ENGR421

HW2

Doğa Demirtürk
68859

In this homework, we are given a data of images of size 28x28 pixels. The aim of the homework is to classify these images of clothing (t-shirt, dress, coat, shirt, bag) using naïve Bayes' classifier algorithm.

First, I read the Section 5.7 from the textbook to learn about naïve Bayes' classifier. Naïve Bayes' algorithm is based on Bayes' Theorem. It is called "naïve" since it assumes independence between the features of the input. Therefore, we use deviations instead of covariances.

I started by importing data into the memory. It takes a while to do so since there are 35000 images of 28x28 pixels (35000x784) and 35000 labels corresponding to each one. Then I divided data into 2: one for the training set (first 30000 images) and the other for the test set (last 5000 images).

In [2]:
```
# read data into memory
images_data_set = np.genfromtxt("hw02_images.csv", delimiter = ",")
labels_data_set = np.genfromtxt("hw02_labels.csv", delimiter = ",").astype(int)
```

In [3]:
```
print(images_data_set.shape)
print(labels_data_set.shape)
```

```
(35000, 784)
(35000,)
```

In [4]:
```
# divide data set into two parts: training set and test set
x_training = images_data_set[0:30000, :]
x_test = images_data_set[30000:35000, :]
y_training = labels_data_set[0:30000]
y_test = labels_data_set[30000:35000]
```

Figure 1 Dividing data into two

After dividing the data, I took the number of classes from the label values to be used in the calculations. Afterwards, I calculated sample means using the same technique I used for the first homework.

```
In [8]: # calculate sample means
        sample_means = np.array([np.mean(x_training[y_training == (c + 1)], axis=0) for c in range(K)])
```

Figure 2 Sample means calculation

When I printed out the calculations, the resulting sample means were the same as desired in the homework description pdf.

```
In [9]: print(sample_means)
        print(sample_means.shape)
```

```
[[254.99866667 254.98416667 254.85616667 ... 254.679      254.87816667
  254.95933333]
 [254.99733333 254.99733333 254.9965     ... 254.96883333 254.99216667
  254.98866667]
 [254.99933333 254.99933333 254.99233333 ... 251.52483333 254.4725
  254.97483333]
 [254.99666667 254.98983333 254.91416667 ... 252.39516667 254.44166667
  254.93666667]
 [254.999      254.98433333 254.93783333 ... 250.673      253.23333333
  254.79083333]]
(5, 784)
```

Figure 3 Sample means

Then I calculated the standard deviation parameters of the classes. I used numpy.std() function to calculate the deviations. Then again, the printed sample deviations were the same as desired in the description pdf.

```
In [10]: # calculate sample deviations
         sample_deviations = np.array([np.std(x_training[y_training == (c + 1)], axis=0) for c in range(K)])
```

```
In [11]: print(sample_deviations)
         print(sample_deviations.shape)
```

```
[[ 0.09127736  0.25609108  1.31090756 ...  5.29826629  3.9117332
   1.93959091]
 [ 0.2065419   0.2065419   0.2163818  ...  1.04076669  0.47057267
   0.70062226]
 [ 0.05163547  0.04081939  0.16002465 ... 18.43665868  6.7881694
   1.1061344 ]
 [ 0.18436076  0.21617116  1.81046936 ... 15.67799977  6.34549162
   1.79971911]
 [ 0.04471018  0.64582342  3.03248555 ... 23.62576428 13.9167006
   4.4727787 ]]
(5, 784)
```

Figure 4 Sample deviations

Finally for step 4 of the homework, I calculated prior class probabilities, again in the same manner as in my first homework. The calculations are the same as in the description pdf.

```
In [12]: # calculate prior probabilities
         class_priors = np.array([np.mean(y_training == (c + 1)) for c in range(K)])

In [13]: print(class_priors)
         print(class_priors.shape)

         [0.2 0.2 0.2 0.2 0.2]
         (5,)
```

Figure 5 Class Prior Probabilities

For the last two parts of the homework, I defined a score function which calculates the score values of each class for inputs. I used score function that I derived from the lecture notes on univariate parametric classification since the features are assumed to be independent. Then I choose the maximum of the scores to predict the labels.

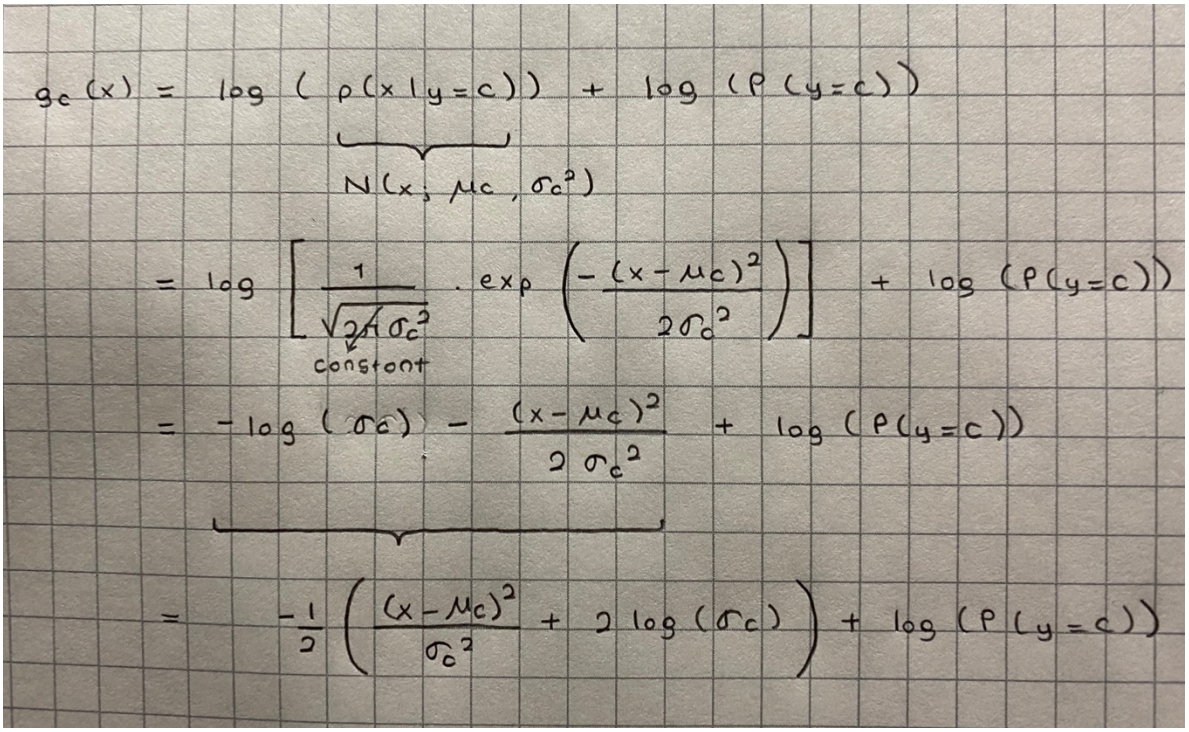

Figure 6 Score function (lecture notes)

Finally, I used the same method as in the first homework to find confusion matrices. In the end, I had two confusion matrices, one for the training data set and the other for the test data set, which are the same as in the manual.

In [15]:
```
y_score_training = score(x_training)
y_pred_training = np.argmax(y_score_training, axis=1) + 1
confusion_matrix_training = pd.crosstab(y_pred_training, y_training, rownames = ['y_pred'], colnames = ['y_truth'])
```

In [16]:
```
print(confusion_matrix_training)
```

```
y_truth     1     2     3     4     5
y_pred
1        3685    49     4   679     6
2        1430  5667  1140  1380   532
3         508   208  4670  2948   893
4         234    60   123   687   180
5         143    16    63   306  4389
```

In [17]:
```
y_score_test = score(x_test)
y_pred_test = np.argmax(y_score_test, axis=1) + 1
confusion_matrix_test = pd.crosstab(y_pred_test, y_test, rownames = ['y_pred'], colnames = ['y_truth'])
```

In [18]:
```
print(confusion_matrix_test)
```

```
y_truth    1    2    3    4    5
y_pred
1        597    6    0  114    1
2        237  955  188  267   81
3         92   25  785  462  167
4         34   11   16  109   29
5         40    3   11   48  722
```

Figure 7 Confusion matrices for training and test sets