CSE-555 Pattern Recognition

Name:Rajiv Ranjan

Ubid:50249099

Ubit:rajivran

[Problem Set 1](https://ublearns.buffalo.edu/webapps/assignment/uploadAssignment?content_id=_4486082_1&course_id=_154023_1&group_id=&mode=view)

**Part 1**

For Part 1 of this assignment we were given the images as 28 x 28 pixels in gray-scale. The categories are 0, 1, ... 9. We concatenate the image rows into a 28 x 28 vector and treat this as our feature, and assume the feature vectors in each category in the training data "train-images-idx3-ubyte.gz") have Gaussian distribution.

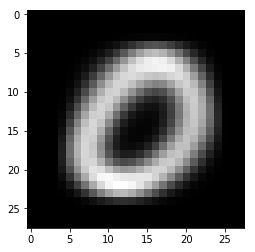
We had to draw the mean and standard deviation of those features for the 10 categories as 28 x 28 images using the training images ("train-images-idx3-ubyte.gz"). There should be 2 images for each of the 10 digits, one for mean and one for standard deviation. We call those "mean digits" and "standard deviation digits" in CSE455/555.

I have used python for my implementation.

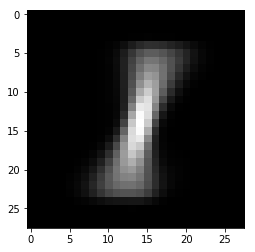
Below are the images:

**Mean of all the dataset’s different digits:**

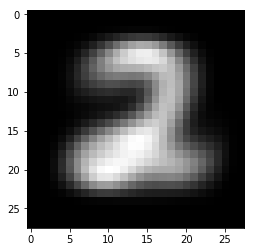
**Mean for Digit 0:**



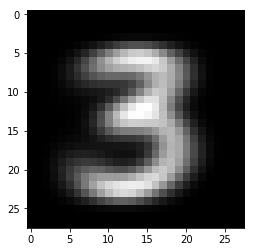
**Mean for Digit 1:**

****

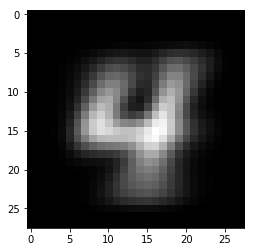
**Mean for Digit 2:**



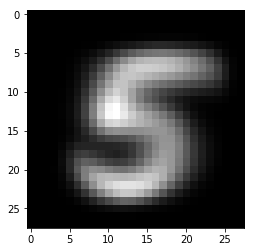
**Mean for Digit 3:**

****

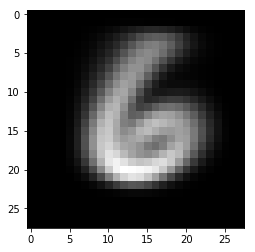
**Mean for Digit 4:**



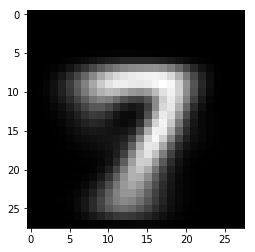
**Mean for Digit 5:**

****

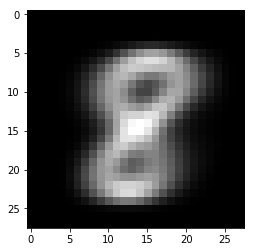
**Mean for Digit 6:**

****

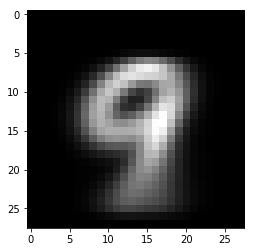
**Mean for Digit 7:**

****

**Mean for Digit 8:**



**Mean for Digit 9:**

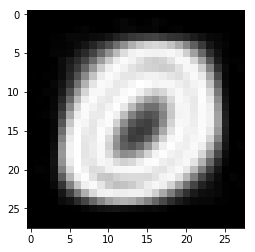


A montage of all the Mean digits:

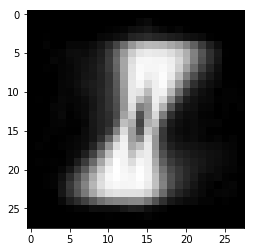


**Standard Deviation of all the dataset’s different images:**

**Standard Deviation for Digit 0:**

****

**Standard Deviation for Digit 1:**

****

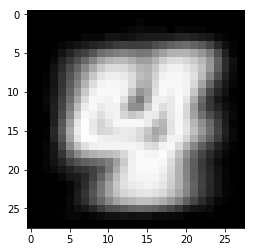
**Standard Deviation for Digit 2:**

****

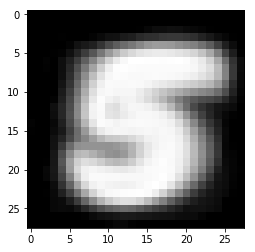
**Standard Deviation for Digit 3:**

****

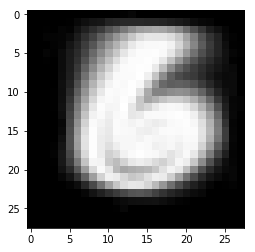
**Standard Deviation for Digit 4:**

****

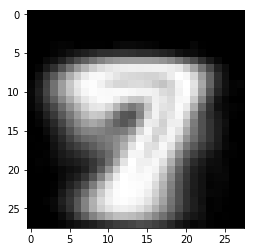
**Standard Deviation for Digit 5:**



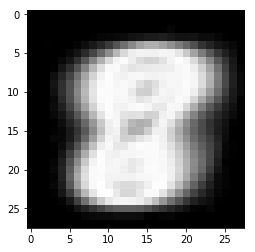
**Standard Deviation for Digit 6:**

****

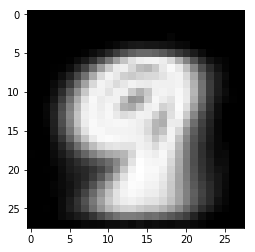
**Standard Deviation for Digit 7:**

****

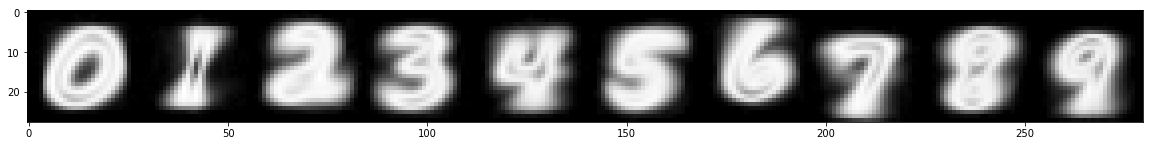
**Standard Deviation for Digit 8:**

****

**Standard Deviation for Digit 9:**

****

A montage of all the Standard Deviation digits:



Implementation Details:

MNIST dataset is available in tensorflow library. For importing the MNIST dataset following code was used:

#using it only for loading the MNIST dataset

import tensorflow as tf

#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)

#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)

This loads the data in *train\_data* variable. *train\_labels* contains the label for this training data.

Now we have stored the test data in *eval\_data* variable. The corresponding labels are in *eval\_labels* variable.

**Calculating Mean of each digits:**

For getting the mean of all the digits say for digit 0 we have to store all the data with digit value as 0 in a separate variable. We have named this variable *X0.* Similarly, for digits 1 we have X1 and so on till X9 for digit 9.

Then after this for each digit the mean value it is stored in m0 for digit 0 ,m1 stores mean of all the 1 and so on.

This has been displayed using *imshow* function.

e.g for digit 0:

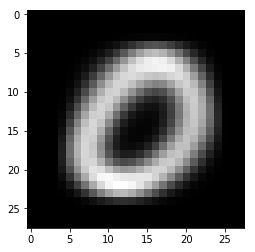
#Mean for Zero

X0=train\_data[train\_labels[:,0]==1]

m0 = np.mean(X0,axis=0)

plt.imshow(m0.reshape(28,28),cmap='gray')

plt.show()



A montage has been created using the below codes:

#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()



**Calculating Standard Deviation of each digits:**

For getting the standard deviation of all the digits say for digit 0 we have to store all the data with digit value as 0 in a separate variable. We have named this variable *X00.* Similarly, for digits 1 we have X11 and so on till X99 for digit 9.

Then after this for each digit the standard deviation value it is stored in m00 for digit 0 ,m11 stores mean of all the 1 and so on.

This has been displayed using *imshow* function.

For e.g. for digit 0:

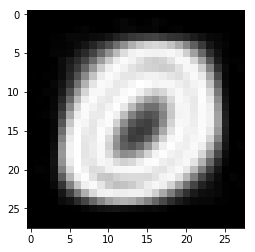
#Standard Deviation for Zero

X00=train\_data[train\_labels[:,0]==1]

m00 = np.std(X00,axis=0)

plt.imshow(m00.reshape(28,28),cmap='gray')

plt.show()

****

A montage has been created using the below codes:

#Drawing a montage of all the standard deviation digit images

plot\_image = np.concatenate((m00.reshape(28,28),m11.reshape(28,28),m22.reshape(28,28),

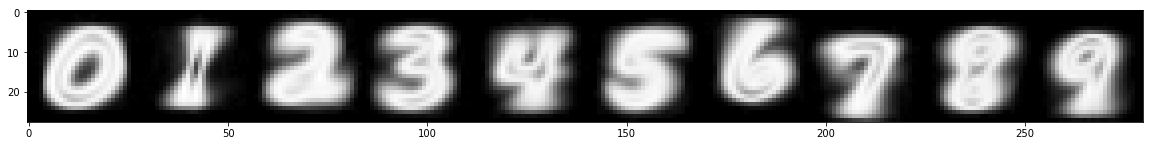
m33.reshape(28,28),m44.reshape(28,28),m55.reshape(28,28),m66.reshape(28,28),

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');

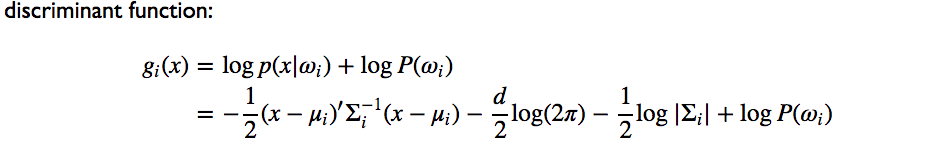
plt.show()



Part 2:

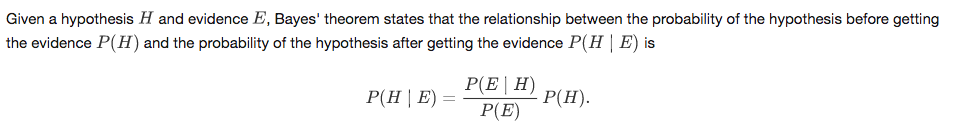
For the second part of the implementation we were supposed to classify the images in the testing data set ("t10k-images-idx3-ubyte.gz") using 0-1 loss function and Bayesian decision rule and report the performance.

For this we had to use the discriminant function:



Using this discriminant function, we can basically calculate the decision boundaries of the different classes 0,1……9 and based on that we estimate how many of the test dataset was correctly categorized in the correct classes.

Bayes theorem for binary class can be defined as follows, where H and E are two classes:



Here P(H|E) is called posterior and P(E|H) is called the class conditional distribution. P(H) is called prior. Also P(E) is called marginal.

So the problem with bayes theorem is that it depends on the class conditional and we don’t what the class conditional is.

Let’s call class E as class 0 and class H as 1.

So we will try to estimate the class conditional here.

Let’s assume that the class conditional distribution is Gaussian and we will try to find the parameters of it And we have assumed that there can be only 2 classes 0 and 1.

So here Y has the 2 values 0 and 1 and X has the set of all the training images which we will be using for estimating the parameters of the class conditional distribution.

So in bayes theorem=

P(Y=k|X=x0)= P(X=x0|Y=k)\*P(Y=k)/P(X=x0)

X=set of all the training images

Y=either class 0 or 1

So this expression here means that given an image what is the probability that this belongs to class k.

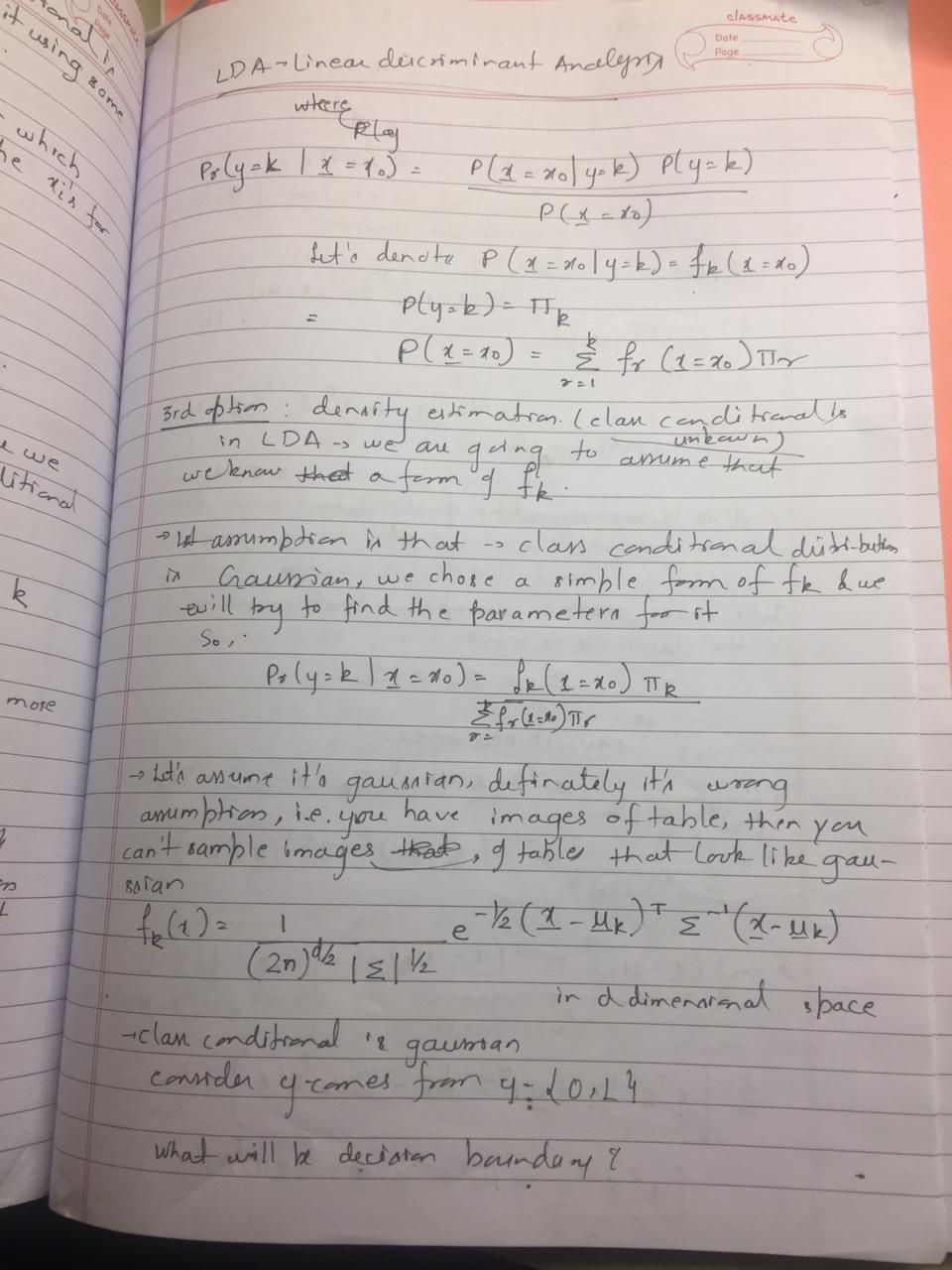
Then at the decision boundary:

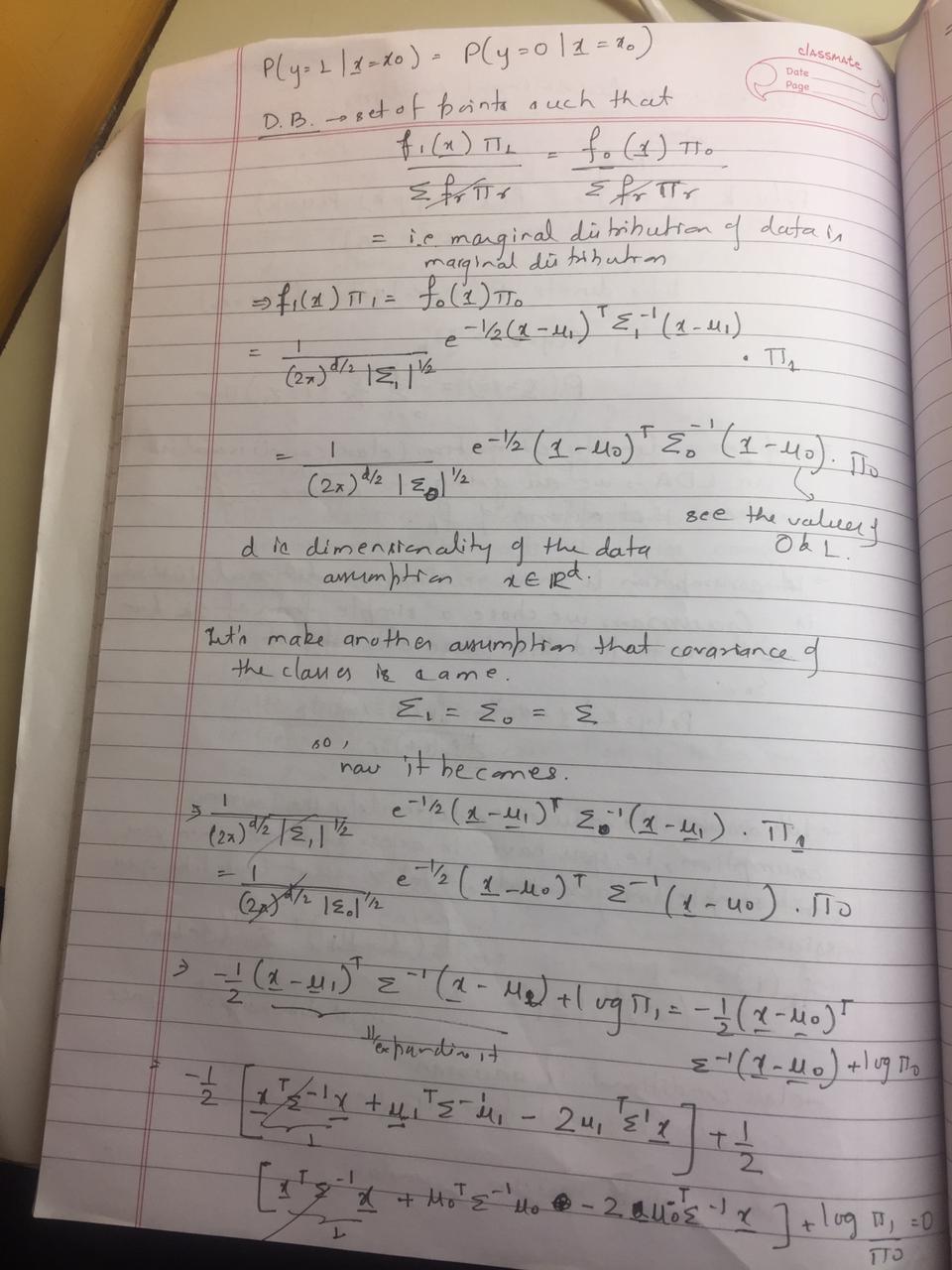
We will have

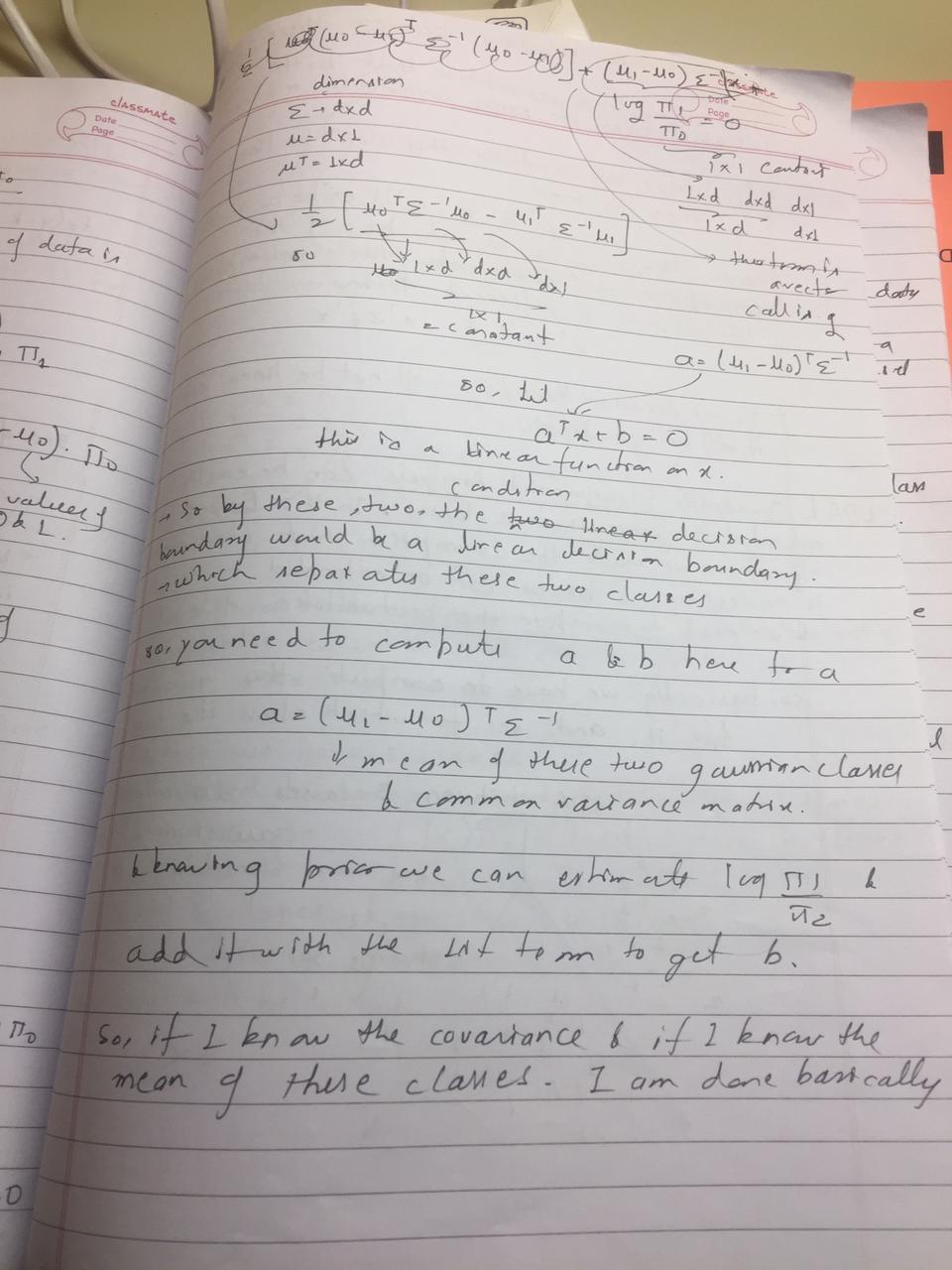
P(Y=1|x=x0) = P(Y=0|x=x0)

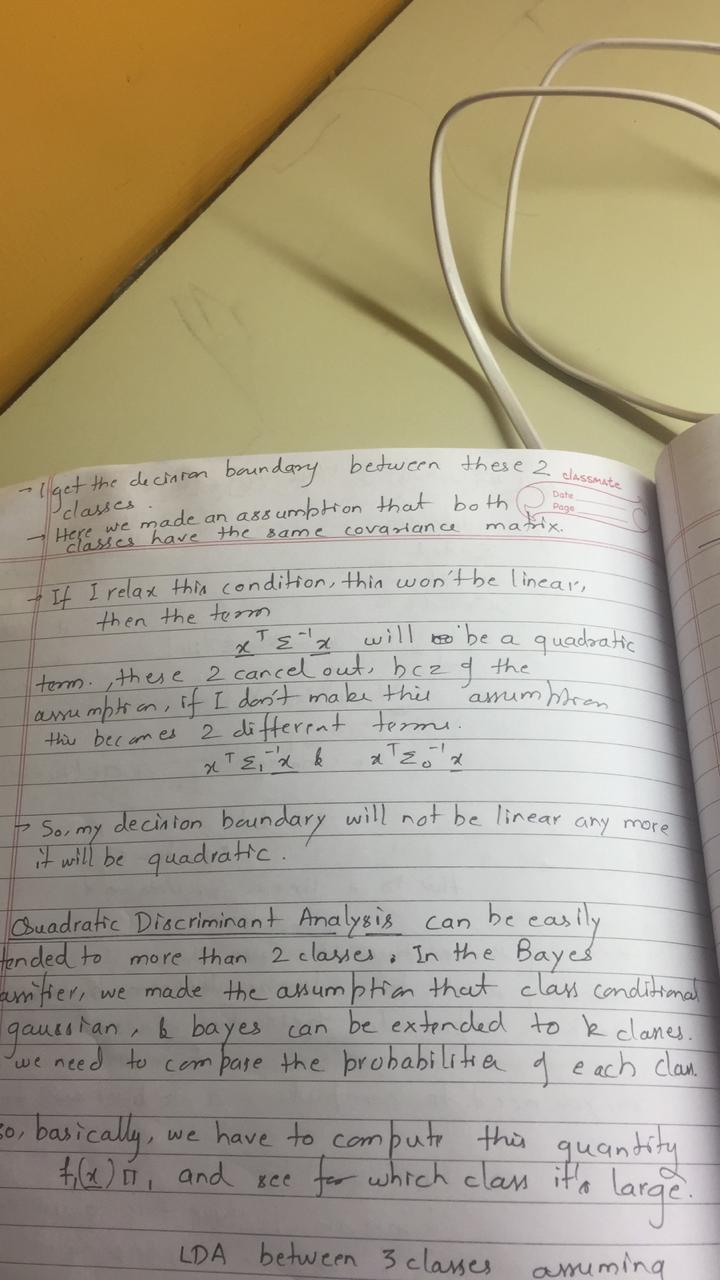
i.e. Given an image the probability of it belonging to class 0 and class 1 is same.

Now equating these terms: we get the following result which I have written below manually on paper:









So as you can see I have clearly derived 2 cases here in which firstly:

We have to calculate the two parameters

a= (mu1-mu0)T E-1

where for 2 classes if we have the same covariance matrix E and mu1 having the mean value of all the images belonging to class 1 and mu0 having the mean value of all the images belonging to class 0.

b= log(prior of class0/prior of class1)

so if we know these two parameters we can easily predict given an image whether it belongs to class 0 or class 1. For whichever class the value comes big it belongs to that class or it is nearer to the mean of that class.

Here one assumption that we make is that the covariance matrix is same. This idea can easily be extended for n no of classes where Y can have n no of classes. So for Y having n no of classes we will calculate for each of the n classes and for whichever class we get maximum value of probability the image will belong to that class. This is called Linear Discriminant Analysis.

Case 2: suppose I don’t relax the condition of covariance matrix being the same for all the classes then in that case we have two terms XT E1-1X and XT E0-1X.

So this term as we see is not linear and hence we have a quadratic determinant analysis. So in this case the decision boundary will not be linear.

This idea can be extended to n classes as well.

Now using the ideas presented above I have implemented a Naïve Bayes Classifier for classes 10 i.e.0,1, 2,…9. Also I have used a Gaussian Function as my class conditional distribution.

So to find the parameters needed here we will use the training dataset. The training dataset is available in following files:

data/MNIST/train-images-idx3-ubyte.gz

data/MNIST/train-labels-idx1-ubyte.gz

So for the implementation of it we need the following information”:

1)prior of each class: occurrence of each class/total occurrence

2)mean of each class: was calculated above

3)Covariance of each class: Covariance is the squared of the standard deviation value.

4) A Gaussian function which uses these parameters and acts like a class conditional distribution.

So the following code was used for this:

#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

#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)

#Creating an array with all the prior values

prior=np.array([prior0,prior1,prior2,prior3,prior4,prior5,prior6,prior7,prior8,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])

#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

#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

#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)

**So these lines of code above basically defines a gaussian function and using the prior, mean and covariance values creates a MODEL that can predict new images i.e. where do the new images belong to.**

**This is the main line of code that calculates to which class the new image belongs to. Let’s call the new image as X and if we want to classify some vector(image), X,**

**argmaxjπj∗Pj(X)**

**i.e maximum of all the values for each class gives to which class this X belongs to.**

**The thing to note here is that we have run it for test dataset that contains 10000 images.**

#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])

Finally, the error rate and the accuracy percent was calculated and it was found to be around **73 percent.** Which is good considering the fact that we were testing for 10000 set of images.

Below lines of code was used for it.

#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)))))

#accuracy percent

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

#73 percent

**Important:**

**I am attaching the pdf file of the code run on Python along with the .py file.**

**Part 2 Question:**

**Why it doesn't perform as good as many other methods on LeCuns web page?**

**Soln:** The following reasons tries to explain why the method implemented by us doesn’t perform as well as other methods on LeCuns web page.

1) Naive Bayes classifier makes a **very strong assumption** on the shape of the data distribution, i.e. any two features are independent given the output class. Due to this, the result can be (potentially) very bad - hence, a “naive” classifier and LeCuns web page discusses a number of methods which do not make this critical assumption.

2) For any possible value of a feature, you need to estimate a likelihood value by a frequentist approach. This can result in probabilities going towards 0 or 1, which in turn leads to numerical instabilities and worse results. In this case, you need to smooth in some way your probabilities, or to impose some prior on your data, however you may argue that the resulting classifier is not naive anymore. This shortcoming is absent in methods involving ANNs and Neural Networks.

3) It is common to use a binning procedure to make them discrete, but if you are not careful you can throw away a lot of information. Another possibility is to use Gaussian distributions for the likelihoods.

4)Baye’s or Naïve Baye’s classifier tends to work well with with very small amounts of training data as compared to methods involving Neural Networks and ANNs. Also here we see that we have 60000 count of training data set and no of classes were 10. So if we had like 2 classes and less no of training data set then we could have got a better accuracy rate. As it is fed increasing quantities of training data, the performance of the Naive Bayes classifier plateaus above a certain threshold. Its simplicity prevents it from benefiting incrementally from training data past a certain point.

5) Naive Bayes’ simplicity prevents it from fitting its training data too closely. In contrast, due to their complexity Neural Networks can very easily over fit training data, especially when provided with large data sets.

6) Here in naïve Bayes’ we have to train each class one by one with the training data where if we consider the neural network implementation mentioned in the website they train on a large dataset for different classes simultaneously.