Pattern Recognition: Assignment 1

Varun Nagaraj: 50290761

March 1st 2019

1 Problem Statement

Part 1: This problem asks us to draw the mean and standard deviation of the image features for the 10 categories as 28 x 28 images using the training images.
Part 2: 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. Compare the performance of this with the ones listed on Lecun’s website and reason why it isn’t as good as the other ones.

2 Bayesian Probability

The Bayesian interpretation of probability can be seen as an extension of propositional logic that enables reasoning with hypotheses, i.e., the propositions whose truth or falsity is uncertain. Unlike traditional probability, which uses a frequency to try to estimate probability, Bayesian probability is generally expressed as a percentage. In its most basic form, it is the measure of confidence, or belief, that a person holds in a proposition.

$$P(\theta|\mathbf{D}) = P(\theta)\frac{P(\mathbf{D}|\theta)}{P(\mathbf{D})} \tag{1}$$

Where,
P($\theta|\mathbf{D}$) = is the conditional probability of θ occurring given D occurs;
P(θ) = is the prior probability θ occurs;
P($\mathbf{D}$—θ) = is the conditional probability that the event D occurs given class θ, which is also termed as the likelihood of occurrence ;

2.1 Discriminant Analysis

Discriminant analysis is a technique that is used by the researcher to analyze the research data when the criterion or the dependent variable is categorical and the predictor or the independent variable is interval in nature. The term categorical variable means that the dependent variable is divided into a number of categories.

The objective of discriminant analysis is to develop discriminant functions that are nothing but the linear combination of independent variables that will discriminate between the categories of the dependent variable in a perfect manner. It enables the researcher to examine whether significant differences exist among the groups, in terms of the predictor variables. It also evaluates the accuracy of the classification.

In practice, the class means and covariances are not known. They can, however, be estimated from the training set. Although the estimates of the covariance may be considered optimal in some sense, this does not mean that the resulting discriminant obtained by substituting these values is optimal in any sense, even if the assumption of normally distributed classes is correct.

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

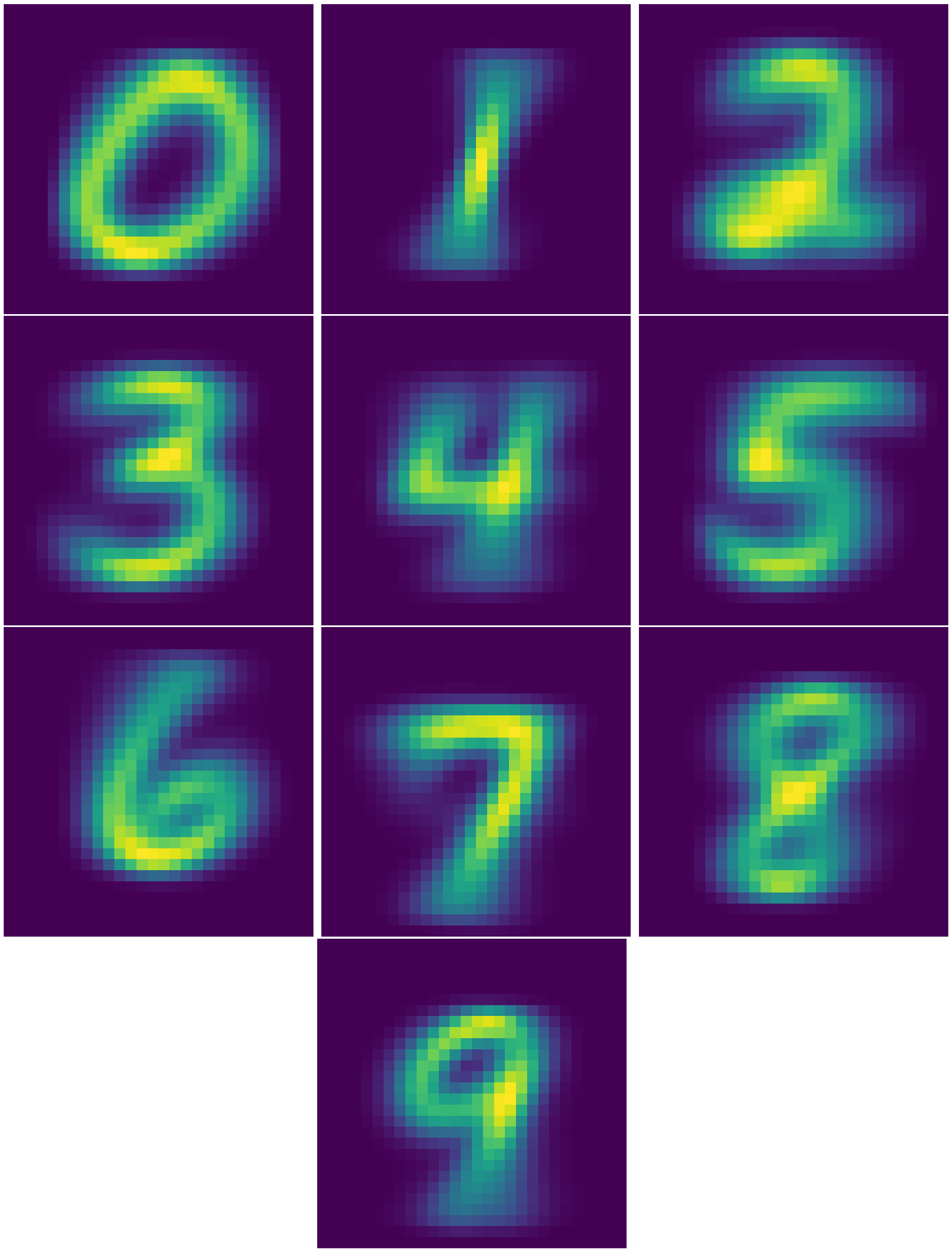
where,
$W_i = -\frac{1}{2}\sum_i^{-1}$,
$N_i = \sum_i^{-1} \mu_i$
$B_{i0} = -\frac{1}{2}\mu_i^t \sum_i^{-1} \mu_i + lnP(\omega_i) - \frac{1}{2}ln|\sum_i|$

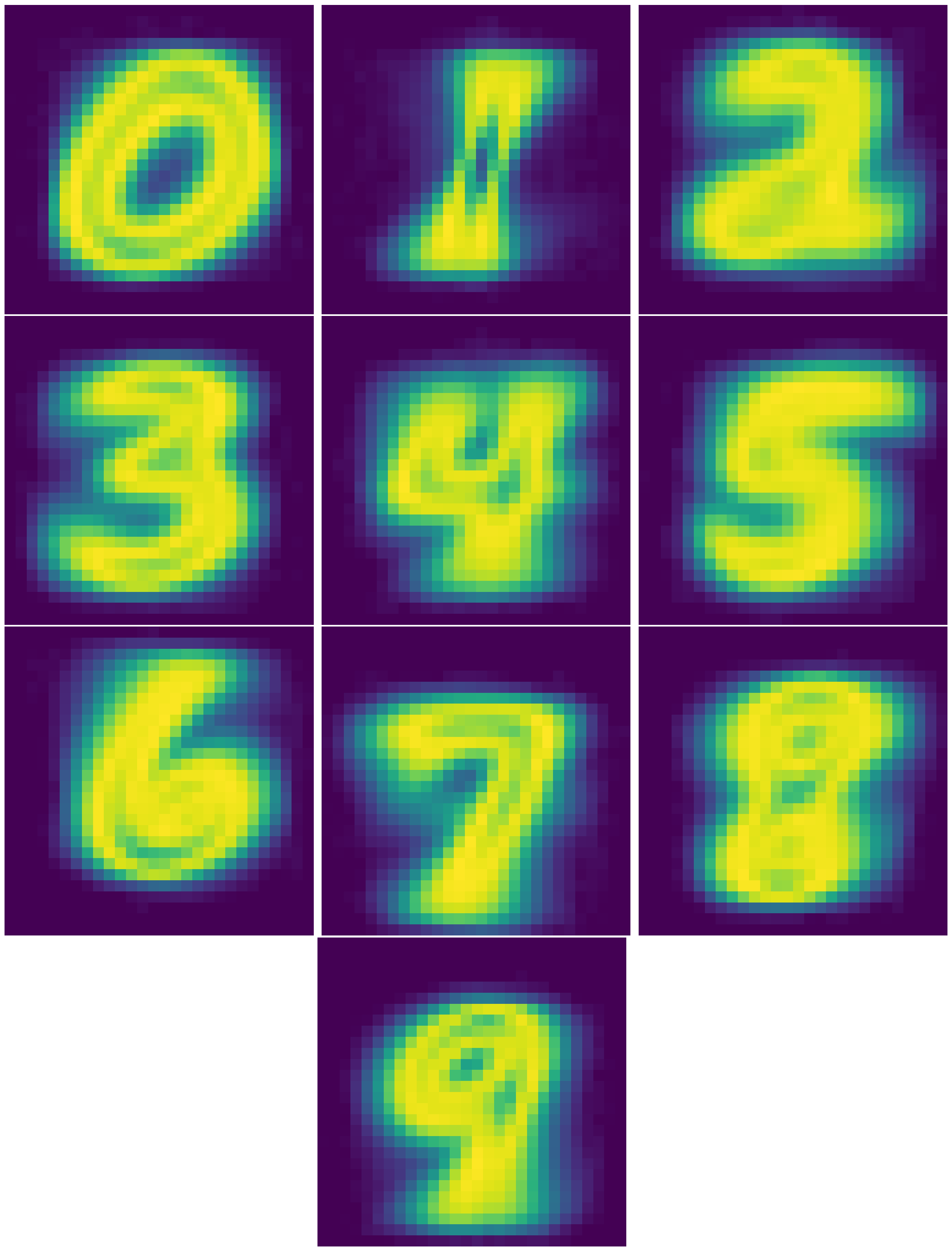
We consider the Quadratic Discriminant function, since there is a possibility that the diagonal elements in the covariance matrix are non-identical.

2.2 Mean and Standard Deviation

The first task of the assignment asks to find the mean and the standard deviation of all the images present in the training set. I iterate over the pixels i.e each element in the 28x28 matrix for 60000 images of training data. At each of these pixels, I maintain a sum array which holds the summation of all the pixel locations of the images. Using this I calculate the mean of each of the 10 numbers i.e 0-9 by dividing the sum array by the number of occurrences of each digit.

Below are the mean and standard deviation images for each of the digits from 0-9.


Mean of all the digit images respectively.



Standard Deviation of all the digits respectively

3 Steps taken for Task 2

In this section, I give a brief overview about the second task in the assignment and the steps I took to accomplish it.
The task is to classify all the images in the testing data my using the mean and the standard deviation of the images that were found. Below steps list the way I performed the classification:

1. Create an array of size 10x784x784 , which will serve as the covariance matrix. Each of the element in the main array will consist of the 784 diagonal covariance values derived from the standard deviation.

2. Add a smoothing factor of 0.0001 to the covariance matrix to prevent division by zero error.

3. For each of the data in the testing set, find the inverse of the covariance matrix of that element and also find the determinant of the covariance matrix.

4. Using these two parameters, given the above mentioned quadratic discriminant function formula, we get the values of each of the computations. Store this in an array.

5. Use the argmax function to find the location of the maximum of the numbers and this result will be between 0-9 and thus will represent the digits that we predict

6. Using the predicted Y values, we compare the values with the actual testing data output.

7. Use the 0-1 loss function to find the loss and the accuracy of the model.

4 Performance Comparison

Using this approach, I managed to get an **accuracy of 0.8173 and a loss of 0.1827**.

This approach lags behind and produces a greater loss compared to the models described on the Lecun's website like Convolution Neural Networks.

The main disadvantage of using Bayesian Decision Rule to solve this problem is that it does not tell us how to select a prior. There is no correct way to choose a prior. Bayesian inferences require skills to translate subjective prior beliefs into a mathematically formulated prior. If you do not proceed with caution, you can generate misleading results.

It can produce posterior distributions that are heavily influenced by the priors. From a practical point of view, it might sometimes be difficult to convince subject matter experts who do not agree with the validity of the chosen prior.

It often comes with a high computational cost, especially in models with a large number of parameters.

These above disadvantages are the reason why the accuracy of the Bayesian Decision rule model isnt the right one for this and we can achieve better accuracy by using the convolutional neural networks where details of the parameters are understood in greater detail.

Although the above disadvantages are significant, there are potential advantages in using the Bayesian rule for analysis. It provides a natural and principled way of combining prior information with data, within a solid decision theoretical framework. We can incorporate past information about a parameter and form a prior distribution for future analysis. When new observations become available, the previous posterior distribution can be used as a prior. All inferences logically follow from Bayes' theorem.

5 Python Code

```
import numpy as np
import matplotlib.pyplot as plt
from mlxtend.data import loadlocal_mnist

#Load the data set using the loadlocal_mnist method
X, y = loadlocal_mnist(
    images_path='train-images-idx3-ubyte',
    labels_path='train-labels-idx1-ubyte')

X_test, y_test = loadlocal_mnist(
    images_path='t10k-images-idx3-ubyte',
    labels_path='t10k-labels-idx1-ubyte')

print('Dimensions: %s x %s' % (X.shape[0], X.shape[1]))

train_features_images = X.reshape(X.shape[0],28,28)
np.amax(train_features_images[0])

def show_images(features_images,labels,start, howmany):
    for i in range(start, start+howmany):
        plt.figure(i)
        plt.imshow(features_images[i], cmap=plt.get_cmap('gray'))
        plt.title(labels[i])
    plt.show()
show_images(train_features_images, y, 1, 20)

plt.imshow(train_features_images[2], cmap=plt.get_cmap('gray'))
plt.show()

# Get the location of elements for each digits and store
   separately
```

```
item_storage = dict()
for i in range(0,10):
    item_storage[i] = np.where(y==i)
print(item_storage)

# Code snippet to find the mean of the images of all digits
mean_array = np.zeros((10,28,28))
for i in range(0,10):
    sum_temp = np.zeros((28,28))
    for z in item_storage[i]:
        for j in z:
            for k in range(0,28):
                for l in range(0,28):
                    sum_temp[k][l]+= train_features_images[j][
                        k][l]
    for k in range(0,28):
        for l in range(0,28):
            sum_temp[k][l]=sum_temp[k][l]/len(item_storage[i][0])
    mean_array[i] = sum_temp

#Save the mean output images of the digits
plt.imsave("/Users/Varun/Documents/Pattern_Recognition/Project1/
    Output/mean_0.png", mean_array[0])
plt.imsave("/Users/Varun/Documents/Pattern_Recognition/Project1/
    Output/mean_1.png", mean_array[1])
plt.imsave("/Users/Varun/Documents/Pattern_Recognition/Project1/
    Output/mean_2.png", mean_array[2])
plt.imsave("/Users/Varun/Documents/Pattern_Recognition/Project1/
    Output/mean_3.png", mean_array[3])
plt.imsave("/Users/Varun/Documents/Pattern_Recognition/Project1/
    Output/mean_4.png", mean_array[4])
plt.imsave("/Users/Varun/Documents/Pattern_Recognition/Project1/
    Output/mean_5.png", mean_array[5])
plt.imsave("/Users/Varun/Documents/Pattern_Recognition/Project1/
    Output/mean_6.png", mean_array[6])
plt.imsave("/Users/Varun/Documents/Pattern_Recognition/Project1/
    Output/mean_7.png", mean_array[7])
plt.imsave("/Users/Varun/Documents/Pattern_Recognition/Project1/
    Output/mean_8.png", mean_array[8])
plt.imsave("/Users/Varun/Documents/Pattern_Recognition/Project1/
    Output/mean_9.png", mean_array[9])

#Code snippet to find the standard deviation of the images
std_array = np.zeros((10,28,28))
variance_arr = np.zeros((10,784))
```

```python
for i in range(0,10):
    sum_temp = np.zeros((28,28))
    for z in item_storage[i]:
        for j in z:
            for k in range(0,28):
                for l in range(0,28):
                    sum_temp[k][l] += np.power((train_features_images[j][k][l] - mean_array[i][k][l]),2)
    for k in range(0,28):
        for l in range(0,28):
            sum_temp[k][l]=np.sqrt(sum_temp[k][l]/len(item_storage[i][0]))
    std_array[i] = sum_temp
    variance_arr[i] = (sum_temp*sum_temp).ravel()

#Save the standard deviation output images
plt.imsave("/Users/Varun/Documents/Pattern_Recognition/Project1/Output/std_0.png", std_array[0])
plt.imsave("/Users/Varun/Documents/Pattern_Recognition/Project1/Output/std_1.png", std_array[1])
plt.imsave("/Users/Varun/Documents/Pattern_Recognition/Project1/Output/std_2.png", std_array[2])
plt.imsave("/Users/Varun/Documents/Pattern_Recognition/Project1/Output/std_3.png", std_array[3])
plt.imsave("/Users/Varun/Documents/Pattern_Recognition/Project1/Output/std_4.png", std_array[4])
plt.imsave("/Users/Varun/Documents/Pattern_Recognition/Project1/Output/std_5.png", std_array[5])
plt.imsave("/Users/Varun/Documents/Pattern_Recognition/Project1/Output/std_6.png", std_array[6])
plt.imsave("/Users/Varun/Documents/Pattern_Recognition/Project1/Output/std_7.png", std_array[7])
plt.imsave("/Users/Varun/Documents/Pattern_Recognition/Project1/Output/std_8.png", std_array[8])
plt.imsave("/Users/Varun/Documents/Pattern_Recognition/Project1/Output/std_9.png", std_array[9])

covar_matrix = np.zeros((10, 784,784), dtype=np.float64)
for i in range(0,10):
    for key, value in enumerate(variance_arr[i]):
        covar_matrix[i][key][key] = value + 0.0001

#Find the predicted values using the formula for Quadratic discriminant analysis
```

```
y_pred = []
for i in X_test:
    final = []
    for j in range(10):
        inv = np.linalg.inv(covar_matrix[j])
        Wi = -0.5* inv
        Ni = np.matmul(mean_array[j].ravel(), inv)
        _, determinant = np.linalg.slogdet(covar_matrix[j])
        Bi = -0.5*np.matmul(np.matmul(mean_array[j].ravel().T, inv
            ), mean_array[j].ravel()) + np.log(0.1)
        final.append(np.matmul(np.matmul(i.T, Wi), i) + np.matmul(
            Ni, i) + Bi)
    y_pred.append(np.argmax(np.array(final)))

#Implement a 0-1 loss function
loss = 0.0
for i, val in enumerate(y_pred):
    if val != y_test[i]:
        loss +=1
print("0-1 Loss is : {}".format((loss/len(y_pred))*100))
print("Accuracy is  {}".format(100 - (loss/len(y_pred))*100))
```

6 References

1. Linear Discriminant Analysis
2. Quadratic Discriminant Analysis
3. Overview of Discriminant Analysis
4. MNIST Dataset of handwritten digits - Lecun