
Prior Probability of class 7 is 0.1
Prior Probability of class 8 is 0.1
Prior Probability of class 9 is 0.1
```

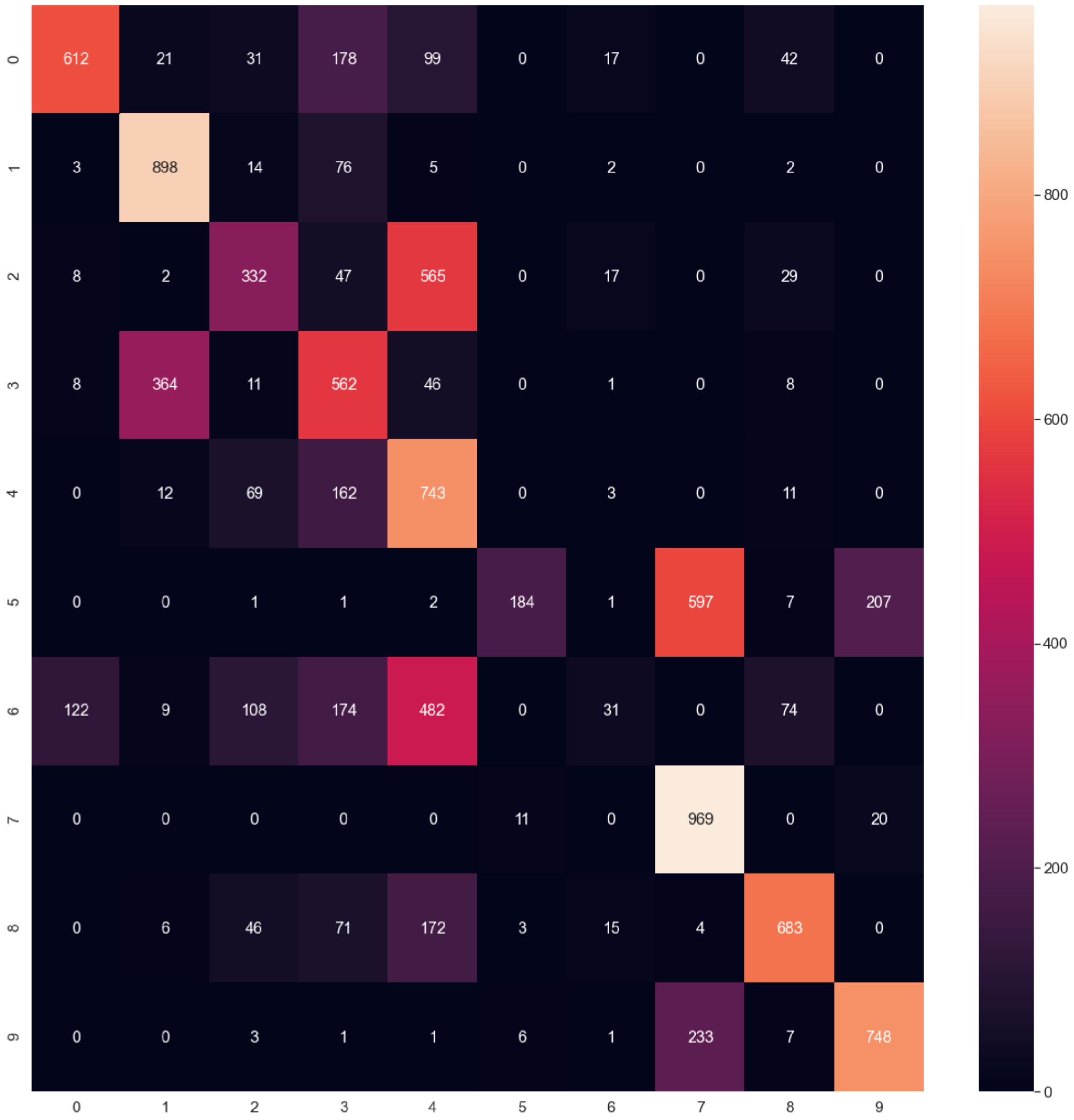


	0	1	2	3	4	5	6	7	8	9
0	613	20	31	179	98	0	17	0	42	0
1	3	894	14	80	5	0	2	0	2	0
2	8	2	332	47	565	0	17	0	29	0
3	8	348	11	579	45	0	1	0	8	0
4	0	12	70	162	742	0	3	0	11	0
5	0	0	1	1	2	187	1	594	7	207
6	122	8	108	175	482	0	31	0	74	0
7	0	0	0	0	0	11	0	969	0	20
8	0	6	47	71	171	3	15	4	683	0
9	0	0	3	1	1	6	1	228	7	753

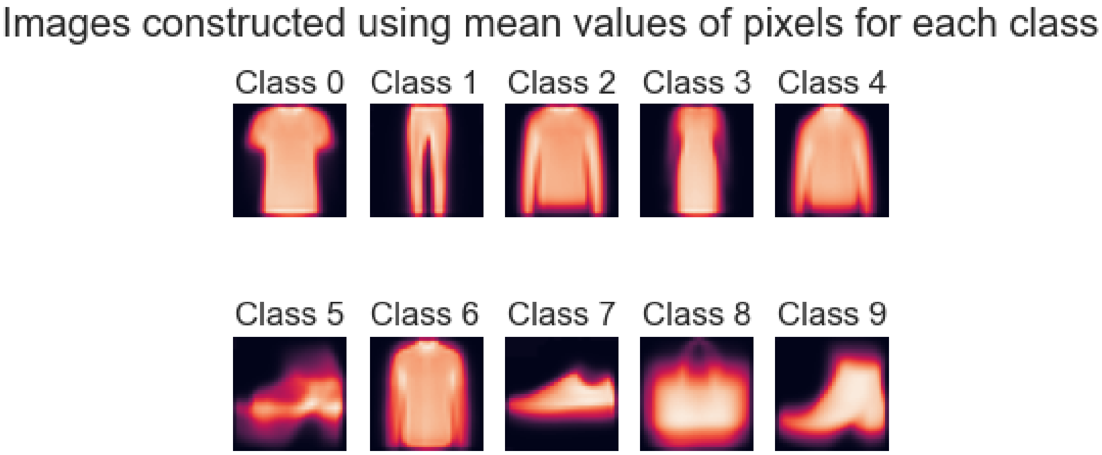
```
The % of misclassification is [42.17]% when the noise added to the lambda is 0.1
```



```
Classifier : Bayes Classifier and Class Conditional densities for training Data
is Mulitvariate Exponential Distribution
Prior Probability of class 0 is 0.1
Prior Probability of class 1 is 0.1
Prior Probability of class 2 is 0.1
Prior Probability of class 3 is 0.1
Prior Probability of class 4 is 0.1
Prior Probability of class 5 is 0.1
Prior Probability of class 6 is 0.1
Prior Probability of class 7 is 0.1
Prior Probability of class 8 is 0.1
Prior Probability of class 9 is 0.1
```



The % of misclassification is [42.38]% when the noise added to the lambda is 0.18

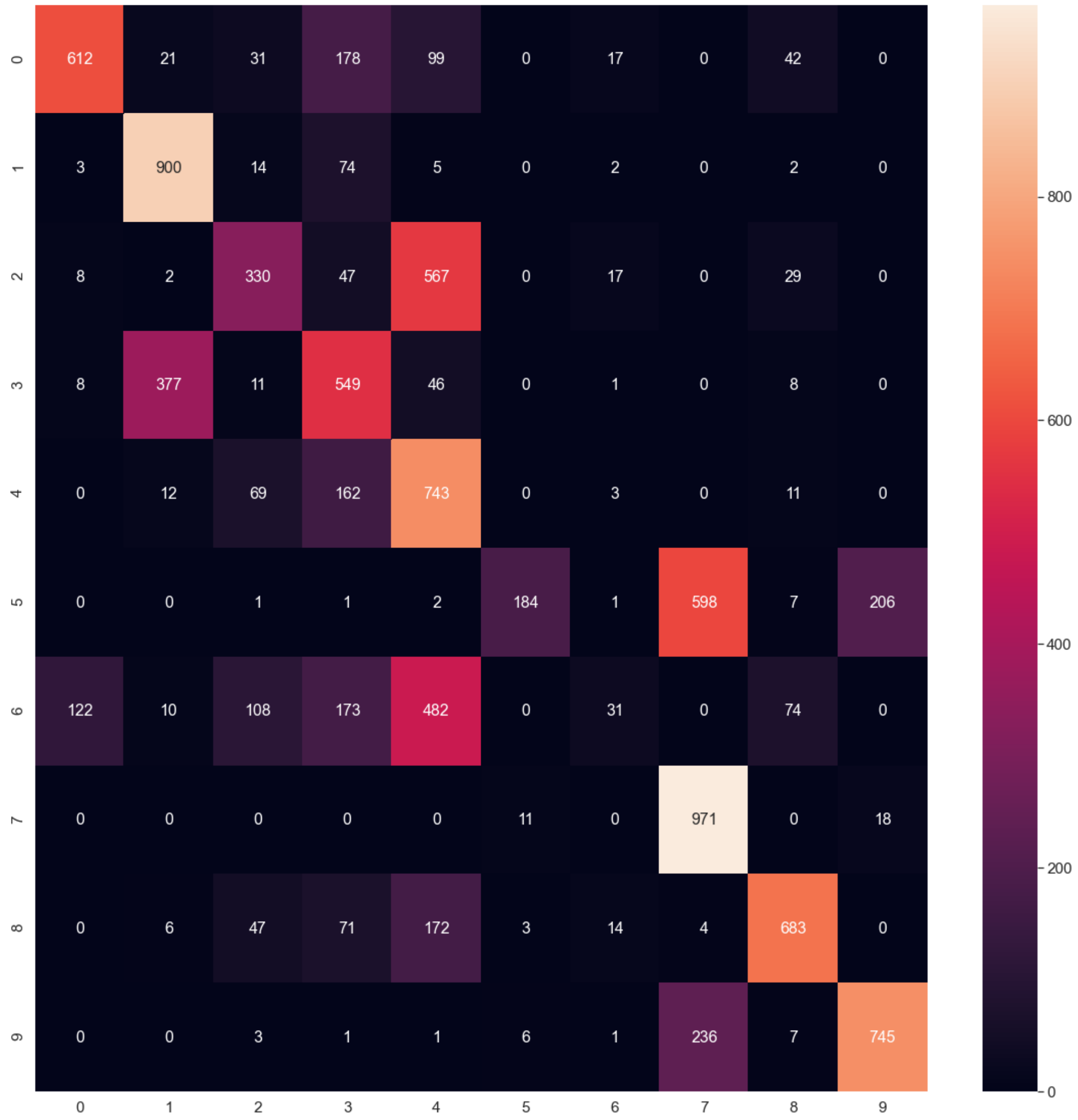


Images constructed using mean values of pixels for each class

Class 0 Class 1 Class 2 Class 3 Class 4

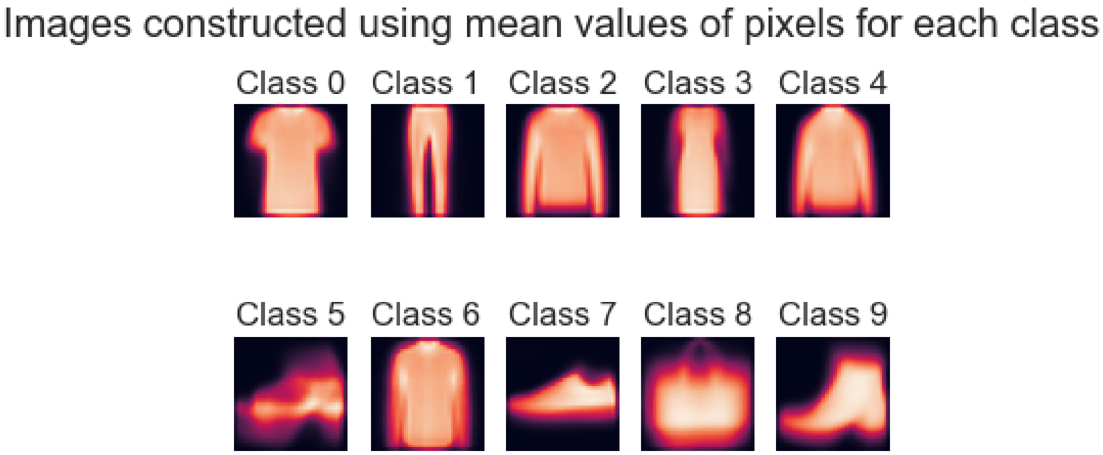
Class 5 Class 6 Class 7 Class 8 Class 9

```
Classifier : Bayes Classifier and Class Conditional densities for training Data
is Mulitvariate Exponential Distribution
Prior Probability of class 0 is 0.1
Prior Probability of class 1 is 0.1
Prior Probability of class 2 is 0.1
Prior Probability of class 3 is 0.1
Prior Probability of class 4 is 0.1
Prior Probability of class 5 is 0.1
Prior Probability of class 6 is 0.1
Prior Probability of class 7 is 0.1
Prior Probability of class 8 is 0.1
Prior Probability of class 9 is 0.1
```



	0	1	2	3	4	5	6	7	8	9
0	612	21	31	178	99	0	17	0	42	0
1	3	900	14	74	5	0	2	0	2	0
2	8	2	330	47	567	0	17	0	29	0
3	8	377	11	549	46	0	1	0	8	0
4	0	12	69	162	743	0	3	0	11	0
5	0	0	1	1	2	184	1	598	7	206
6	122	10	108	173	482	0	31	0	74	0
7	0	0	0	0	0	11	0	971	0	18
8	0	6	47	71	172	3	14	4	683	0
9	0	0	3	1	1	6	1	236	7	745

```
The % of misclassification is [42.52]% when the noise added to the lambda is 0.3
```

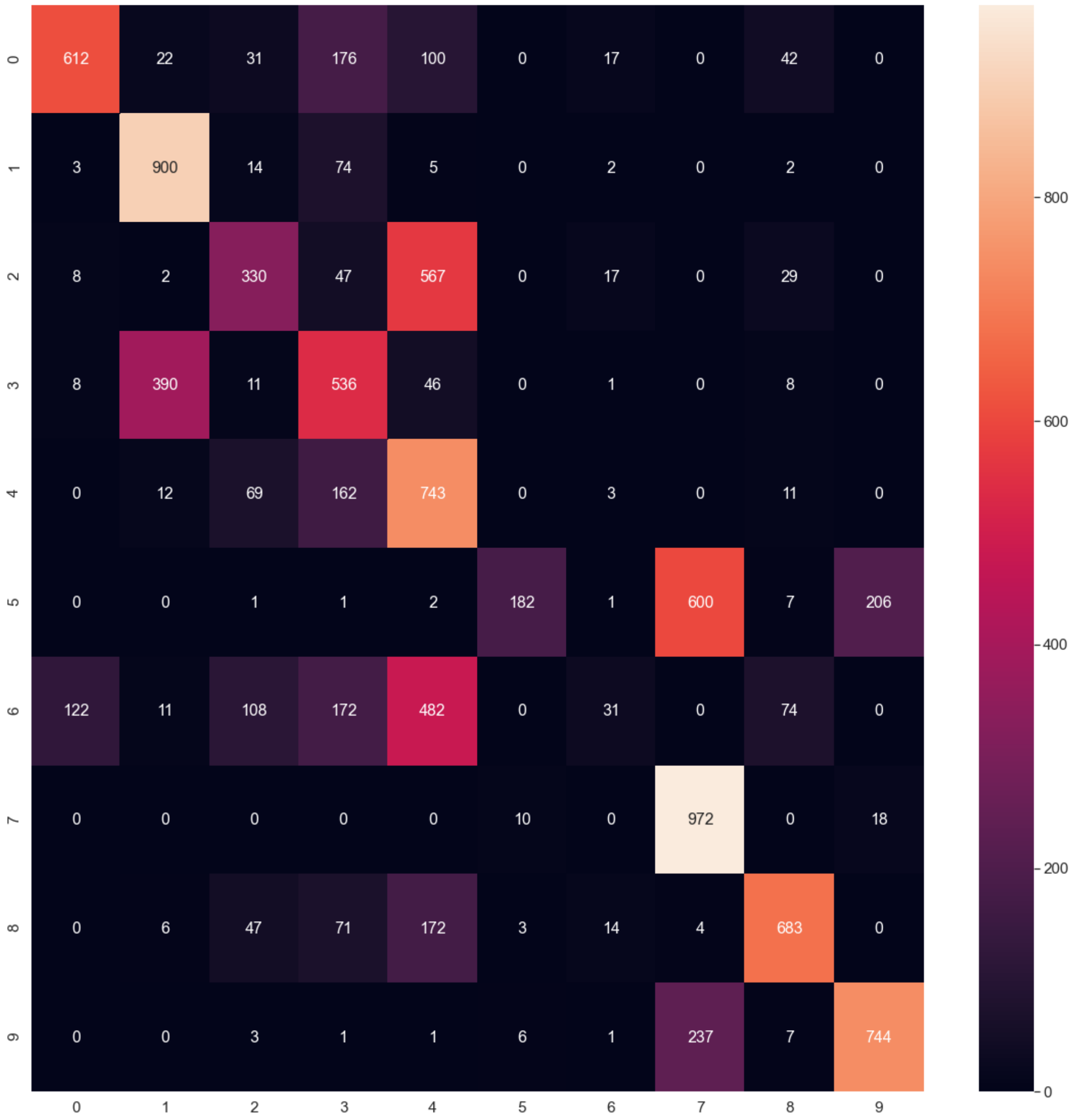


Images constructed using mean values of pixels for each class

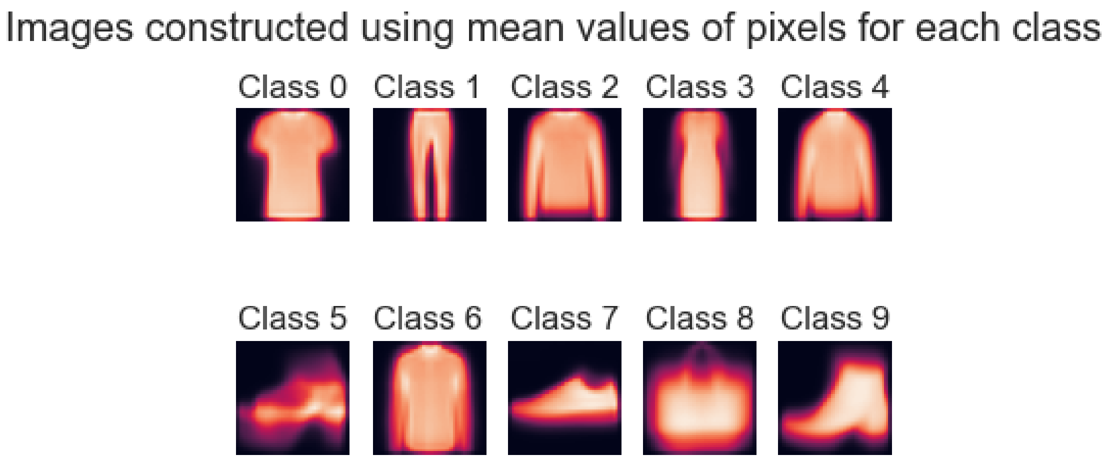
Class 0 Class 1 Class 2 Class 3 Class 4

Class 5 Class 6 Class 7 Class 8 Class 9

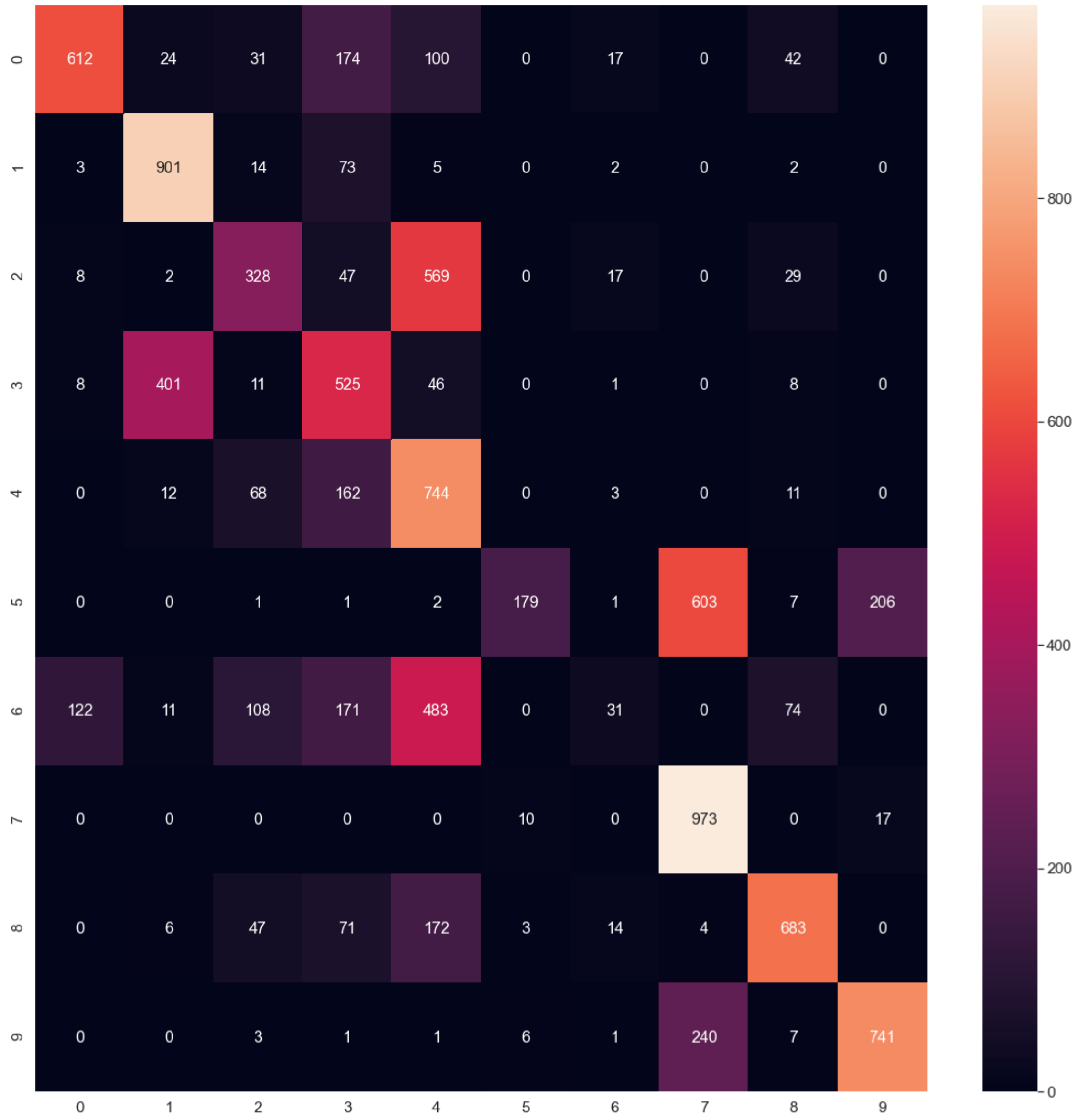
```
Classifier : Bayes Classifier and Class Conditional densities for training Data
is Mulitvariate Exponential Distribution
Prior Probability of class 0 is 0.1
Prior Probability of class 1 is 0.1
Prior Probability of class 2 is 0.1
Prior Probability of class 3 is 0.1
Prior Probability of class 4 is 0.1
Prior Probability of class 5 is 0.1
Prior Probability of class 6 is 0.1
Prior Probability of class 7 is 0.1
Prior Probability of class 8 is 0.1
Prior Probability of class 9 is 0.1
```



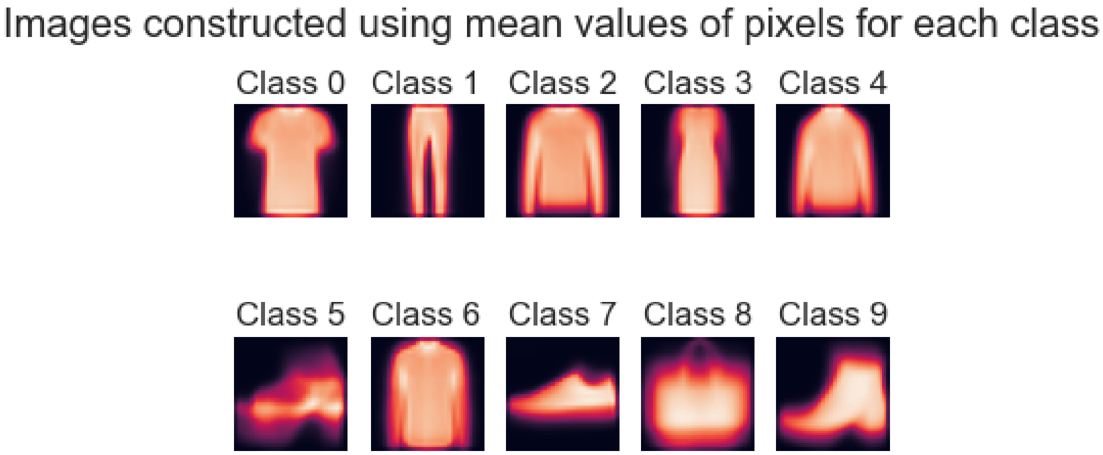
```
The % of misclassification is [42.67]% when the noise added to the lambda is 0.5
```



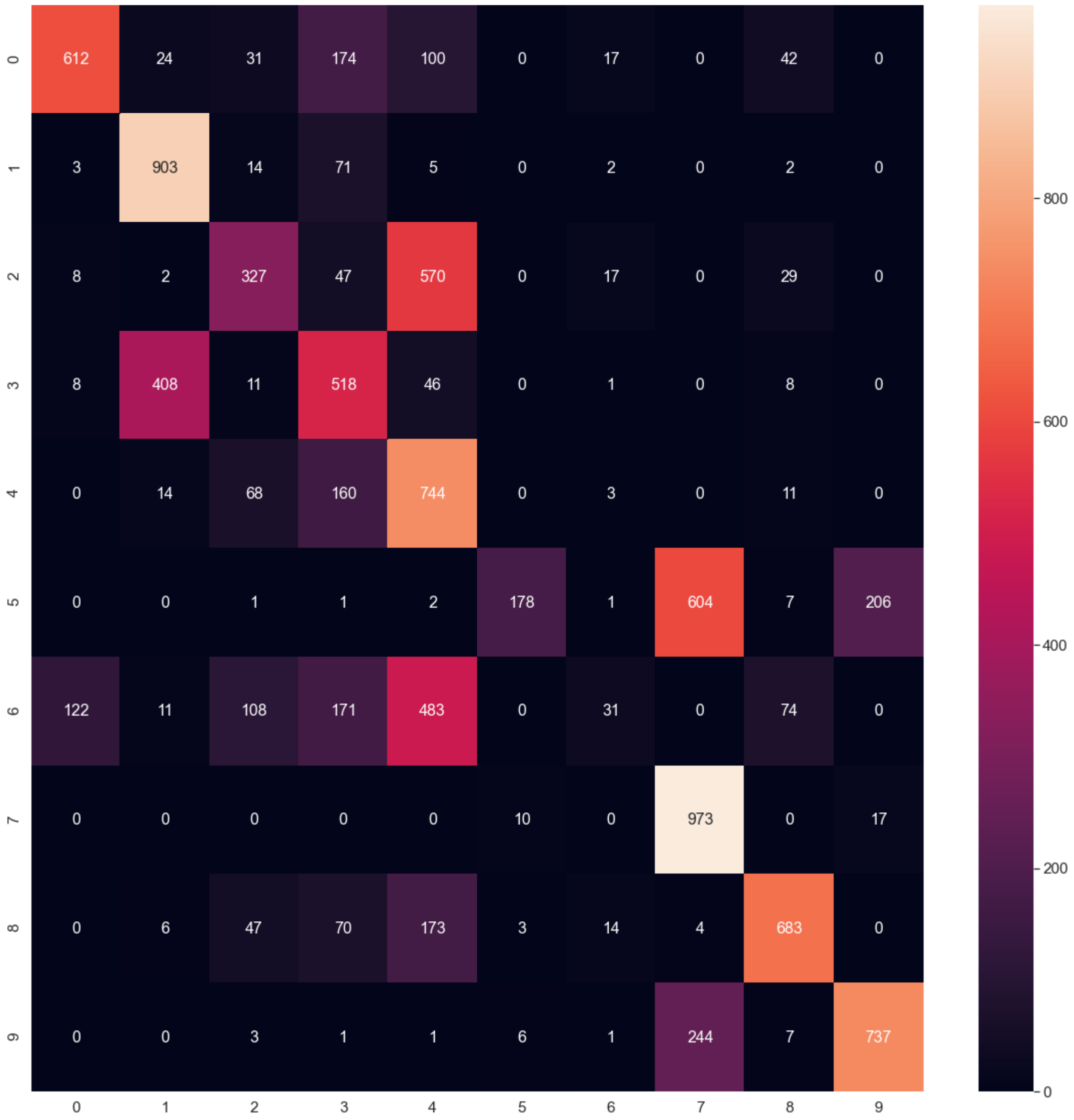
```
Classifier : Bayes Classifier and Class Conditional densities for training Data
is Mulitvariate Exponential Distribution
Prior Probability of class 0 is 0.1
Prior Probability of class 1 is 0.1
Prior Probability of class 2 is 0.1
Prior Probability of class 3 is 0.1
Prior Probability of class 4 is 0.1
Prior Probability of class 5 is 0.1
Prior Probability of class 6 is 0.1
Prior Probability of class 7 is 0.1
Prior Probability of class 8 is 0.1
Prior Probability of class 9 is 0.1
```



```
The % of misclassification is [42.83]% when the noise added to the lambda is 0.8
```



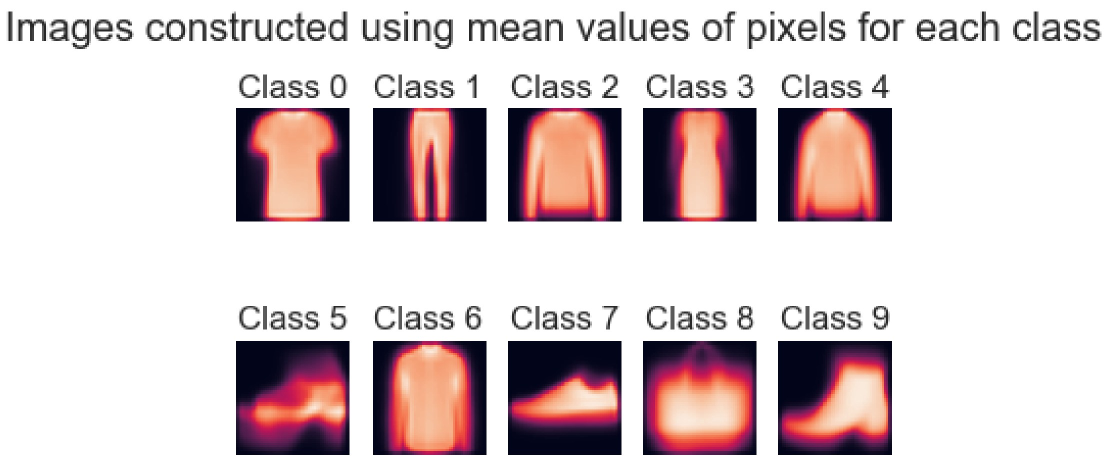
```
Classifier : Bayes Classifier and Class Conditional densities for training Data
is Mulitvariate Exponential Distribution
Prior Probability of class 0 is 0.1
Prior Probability of class 1 is 0.1
Prior Probability of class 2 is 0.1
Prior Probability of class 3 is 0.1
Prior Probability of class 4 is 0.1
Prior Probability of class 5 is 0.1
Prior Probability of class 6 is 0.1
Prior Probability of class 7 is 0.1
Prior Probability of class 8 is 0.1
Prior Probability of class 9 is 0.1
```

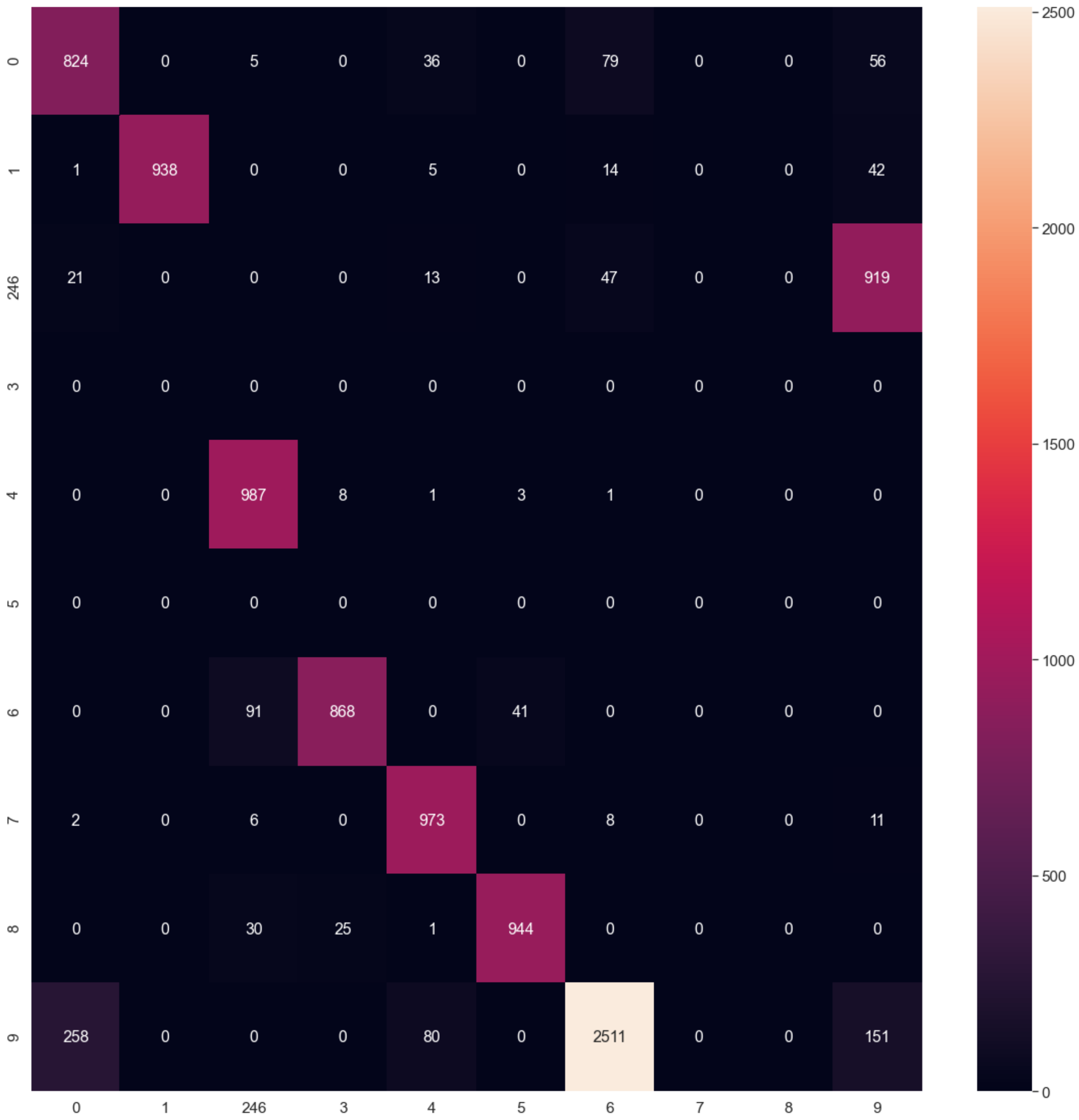


```
The % of misclassification is [42.94]% when the noise added to the lambda is 1
The % of misclassification is less for noise value 0.1 added to the lamda when
lambda goes to infinity
The corresponding % misclassification is 0.4217
```

We combine the classes 0, 2, 4 and 6

[896]:
```
mndata=MNIST(r"MNIST_fashion")  #Extract MNIST data from the specified folder.␣
 ↪mndata has images of numbers 0-9 with their respective labels

combined_class=[2,4,6]
choose_dataset(mndata,combined_class)
```


Images constructed using mean values of pixels for each class
Class 0
Class 1
Class 2
Class 3
Class 4
Class 5
Class 6
Class 7
Class 8
Class 9



	0	1	246	3	4	5	6	7	8	9
0	824	0	5	0	36	0	79	0	0	56
1	1	938	0	0	5	0	14	0	0	42
246	21	0	0	0	13	0	47	0	0	919
3	0	0	0	0	0	0	0	0	0	0
4	0	0	987	8	1	3	1	0	0	0
5	0	0	0	0	0	0	0	0	0	0
6	0	0	91	868	0	41	0	0	0	0
7	2	0	6	0	973	0	8	0	0	11
8	0	0	30	25	1	944	0	0	0	0
9	258	0	0	0	80	0	2511	0	0	151

```
The % of misclassification is 80.86% for MLE estimation assuming the data is
sampled from Multivariate Gaussian Distribution with classes 2,4,6 combined
```

Observations for Problem 1 and 3:

- Modeling the class conditional densities as Multivariate Gaussian distribution results in high accuracy compared to the Multivariate Exponential model.
- For MNIST dataset with multivariate gaussian model class 1 and 3 are misclassified as 8 many times. Hence combining these 3 as a single class reduced our misclassification. Hence increasing the accuracy of the model.
- For MNIST Fashion dataset with multivariate exponential model, class 6 is misclassified as 0 many times. Hence combining these 2 would reduce the misclassification thereby increasing

the accuracy.
- We have added a noise of 0.35 to the diagonal elements of covariance matrix. Adding a value below this resulted in determinant of covariance matrices of some classes to be zero. When inbuit function multivariate_normal.logpdf(x,mean,covariance) is used to estimate the pdf at x, we were able to get an accuarcy of 87% with noise added to diagonal elements being only 0.01.
- When Multivariate Exponential Model is used, when lambda parameter goes to infinity it is replaced by different values between 0 to 1 (noise addition). We got best accuracy for adding noise as 0.18 when it is infinity. But always the accuracy of multivariate exponential model is less than the multivariate gaussian distribution model. We think this is because both MNIST dataset and MNIST fashion dataset are obtained from Multivariate Gaussian Distribution

5 20 NEWS GROUP DATASET

Title : Bayes classifier

Members : Jayanth S, Praveen Kumar N, Rishabh Roy

6 Problem 2:

Read and implement Multinomial Naive Bayes classifier on 20 Newsgroups Dataset. Compare it with sklearn implementation of Multinomial Naive Bayes Classifier.

DATASET SOURCE:http://qwone.com/~jason/20Newsgroups/

Background:

- Dataset consists of 18,886 files(11,334 for Training and 7,552 for Testing) of 20 different categories.
- The data is organized into 20 different newsgroups, each corresponding to a different topic. Some of the newsgroups are very closely related to each other, while others are highly unrelated. Here is a list of the 20 newsgroups.

6.1 Importing and Analysing Dataset:

6.1.1 Importing:

Method 1:

```
[ ]: ################################## Method 1 ####################################

# Importing Dataset directly from sklearn.datasets

from sklearn.datasets import fetch_20newsgroups

train_data = fetch_20newsgroups(subset ="train",shuffle=True) # loading all␣
 ↪Training data(files and labels)
```

```
test_data  = fetch_20newsgroups(subset ="test",shuffle=True)  # loading all␣
 ↪Testing data(files and labels)

# # If we want to read all the data of selected categories and
# # then split into train and test data, we can do it using following code.

# from sklearn.model_selection import train_test_split
# Categories                                    = ["alt.atheism","comp.
 ↪graphics","sci.med"]
# all_data                                      =␣
 ↪fetch_20newsgroups(subset="all",categories=Categories,shuffle=True)
# train_news,test_news,train_labels,test_labels = train_test_split(all_data.
 ↪data,all_data.target,test_size=0.2,random_state=8,stratify=all_data.target)
```

```
Downloading 20news dataset. This may take a few minutes.
Downloading dataset from https://ndownloader.figshare.com/files/5975967 (14 MB)
```

Method 2:

```
[ ]: ################################## Method 2 ####################################

# Importing Dataset from local folder(downloaded from http://qwone.com/~jason/
 ↪20Newsgroups/)

# import sklearn.datasets as sd

# train_data = sd.load_files('drive/MyDrive/Colab Notebooks/
 ↪20news-bydate-train') # loading all Training data(files and labels) from the␣
 ↪Training dataset folder
# test_data  = sd.load_files('drive/MyDrive/Colab Notebooks/20news-bydate-test')␣
 ↪ # loading all Testing  data(files and labels) from the Testing dataset folder

# # Choosing data realted to these categories for Training and Testing
# categories = ["comp.graphics","rec.motorcycles","sci.electronics","misc.
 ↪forsale","talk.politics.misc","alt.atheism"]

# train_data = sd.load_files('drive/MyDrive/Colab Notebooks/
 ↪20news-bydate-train',categories = categories) # loading selected Training␣
 ↪data(files and labels) from the Training dataset folder
# test_data  = sd.load_files('drive/MyDrive/Colab Notebooks/
 ↪20news-bydate-test',categories = categories)  # loading selected Testing ␣
 ↪data(files and labels) from the Testing dataset folder
```

6.1.2 Extracting Training and Testing data from Dataset:

[ ]:

```
# # Extracting files(news) and corresponding labels(0-19) from the Training and␣
 ↪Testing data
# # train_data.target_names gives the different category names in Training data
# # test_data.target_names gives the different category names in Testing data

category_names  = train_data.target_names        # Different category names(20)␣
 ↪of the news data
train_news   = train_data.data
train_labels= train_data.target
test_news    = test_data.data
test_labels = test_data.target
```

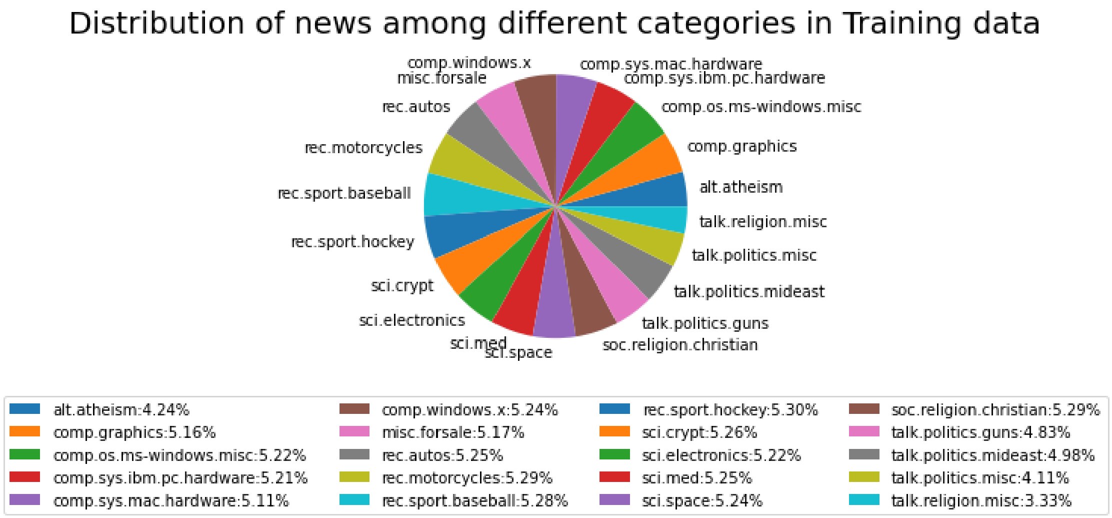
6.1.3 Analysis of the extracted Training and Testing data:

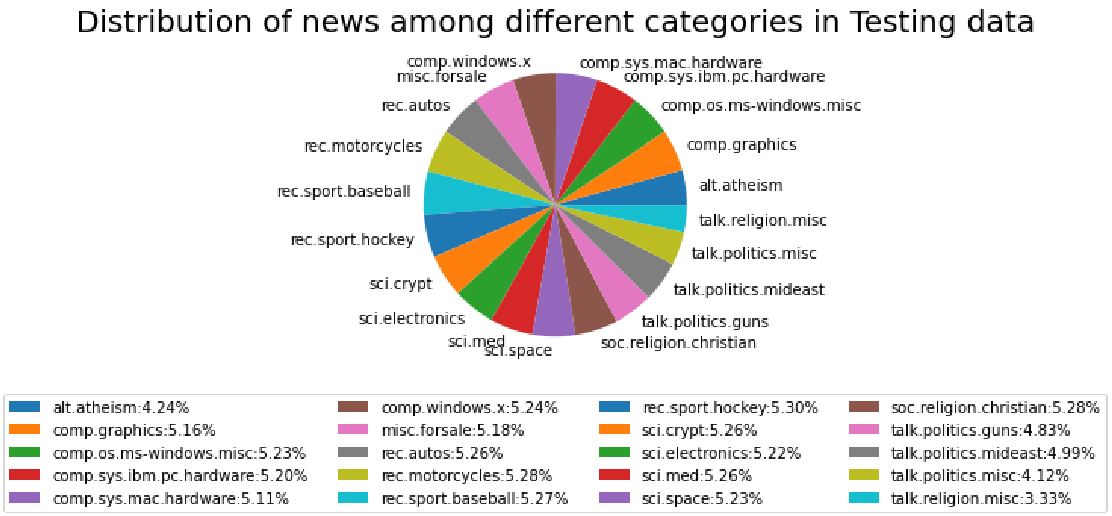
[ ]:

```
import numpy as np
import matplotlib.pyplot as plt

# Finding the number of Training and Testing news(files)
num_train_news = len(train_news)
num_test_news  = len(test_news)

# Finding the probabilities of each category in the Training and Testing data
prob_cat_train = [np.count_nonzero(train_labels == cat)/len(train_labels) for␣
 ↪cat in np.unique(train_labels)]
prob_cat_test  = [np.count_nonzero(test_labels == cat)/len(test_labels) for cat␣
 ↪in np.unique(train_labels)]

# print(f"Different categories of the news data : {category_names}\n")
print(f"Number of news(files) in Training data :{num_train_news}\n")
print(f"Number of news(files) in Testing data  :{num_test_news}\n")

plt.pie([p*100 for p in prob_cat_train], labels = category_names)
plt.title("Distribution of news among different categories in Training␣
 ↪data",fontsize=20)
plt.legend(["{}:{:.2f}%".format(c,p*100) for c,p in␣
 ↪zip(tuple(category_names),tuple(prob_cat_train))],loc='upper␣
 ↪center',bbox_to_anchor=(0.5, -0.05),ncol=4)
plt.show()
plt.pie([p*100 for p in prob_cat_test], labels = category_names)
plt.title("Distribution of news among different categories in Testing␣
 ↪data",fontsize=20)
plt.legend(["{}:{:.2f}%".format(c,p*100) for c,p in␣
 ↪zip(tuple(category_names),tuple(prob_cat_test))],loc='upper␣
 ↪center',bbox_to_anchor=(0.5, -0.05),ncol=4)
```

```
plt.show()
```

```
Number of news(files) in Training data :11314

Number of news(files) in Testing data  :7532
```





6.2 Data Pre-processing:

6.2.1 Feature Extraction:

Method 1:

[ ]:

```
# from sklearn.feature_extraction.text import CountVectorizer

# count_vect = CountVectorizer()   # Creating a object

# # Find different words present in the Training files(news)
# # create a vocabulary of all these words which are the features
# # Associate each word (feature) in the vocabulary a unique number(token)
# count_vect.fit(train_news)       # fit the Training files(news)

# # count the number of times a feature(or a word in vocabulary) occurs for each␣
 ↪feature
# # and for all Training files(news)
# # Now each file is represented by a feature vector with number of occurences␣
 ↪of features as its entry
# counts  =  count_vect.transform(train_news)

# # count_vect.vocabulary :
# # gives the dictionary of all different words(vocabulary) in the Training␣
 ↪files(news) and corresponding tokens(or numbers) associated with that
# # count_vect.get_feature_names() :
# # gives the list of features or words in the vocabulary(here each word in the␣
 ↪vocabulary is a feature)

## ------------------------------------------------------------------------ ##
#  UNCOMMENT THE CODE BELOW TO OBTAIN FUNCTIONALITY OF CODE WRITTEN ABOVE
## ------------------------------------------------------------------------ ##

# # Instead of doing fit and transform separately we can do it at a time

# train_news_worc  =  count_vect.fit_transform(train_news)
# train_news_worc.shape

# from sklearn.feature_extraction.text import TfidfTransformer

# tfidf_trans = TfidfTransformer()    # Creating a object

# train_news_tfidf  =  tfidf_trans.fit_transform(train_news_worc)
# train_news_tfidf.shape

## ------------------------------------------------------------------------ ##
#  RUN THE CELL BELOW TO OBTAIN FUNCTIONALITY OF CODE WRITTEN ABOVE
## ------------------------------------------------------------------------ ##

# Instead of using 'CountVectorizer' and 'TfidfTransformer' separately we can␣
 ↪use
# 'TfidfVectorizer' which will do both the work
```

Method 2(Alternate Method):

```
[ ]: # Extracting the features and representing each files(news) using this feature

     from sklearn.feature_extraction.text import TfidfVectorizer

     # create a tfidfvectorizer object which creates vocabulary(collection of all␣
      ↪different words)
     # and feature vector(with words in vocabulary as the features ) for each␣
      ↪news(file)
     # we can add stop_words = "english" to remove stop words(commonly used words) in␣
      ↪english
     # such as “the”, “a”, “an”, “in” from vocabulary
     tfidf_vect       = TfidfVectorizer(stop_words="english")

     # Representing each file by a feature vector with different words in the
     # Vocabulary as the features and tfidf of the feature(word) as its value
     # Here TfIdf :Term frequency inverse document frequency

     train_news_featmat = tfidf_vect.fit_transform(train_news)   # Training news data␣
      ↪transformed into feature matrix using TFIDF transform
                                                                 #␣
      ↪dim(train_news_featmat) = num_train_news X num_features
     num_features       = train_news_featmat.shape[1]
     print(f"Number of features extracted(Number of words in the vocabulary) :␣
      ↪{num_features}\n")
```

```
Number of features extracted(Number of words in the vocabulary) : 129796
```

6.3 Building Multinomial Naive Bayes classifier:

- Here we have M(11314) number of training data(files) with class labels, distributed among K(20) classes.
- First we estimate prior probabilities(class probabilities) $P(C_k)$ for all $k = 1, \dots, K$ from M training examples.
- Then we build a vocabulary(set of all distinct words in training data). here each word in vocabulary is a feature(there are n features).
- Next we transform each training data(file) into a feature vector $X = [x_1, \dots x_n]$.
- Now using training data we estimate $P(X/C_k) = P(x_1/C_k) \times \cdots \times P(x_n/C_k)$(since in Naive bayes classifier we assume features are independent) for all $k = 1, \dots, K$.
- Once we have estimated the likelihood of all classes, given a new feature vector X we compute the posterior probability \$P(C_k/X) = P(X/C_k)×$P(C\_k)/P(X) \propto P(X/C\_k) \times P(C\_k)$\$$forallk = 1, \dots, K$.
- Finally we use MAP rule to classify the given new feature vector to the class C_{NB} which maximizes the posterior probability i.e, $C_{NB} = argmax_{C_k} P(X/C_k) \times P(C_k)$.

6.3.1 Defining function for training the classifier and for predicting:

```
[ ]: def my_MultinomialNB_train(data,labels,alpha):

       print(f"Training with {data.shape[0]} Training data...............\n")

       # finding number of features (n) in the feature vector X
       n = data.shape[1]

       # # finding different classes(categories)
       categories = np.unique(labels)

       # finding probabilities of different categories(prior probabilities)
       # P(C_k) = prob_cat[k] ==> probability of category(class) k, estimated from␣
     ↪the Training dataset
       est_prob_cat = [np.count_nonzero(labels == cat)/len(labels) for cat in␣
     ↪categories]

       # (np.count_nonzero(labels == cat) + alpha)/(len(labels)+alpha*len(categories))
       # P_ki is the probabilty of occurence of features x_i's given class C_k
       # x_i is the number of times feature i occurs

       P_ki_lst = []                                                  # list of␣
     ↪probabilities of occurence of features x_i's given a class                    ␣
     ↪
       for category in categories:

         num_cat_data = np.count_nonzero(labels == category)
         print(f"--> Training with {num_cat_data} category {category} data.....")

         X_cat = np.array([data[row,:].toarray()[0] for row in np.where(labels ==␣
     ↪category)[0]]) # collection of feature vectors belonging to category =␣
     ↪"category"
         X_cat.reshape((num_cat_data,n))                                            ␣
     ↪          # dim(X_cat) = num_cat_data X n

         # appending probabilities of occurence of features x_i's, given class =␣
     ↪"category"
         # P_ki = (number of files feature(word) x_i present, in category C_k +␣
     ↪alpha)/(total number of files in category C_k + alpha * number of features)  ␣
     ↪
         # Here we are use laplace smoothing(with parameter "alpha") while␣
     ↪calculating  P_ki,
         # since there may be no occurence of a feature x_i in the given category
         # which will give probability of occurence of that particular feature x_i = 0
         # this will make P(X/C_k) = 0 , which is not desirable.
```

```
    P_ki_lst.append([(np.count_nonzero(X_cat[:,i]!=0) +alpha)/
 ↪(num_cat_data+alpha*n)  for i in range(n)])

  print("\nTraining Done...........")
  return est_prob_cat,P_ki_lst,categories


def my_MultinomialNB_predict(data,est_prob_cat,P_ki_lst,categories):

  # Here given the Testing dataset we need to find the posterior probability
  # that a given feature vector X belongs to some class C_k: P(C_k/X)
  # then classify the feature vector X to calss C_k for which P(C_k/X) is maximum
  # P(C_k/X) = P(X/C_k)*P(C_k)/P(X)

  # To find P(C_k/X) we need to find likelihood of feature vector X given class␣
 ↪C_k: P(X/C_k)
  # here we model P(X/C_k) as Multinomial distribution
  # P(X/C_k) = prod[(P_ki)^x_i] , where P_ki = P(x_i/C_k)

  from scipy.sparse import find

  print(f"Testing dataset size: {data.shape} ...............\n")
  print(f"Testing with {data.shape[0]} Testing data...............\n")

  predicted_labels = []                          # list for predicted labels
  for i in range(data.shape[0]):

    X_nonzero_lst = list(find(data[i,:]))        # finding nonzero element␣
 ↪indices(i,nz_col) and corresponding nonzero values(list) from i-th feature␣
 ↪vector
    X_values      = X_nonzero_lst.pop()          # extracting nonzero values
    X             = []                           # list containing position and␣
 ↪value of nonzero feature in selected i-th feature vector
    for cind in range(len(X_nonzero_lst[0])) :
      X.append((X_nonzero_lst[1][cind],X_values[cind]))


    # Now find posterior probability for all the categories given this feature␣
 ↪vector

    post_prob_lst = []
    for category in categories:

      P_ki = P_ki_lst[category]                  # select the probabilty vector␣
 ↪for selected category = "category"
```

```
      # Computing P(X/C_k)
      likelihood_of_category = np.prod([P_ki[ind]**x_i for ind,x_i in X])

      # Computing P(C_k/X) = P(X/C_k)*P(C_k) and appending to post_prob_lst
      post_prob              = likelihood_of_category * est_prob_cat[category]
      post_prob_lst.append(post_prob)

    # finding the category for which posterior probability is maximum
    # and appending that category as predicted category(label)
    predicted_labels.append(np.argmax(post_prob_lst))

  print("\nTesting Done...........")
  return predicted_labels
```

6.3.2 Training the classifier:

[ ]:

```
from sklearn.feature_extraction.text import CountVectorizer

tfidf_vect_m    = TfidfVectorizer(stop_words="english")
train_news_worc = tfidf_vect_m.fit_transform(train_news)
# count_vect        = CountVectorizer(stop_words="english")   # create a count␣
 ↪vectorizer object(here stop_words is removed)
# train_news_worc   = count_vect.fit_transform(train_news)    # converting each␣
 ↪news(files) in Training data into feature vector X
```

[ ]:

```
alpha = 0.05
est_prob_cat,P_ki_lst,categories =␣
 ↪my_MultinomialNB_train(train_news_worc,train_labels,alpha)
```

```
Training with 11314 Training data...

--> Training with 480 category 0 data...
--> Training with 584 category 1 data...
--> Training with 591 category 2 data...
--> Training with 590 category 3 data...
--> Training with 578 category 4 data...
--> Training with 593 category 5 data...
--> Training with 585 category 6 data...
--> Training with 594 category 7 data...
--> Training with 598 category 8 data...
--> Training with 597 category 9 data...
--> Training with 600 category 10 data...
--> Training with 595 category 11 data...
--> Training with 591 category 12 data...
--> Training with 594 category 13 data...
--> Training with 593 category 14 data...
--> Training with 599 category 15 data...
--> Training with 546 category 16 data...
--> Training with 564 category 17 data...
--> Training with 465 category 18 data...
--> Training with 377 category 19 data...

Training Done...
```

6.3.3 Testing the classifier:

[ ]:

```
#test_news_worc  = count_vect.transform(test_news)    # converting each news(files) in Testing data into feature vector X
test_news_worc  = tfidf_vect_m.transform(test_news)
pred_labels     = my_MultinomialNB_predict(test_news_worc,est_prob_cat,P_ki_lst,categories)
```

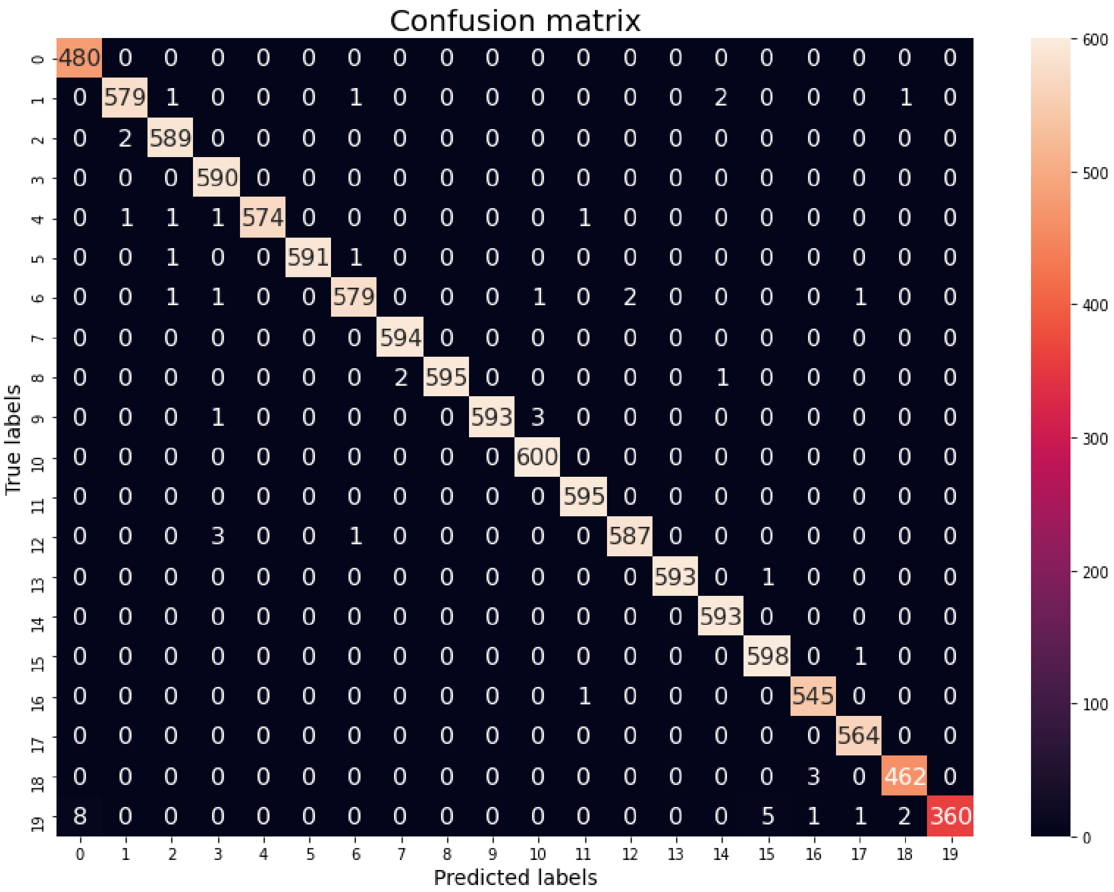
```
Testing dataset size: (7532, 129796) ...

Testing with 7532 Testing data...


Testing Done...
```

6.3.4 Analysis of results:

[ ]:

```
from sklearn.metrics import classification_report,accuracy_score,confusion_matrix
import seaborn as sns

Accuracy = accuracy_score(test_labels,pred_labels)*100
print(f"Accuracy of the Multinomial NB classifier with alpha = {alpha}: {Accuracy} %\n ")

# report   = classification_report(test_labels,pred_labels,target_names=test_data.target_names)
# print("-"*24+" Classification report "+"-"*24+"\n")
# print(report)

conf_mat = confusion_matrix(test_labels,pred_labels)
fig= plt.figure(figsize=(14,10))
sns.heatmap(conf_mat,annot=True,annot_kws={"size":16},fmt="d")
plt.xlabel("Predicted labels",fontsize=14)
plt.ylabel("True labels",fontsize=14)
plt.title("Confusion matrix",fontsize=20)
plt.show()
```

```
Accuracy of the Multinomial NB classifier with alpha = 0.05: 99.53155382711685 %
```



6.4 Using in-built Multinomial Naive Bayes classifier:

6.4.1 Training the classifier:

[ ]:
```
# Import Multinomial NB classifier package from sklearn

from sklearn.naive_bayes import MultinomialNB

alpha = 0.05                                # Additive (Laplace) smoothing␣
 ↪parameter (0 for no smoothing).
clsfr = MultinomialNB(alpha=alpha)          # create classifier object
print("Training...............\n")
clsfr.fit(train_news_featmat, train_labels)  # fit the classifier for Training␣
 ↪data
```

```
Training...
```

[ ]:
```
MultinomialNB(alpha=0.05, class_prior=None, fit_prior=True)
```

6.4.2 Testing the classifier:

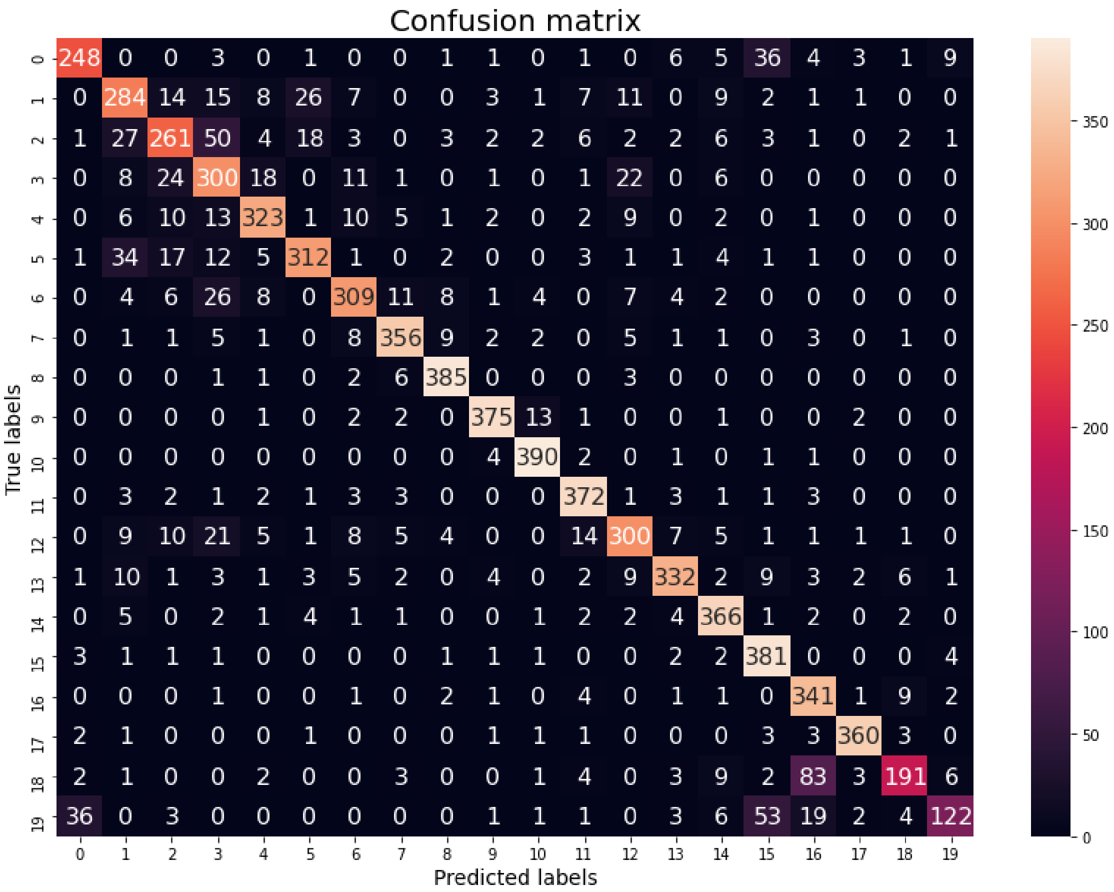
```
[ ]: print("Testing...............\n")

     test_news_featmat  = tfidf_vect.transform(test_news)   # Testing news data␣
      ↪transformed into feature matrix using TFIDF transform
     pred_labels        = clsfr.predict(test_news_featmat)
```

```
Testing...
```

6.4.3 Analysis of results:

```
[ ]: from sklearn.metrics import classification_report,accuracy_score,confusion_matrix
     import seaborn as sns

     Accuracy = accuracy_score(test_labels,pred_labels)*100
     print(f"Accuracy of the Multinomial NB classifier with alpha = {alpha}:␣
      ↪{Accuracy} %\n ")

     # report   =␣
      ↪classification_report(test_labels,pred_labels,target_names=test_data.
      ↪target_names)
     # print("-"*24+" Classification report "+"-"*24+"\n")
     # print(report)

     conf_mat = confusion_matrix(test_labels,pred_labels)
     fig= plt.figure(figsize=(14,10))
     sns.heatmap(conf_mat,annot=True,annot_kws={"size":16},fmt="d")
     plt.xlabel("Predicted labels",fontsize=14)
     plt.ylabel("True labels",fontsize=14)
     plt.title("Confusion matrix",fontsize=20)
     plt.show()
```

```
Accuracy of the Multinomial NB classifier with alpha = 0.05: 83.74933616569305 %
```



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	248	0	0	3	0	1	0	0	1	1	0	1	0	6	5	36	4	3	1	9
1	0	284	14	15	8	26	7	0	0	3	1	7	11	0	9	2	1	1	0	0
2	1	27	261	50	4	18	3	0	3	2	2	6	2	2	6	3	1	0	2	1
3	0	8	24	300	18	0	11	1	0	1	0	1	22	0	6	0	0	0	0	0
4	0	6	10	13	323	1	10	5	1	2	0	2	9	0	2	0	1	0	0	0
5	1	34	17	12	5	312	1	0	2	0	0	3	1	1	4	1	1	0	0	0
6	0	4	6	26	8	0	309	11	8	1	4	0	7	4	2	0	0	0	0	0
7	0	1	1	5	1	0	8	356	9	2	2	0	5	1	1	0	3	0	1	0
8	0	0	0	1	1	0	2	6	385	0	0	0	3	0	0	0	0	0	0	0
9	0	0	0	0	1	0	2	2	0	375	13	1	0	0	1	0	0	2	0	0
10	0	0	0	0	0	0	0	0	0	4	390	2	0	1	0	1	1	0	0	0
11	0	3	2	1	2	1	3	3	0	0	0	372	1	3	1	1	3	0	0	0
12	0	9	10	21	5	1	8	5	4	0	0	14	300	7	5	1	1	1	1	0
13	1	10	1	3	1	3	5	2	0	4	0	2	9	332	2	9	3	2	6	1
14	0	5	0	2	1	4	1	1	0	0	1	2	2	4	366	1	2	0	2	0
15	3	1	1	1	0	0	0	0	1	1	1	0	0	2	2	381	0	0	0	4
16	0	0	0	1	0	0	1	0	2	1	0	4	0	1	1	0	341	1	9	2
17	2	1	0	0	0	1	0	0	0	1	1	1	0	0	0	3	3	360	3	0
18	2	1	0	0	2	0	0	3	0	0	1	4	0	3	9	2	83	3	191	6
19	36	0	3	0	0	0	0	0	0	1	1	1	0	3	6	53	19	2	4	122

6.5 Observations:

- Accuracy of the In built Multinomial Naive Bayes classifier was around 84%.
- Lower values of alpha(smoothing parameter) gave better accuracy.
- Accuracy of the Multinomial Naive Bayes classifier bulit from scratch was around 81%