Assignment_1(1)

Handling missing values:

1. remove the samples having miss values(data having missing values is within a tolerable limit)
2. Imputing missing values
    - For quantitative attributes, missing values can be imputed with the mean, median, or mode of the remaining values under the same attribute
    - For qualitative attributes, missing values can be imputed by the mode of all remaining values of the same attribute
    - Another strategy is to identify the similar types of observations whose values are known and use the mean/median/mode of those known values

3- Estimate missing values(For finding similar data points or observations, distance function can be used)

Handling outliers

1. Remove outliers: If the number of outliers is not many, a simple approach may be to remove them
2. Imputation: Impute the value with mean or median or mode. The mean/median/mode of the most similar samples may also be used for imputation
3. Capping: For values that lie outside the 1.5 times of IQR limits, we can cap them by replacing those observations with the value of 5th percentile or the value of 95th percentile

otherwise:If there is a significant number of outliers, they should be treated separately in the statistical model: the data should be treated as two different datasets, the model should be built for both. However, if the outliers are natural, i.e. because of a valid reason, then we should not amend it.

Assignment_1(2)

(2) Compute Class Priors.

class label: $E \in \{0, 1, 2, 3\}$

$$P(w_k) = \frac{N_k}{N}$$

estimate feature distribution: $p(x_1, x_2, x_3, x_4 | w_k) = \prod_{j=1}^{4} p(x_j | w_k)$

for Gaussian naive Bayes: $p(x | w_j) = \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left[-\frac{1}{2}\left(\frac{x-\mu_j}{\sigma_j}\right)^2\right]$

Derive the discriminant function: $g_k(x) = \ln P(w_k) + \sum_{j=1}^{4} \ln p(x_j | w_k)$

$$\hat{w}(x) = \arg \max_k g_k(x)$$

Assignment_1(3)

```
import pandas as pd
import numpy as np

pd.set_option('future.no_silent_downcasting', True)

TRAIN_PATH = "TrainingData.xlsx"
TEST_PATH  = "TestData.xlsx"

train = pd.read_excel(TRAIN_PATH, header=None)
test  = pd.read_excel(TEST_PATH, header=None)

train.columns = list("ABCDE")  # A-D features, E label
test.columns  = list("ABCD")

# Replace "?" with NaN
train = train.replace("?", np.nan)
test  = test.replace("?", np.nan)

# Remove rows with missing values
train = train.dropna()
test  = test.dropna()

X_train = train[["A", "B", "C", "D"]].astype(float)
y_train = train["E"].astype(int)
X_test  = test[["A", "B", "C", "D"]].astype(float)

# Calculate class prior probabilities
classes = np.unique(y_train)
priors = {c: np.mean(y_train == c) for c in classes}

# Calculate mean and variance for each feature per class (Gaussian assumption)
mu = {c: X_train[y_train == c].mean().values for c in classes}
var = {c: X_train[y_train == c].var(ddof=0).values + 1e-9 for c in classes}

# Define log Gaussian discriminant function
def log_gaussian(x, mean, var):
    return -0.5 * np.sum(np.log(2 * np.pi * var) + ((x - mean) ** 2) / var)

def discriminant_scores(x):
    scores = {}
    for c in classes:
        scores[c] = np.log(priors[c]) + log_gaussian(x, mu[c], var[c])
    return scores

# Predict on the test set
preds, scores_list = [], []
for i, row in X_test.iterrows():
```

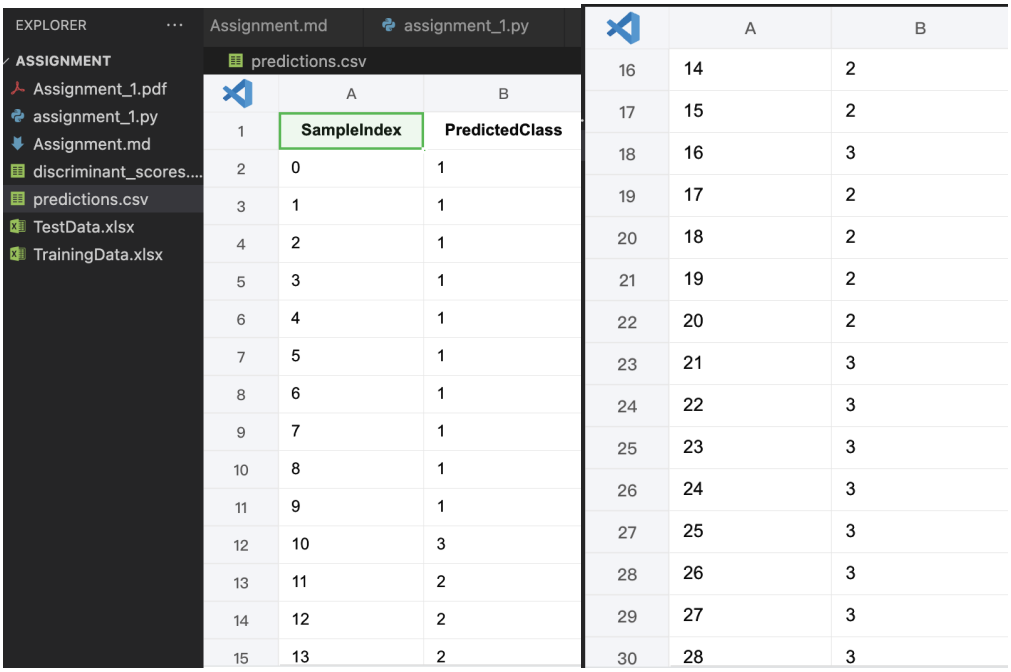
```
    x = row.values
    scores = discriminant_scores(x)
    scores_list.append({'id': i, **scores})
    preds.append((i, max(scores, key=scores.get)))

pred_df = pd.DataFrame(preds, columns=["SampleIndex", "PredictedClass"])
pred_df.to_csv("predictions.csv", index=False, encoding="utf-8-sig")

scores_df = pd.DataFrame(scores_list)
scores_df.to_csv("discriminant_scores.csv", index=False, encoding="utf-8-sig")

print("Finished! Predictions saved to predictions.csv")
```

And the result:



Assignment_2

(1) calculate the mean vectors m_1 and m_2 of samples in the two classes, respectively: $m_i = \frac{1}{n_i} \sum_{X \in D_i} X$

Calculate the within-class scatter matrix S_w

$$S_i = \sum_{X \in D_i} (X - m_i)(X - m_i)^T$$

$$S_w = S_1 + S_2$$

Calculate w and w_0

$$w = S_w^{-1}(m_1 - m_2)$$

$$w_0 = -\frac{w^T(m_1 + m_2)}{2}$$

Construct discriminant function and classify samples

$$g(x) = w^T x + w_0$$

Decide w_1 if $g(x) > 0$

decide w_2 if $g(x) < 0$

On decision boundary if $g(x) = 0$.

```
import numpy as np
import pandas as pd
import scipy.io as sio

# use scipy.io.loadmat read .mat file
data_train = sio.loadmat("data_train.mat")['data_train']
label_train = sio.loadmat("label_train.mat")['label_train'].ravel()
data_test = sio.loadmat("data_test.mat")['data_test']

# divide data by class
X1 = data_train[label_train == 1]
X2 = data_train[label_train == -1]

# compute class means
m1 = np.mean(X1, axis=0)
m2 = np.mean(X2, axis=0)

# within-class scatter matrix
S1 = (X1 - m1).T @ (X1 - m1)
S2 = (X2 - m2).T @ (X2 - m2)
Sw = S1 + S2

# weight vector w
w = np.linalg.inv(Sw) @ (m1 - m2)

# threshold w0
y0 = 0.5 * w.T @ (m1 + m2)
w0 = -y0

print("Weight vector w =", w)
print("Bias term w0 =", w0)

# discriminant function
def predict(x):
    return 1 if w.T @ x + w0 > 0 else -1

# predict test set
preds = np.array([predict(x) for x in data_test])
pd.DataFrame(preds, columns=
["PredictedClass"]).to_csv("fisher_predictions.csv", index=False)
print("Predictions saved to fisher_predictions.csv")
```

result of w and w0:

```
(normal-ML) yuhang@192 assignment_2 % /Users/yuhang/miniconda3/envs/normal-ML/bin/python /Users/yuh
ang/Desktop/NTUS2/6407/assignment/assignment_2/assignment_2.py
Weight vector w = [-0.00178602 -0.0034729  -0.00194673 -0.00168257 -0.00265808]
Bias term w0 = 0.008700458474367305
Predictions saved to fisher_predictions.csv
```

predict result: