Homework 02 – Naïve Bayes' Classifier

Importing Data

- After importing NumPy and Pandas libraries, I've imported and combined images and labels csv files into a single DataFrame object called "data".

Train-Test Split

- I have divided the data into "train" and "test" DataFrames by assigning the first 30000 data points to the former, and the rest of the 5000 data points to the latter:

```
Dataset shape: (35000, 785)

Number of unique labels: 5

Unique label values: [1, 2, 3, 4, 5]

Train set shape: (30000, 785)
Test set shape: (5000, 785)
X_train shape: (30000, 784)
X_test shape: (5000, 784)
y_train shape: (30000,)
y_test shape: (5000,)
```

- I also splitted the train data into five different DataFrames, "X_1", "X_2", "X_3", "X_4", and "X_5", with respect to their labels:

```
X_1 = train[train['Label'] == 1].drop('Label', axis=1)
X_2 = train[train['Label'] == 2].drop('Label', axis=1)
X_3 = train[train['Label'] == 3].drop('Label', axis=1)
X_4 = train[train['Label'] == 4].drop('Label', axis=1)
X_5 = train[train['Label'] == 5].drop('Label', axis=1)

Xs = [X_1, X_2, X_3, X_4, X_5]
```

Estimating the Parameters

- To estimate sample means, sample deviations and class priors, I used the following formulas from the 4th chapter of the book:

Sample Mean

Sample mean: $m = \frac{\sum_t x^t}{N}$

Standard Deviation

Variance: $s^2 = \frac{\sum_t (x^t - m)^2}{N}$

Standard Deviation: $s = \sqrt{\frac{\sum_t (x^t - m)^2}{N}}$

Prior Probabilities

Prior Probability: $\hat{P}(C_i) = \frac{\sum_t r_i^t}{N}$

- I've written "estimate_sample_mean", "estimate_standard_deviation", and "prior_probability" functions using the previous formulas in order to apply each of them separately on the five DataFrames I have generated previously:

```
def estimate_sample_mean(X):
    return [np.sum(X.iloc[:, i]) / X.shape[0] for i in range(X.shape[1])]
```

```
sample_means = np.array([estimate_sample_mean(X) for X in Xs])
print(sample_means)
```

```
def estimate_standard_deviation(X, sample_mean):
    return [np.sqrt(np.sum((X.iloc[:, i] - sample_mean[i])**2)/X.shape[0]) for i in range(X.shape[1])]
```

```
sample_deviations = np.array([estimate_standard_deviation(X, sample_mean) for X, sample_mean in zip(Xs, sample_means)])
print(sample_deviations)
```

```
def prior_probability(X, all_X):
    return X.shape[0] / all_X.shape[0]
```

```
class_priors = [prior_probability(X, train) for X in Xs]
print(class_priors)
```

Naïve Bayes' Classifier

- Then, I had to develop a classification algorithm in order to make predictions and calculate the confusion matrix. To develop a classification algorithm, I used the following discriminant function for Gaussian Density from the section 4.5 of our book:

$$g_i(\mathbf{x}) = -\frac{1}{2}\log 2\pi - \log s_i - \frac{(x-m_i)^2}{2s_i^2} + \log \hat{P}(C_i)$$

- I have created a function named "discriminant_function" in which I could place x, sample mean, sample deviation, and prior probability I have calculated earlier, and apply the above formula.

```python
def discriminant_function(x, sample_mean, sample_deviation, class_prior):
    return np.sum((-1/2 * np.log(2*np.pi))
                  - (np.log(sample_deviation)) - ((x - sample_mean)**2) / (2 * sample_deviation**2)
                  + np.log(class_prior))
```

Classification Algorithm

- Lastly, I have written a "predict" function, which calculates scores for each class using "discriminant_function", append them to a list named "score", then return the index with the highest value. I had to add 1 to the index, since indices were [0, 1, 2, 3, 4] while the classes I am predicting had values [1, 2, 3, 4, 5]. I applied the "predict" function for all the data points in my data matrix and appended the predictions into arrays called "y_pred_train" and "y_pred_test".

```python
def predict(x):

    scores = []

    for i in range(5):
        scores.append(discriminant_function(x, sample_means[i], sample_deviations[i], class_priors[i]))

    scores = pd.Series(scores)
    return scores[scores == np.max(scores)].index[0] + 1
```

```python
y_pred_train = np.array([predict(X_train.iloc[i, :]) for i in range(X_train.shape[0])])
y_pred_test = np.array([predict(X_test.iloc[i, :]) for i in range(X_test.shape[0])])
```

Confusion Matrix

- By using Pandas's crosstab function, I have created two confusion matrices, one for the training set, and another for the test set.

Calculating Confusion Matrix

Train Set

```python
confusion_matrix_train = pd.crosstab(y_pred_train, y_train, rownames = ["y_pred"], colnames = ["y_truth"])

print("Confusion Matrix - Training Set:")
display(confusion_matrix_train)
```

Confusion Matrix - Training Set:

y_truth / y_pred	1	2	3	4	5
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

Test Set

```python
confusion_matrix_test = pd.crosstab(y_pred_test, y_test, rownames = ["y_pred"], colnames = ["y_truth"])

print("Confusion Matrix - Test Set:")
display(confusion_matrix_test)
```

Confusion Matrix - Test Set:

y_truth / y_pred	1	2	3	4	5
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