2)

first imported needed libraries, then read data from csv files similar to the lab sessions.

```
In [1]: import math
        import matplotlib.pyplot as plt
        import numpy as np
        import scipy.stats as stats
        import pandas as pd
        from sklearn.metrics import confusion_matrix
```

Part 2

```
In [2]: images_data = np.genfromtxt("hw02_images.csv", delimiter = ",")
        labels_data = np.genfromtxt("hw02_labels.csv", delimiter = ",")
```

3)

In this part, I divide data in two part. First one is train data which from 0 to 30000th data. remaining 5000 data is the test data for us. Test data will be used for comparing with results.

Part 3

```
In [3]: train_images = images_data[:30000,:]
        test_images = images_data[30000:,:]
        train_label = labels_data[:30000]
        test_label = labels_data[30000:]
```

4)

In calculating parameters, first I got K(number of classes),N(number of data for train) values. Then I calculated sample means from given data with corresponding formula below(similar to the hw1).

$$\hat{\mu} = \frac{\sum_{i=1}^{N} x_i}{N}$$

Part 4

```
In [4]: K = int(np.max(train_label))
        N = train_images.shape[0]
```

```
In [5]: # calculate sample means
        sample_means = [np.mean(train_images[train_label == (c + 1)], axis=0) for c in range(K)]

        print("\033[4msample_means\033[0m \n")
        for sample_mean in sample_means:
            print(sample_mean,"\n")
```

```
 109.96      110.918      108.76983333 107.04866667 108.504
 115.60616667 119.88633333 122.72216667 148.33566667 211.3505
 246.74566667 253.69616667 254.792      254.82066667 253.881
 249.504     222.3685     156.3565     121.28766667 114.62916667
 113.30266667 107.60866667 106.1145     108.56866667 110.5355
 111.91583333 110.12183333 110.85366667 111.216      111.39216667
 111.08433333 108.57683333 107.60633333 114.49516667 117.17116667
 119.638     137.58466667 193.77783333 240.33333333 252.56983333
 254.73466667 254.658      253.2545     246.29216667 209.29166667
 147.5005    121.31816667 110.16766667 106.227      104.98583333
 106.128     109.18483333 110.16783333 111.1915     111.12633333
 111.79216667 111.59033333 111.502      112.1635     110.057
```

I calculated sample deviations by using np.std. Calculated sample deviations is a matrix with shape (5,784)

```
In [7]: np.shape(sample_deviations)
Out[7]: (5, 784)
```

Then calculated class priors with given formula, similar to the hw1.

$$\widehat{P}(y_i = c) = \frac{\sum_{i=1}^{N}(y_i = c)}{N}$$

```
In [8]: # calculate sample deviations
        sample_deviations = [np.std(train_images[train_label == (c + 1)], axis=0) for c in range(K)]

        print("\033[4msample_deviations\033[0m \n")
        for sample_deviation in sample_deviations:
            print(sample_deviation,"\n")
```

```
[9.12773551e-02 2.56091075e-01 1.31090756e+00 3.80543465e+00
 5.27948907e+00 6.97889132e+00 1.07720867e+01 2.09088724e+01
 3.74438435e+01 5.25122406e+01 6.43785189e+01 7.09060378e+01
 6.86627306e+01 6.22709378e+01 6.19797698e+01 6.60298794e+01
 7.33258709e+01 7.11195000e+01 6.13707817e+01 4.58070656e+01
 2.86522563e+01 1.50082488e+01 7.59281098e+00 5.46698180e+00
 4.67702088e+00 2.99681671e+00 3.74178211e-01 1.58063278e-01
 2.62280543e-01 4.01607880e-01 5.07890964e+00 1.03206524e+01
 1.34200830e+01 2.58190550e+01 5.23572148e+01 7.39564543e+01
 8.42104109e+01 8.71560628e+01 8.38912910e+01 7.73179110e+01
 8.03720088e+01 8.25237400e+01 8.31962000e+01 8.15289709e+01
 7.79408454e+01 8.11367681e+01 8.62364334e+01 8.68170035e+01
 8.12445772e+01 6.58150663e+01 3.95159939e+01 1.96698496e+01
 1.21747777e+01 7.81098963e+00 3.62960786e+00 2.21591915e+00
 2.89996935e-01 1.37410164e+00 7.24785055e+00 1.33104165e+01
 2.90947684e+01 6.50654043e+01 8.43708969e+01 8.47847791e+01
 7.97477738e+01 7.39575743e+01 7.21152673e+01 7.39621616e+01
 7.38031151e+01 7.18858657e+01 7.35252917e+01 7.27806823e+01
 7.50570243e+01 7.30847213e+01 7.27984804e+01 7.65890656e+01
```

```
In [10]: # calculate prior probabilities
         class_priors = [np.mean(train_label == (c + 1)) for c in range(K)]

         print("\033[4mclass_priors\033[0m \n", class_priors)
```

```
class_priors
 [0.2, 0.2, 0.2, 0.2, 0.2]
```

5)

Using estimated parameters, I used naive bayes classifier as parametric classification rule. Used this g score formula to classify data.

Part 5

$$g_c(x) = \log\left(p(x|y=c)\right) + \log\left(P(y=c)\right)$$

$$g_c(x) = \log\left(\frac{1}{\sqrt{2\pi\sigma_c^2}} . e^{-\frac{(x-\mu_c^2)^2}{2\sigma_c^2}}\right) + \log\left(P(y=c)\right)$$

$$g_c(x) \approx \sum_{i=1}^{N} \left[-\frac{1}{2}\log 2\pi\sigma_c^2 - \frac{(x-\mu_c^2)^2}{2\sigma_c^2}\right] + \log\left(P(y=c)\right)$$

Created g score function for the train and test data with given formula above.

```
In [20]: def score_def(x):
             scores = np.array([0, 0, 0, 0, 0])
             for i in range(K):
                 scores[i]  = np.sum((-0.5 * ( np.log(2 * math.pi * (sample_deviations[i]**2) ))) +
                                     (-0.5 * ((x-sample_means[i])**2) /  sample_deviations[i]**2 )) + np.log(class_priors[i])
             return scores
```

```
In [15]: g_scores_train = [score_def(train_images[i]) for i in range(np.shape(train_images)[0])]
         g_scores_test = [score_def(train_images[i]) for i in range(np.shape(test_images)[0])]
```

After finding g scores of the data, to find their class I used this function. I could use np.argmax(g_scores, axis=1) where axis=1 means for every row.

```
In [35]: def get_labels(pred,g_scores):
             for i in range(len(g_scores)):
                 max_g=np.max(g_scores[i])
                 if g_scores[i][0]==max_g:
                     pred.append(1)
                 elif g_scores[i][1]==max_g:
                     pred.append(2)
                 elif g_scores[i][2]==max_g:
                     pred.append(3)
                 elif g_scores[i][3]==max_g:
                     pred.append(4)
                 else :
                     pred.append(5)

             pred_label=np.array(pred)
             return pred_label

         train_pred = get_labels([],g_scores_train)
         test_pred = get_labels([],g_scores_test)

         #train_pred = np.argmax(g_scores_train, axis = 1)+1
         #test_pred = np.argmax(g_scores_test, axis = 1)+1
```

Finally, I found confusion matrices for data points in train data and test data. confusion matrix for train data is similar to what we expected. However, in test data, we are not good at predictions.

```
In [36]: confusion_matrix = pd.crosstab(train_pred, train_label, rownames=['y_pred'], colnames=['y_truth'])
         print("\033[4mconfusion_matrix\033[0m \n", confusion_matrix)
```

```
confusion_matrix
 y_truth   1.0   2.0   3.0   4.0   5.0
y_pred
1         3688    49     4   680     6
2         1430  5667  1141  1380   533
3          505   208  4669  2949   892
4          234    60   123   686   180
5          143    16    63   305  4389
```

```
In [37]: confusion_matrix = pd.crosstab(test_pred, test_label, rownames=['y_pred'], colnames=['y_truth'])
         print("\033[4mconfusion_matrix\033[0m \n", confusion_matrix)
```

```
confusion_matrix
 y_truth  1.0  2.0  3.0  4.0  5.0
y_pred
1         148  138  145  143  174
2         328  362  346  326  323
3         300  309  285  337  288
4          40   46   69   46   42
5         184  145  155  148  173
```