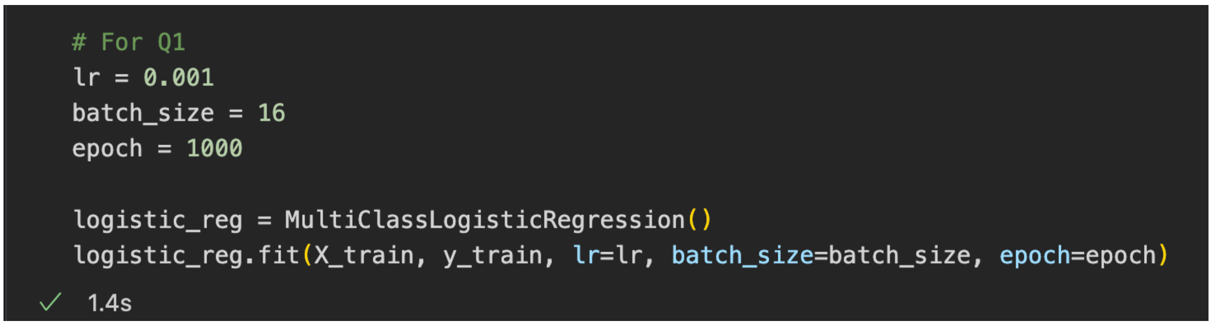
NYCU Pattern Recognition, Homework 2

311552034, 林柏翰

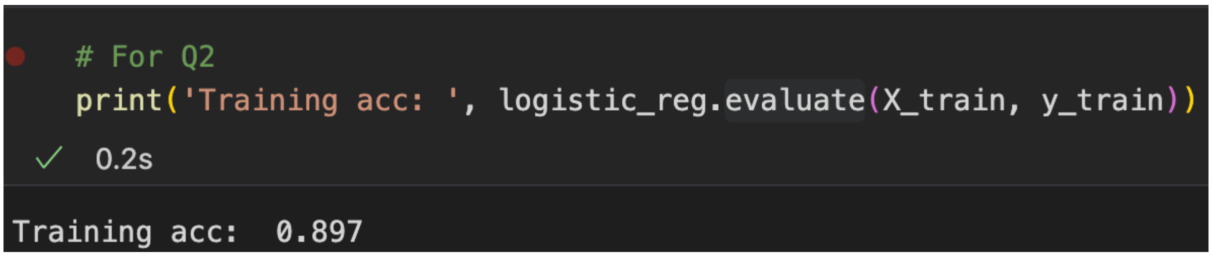
Part. 1, Coding (70%):

1. (0%) Show the learning rate, epoch, and batch size that you used.
   learning rate: _0.001__
   epoch: _1000__
   batch size: _16__



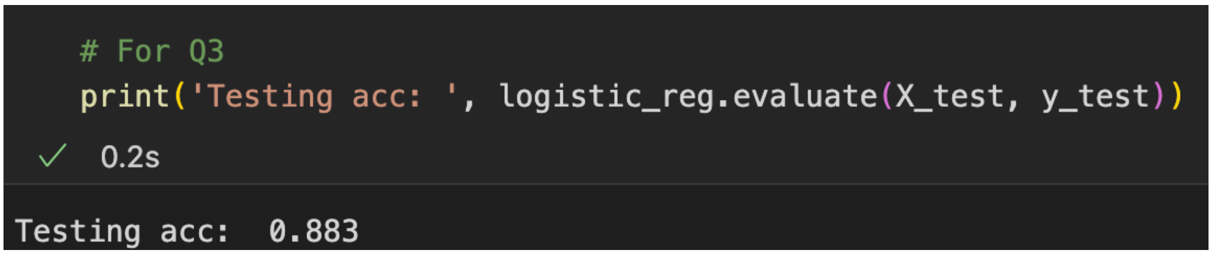
```
# For Q1
lr = 0.001
batch_size = 16
epoch = 1000

logistic_reg = MultiClassLogisticRegression()
logistic_reg.fit(X_train, y_train, lr=lr, batch_size=batch_size, epoch=epoch)
✓ 1.4s
```

2. (5%) What's your training accuracy?
   Training acc:___0.897___



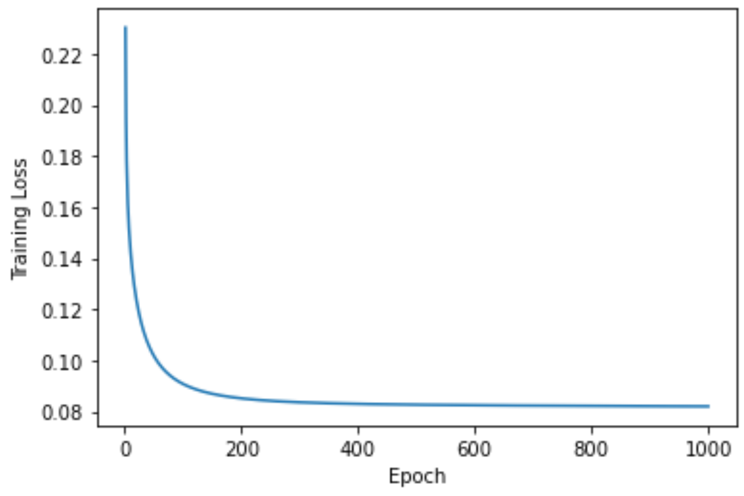
```
# For Q2
print('Training acc: ', logistic_reg.evaluate(X_train, y_train))
✓ 0.2s

Training acc:  0.897
```

3. (5%) What's your testing accuracy?
   Testing acc:___0.883___

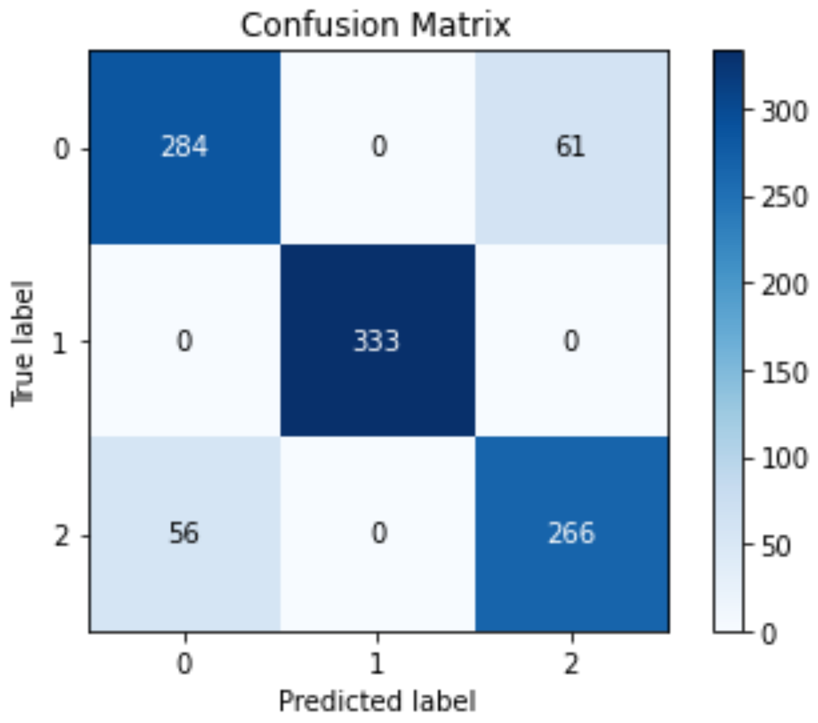


```
# For Q3
print('Testing acc: ', logistic_reg.evaluate(X_test, y_test))
✓ 0.2s

Testing acc:  0.883
```

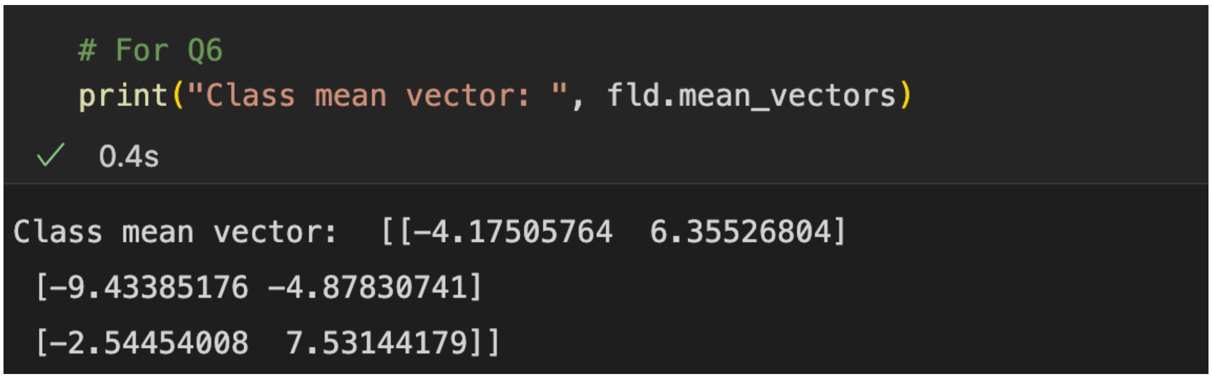
4. (5%) Plot the learning curve of the training. (x-axis=epoch, y-axis=loss)



5. (5%) Show the confusion matrix on testing data.

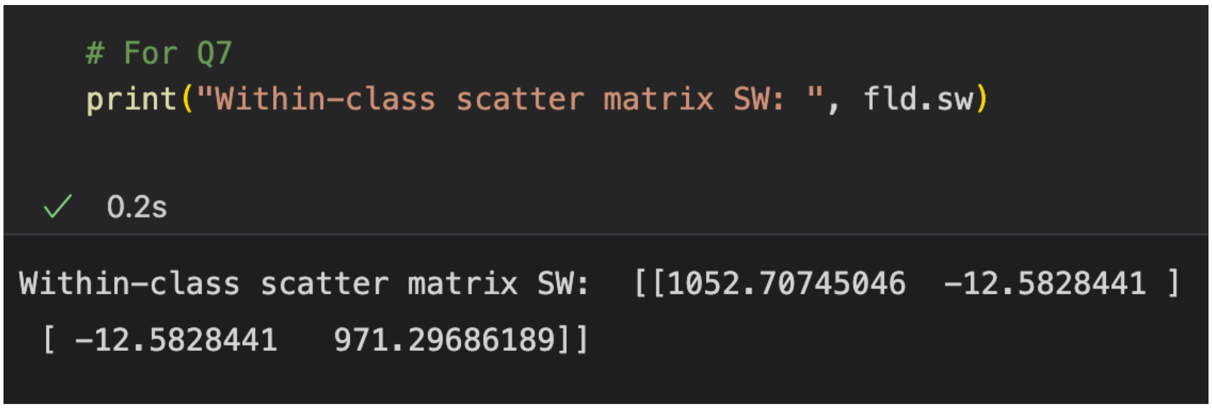


6. (2%) Compute the mean vectors mi (i=1, 2, 3) of each class on training data.



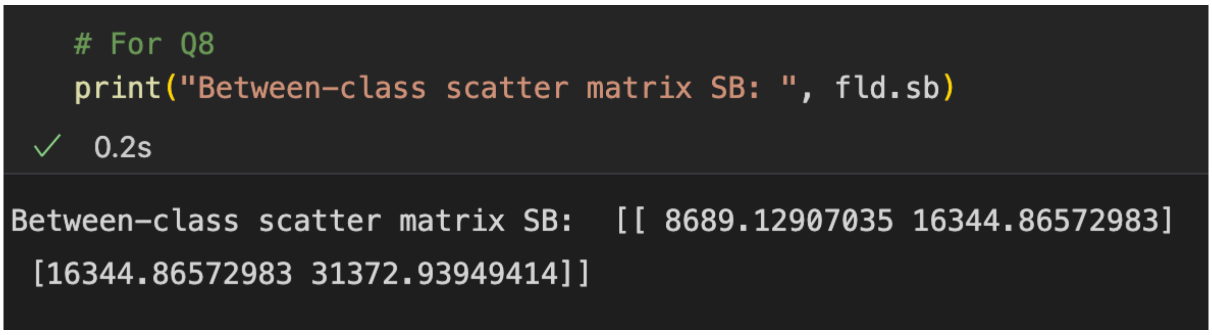
```
# For Q6
print("Class mean vector: ", fld.mean_vectors)
✓ 0.4s

Class mean vector:  [[-4.17505764  6.35526804]
 [-9.43385176 -4.87830741]
 [-2.54454008  7.53144179]]
```

7. (2%) Compute the within-class scatter matrix S_W on training data.



```
# For Q7
print("Within-class scatter matrix SW: ", fld.sw)

✓ 0.2s

Within-class scatter matrix SW:  [[1052.70745046  -12.5828441 ]
 [ -12.5828441   971.29686189]]
```

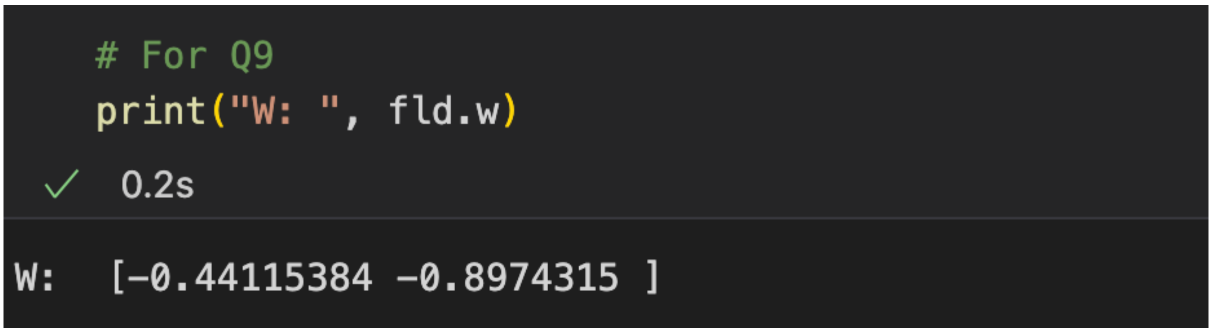
8. (2%) Compute the between-class scatter matrix S_B on training data.



```
# For Q8
print("Between-class scatter matrix SB: ", fld.sb)
```

```
Between-class scatter matrix SB:  [[ 8689.12907035 16344.86572983]
 [16344.86572983 31372.93949414]]
```

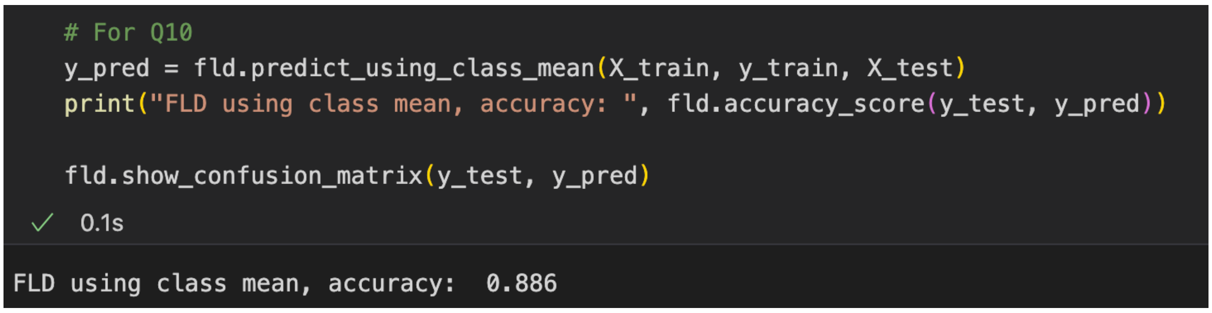
9. (4%) Compute the Fisher's linear discriminant w on training data.



```
# For Q9
print("W: ", fld.w)
```

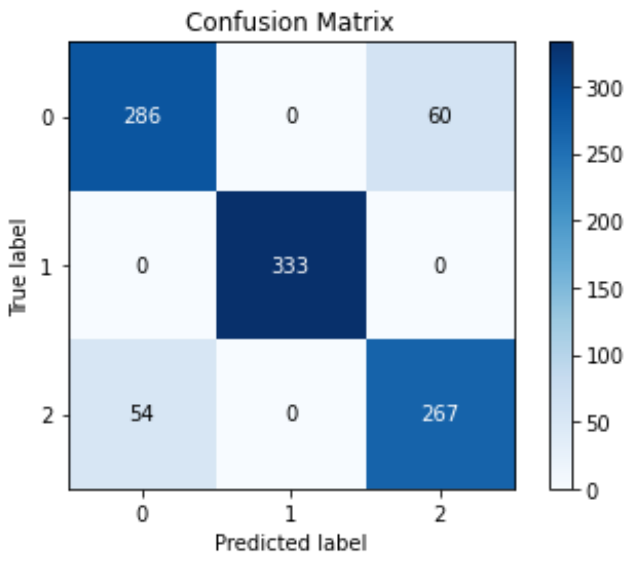
```
W:  [-0.44115384 -0.8974315 ]
```

10. (8%) Project the testing data to get the prediction using the shortest distance to the class mean. Report the accuracy score and draw the confusion matrix on testing data.

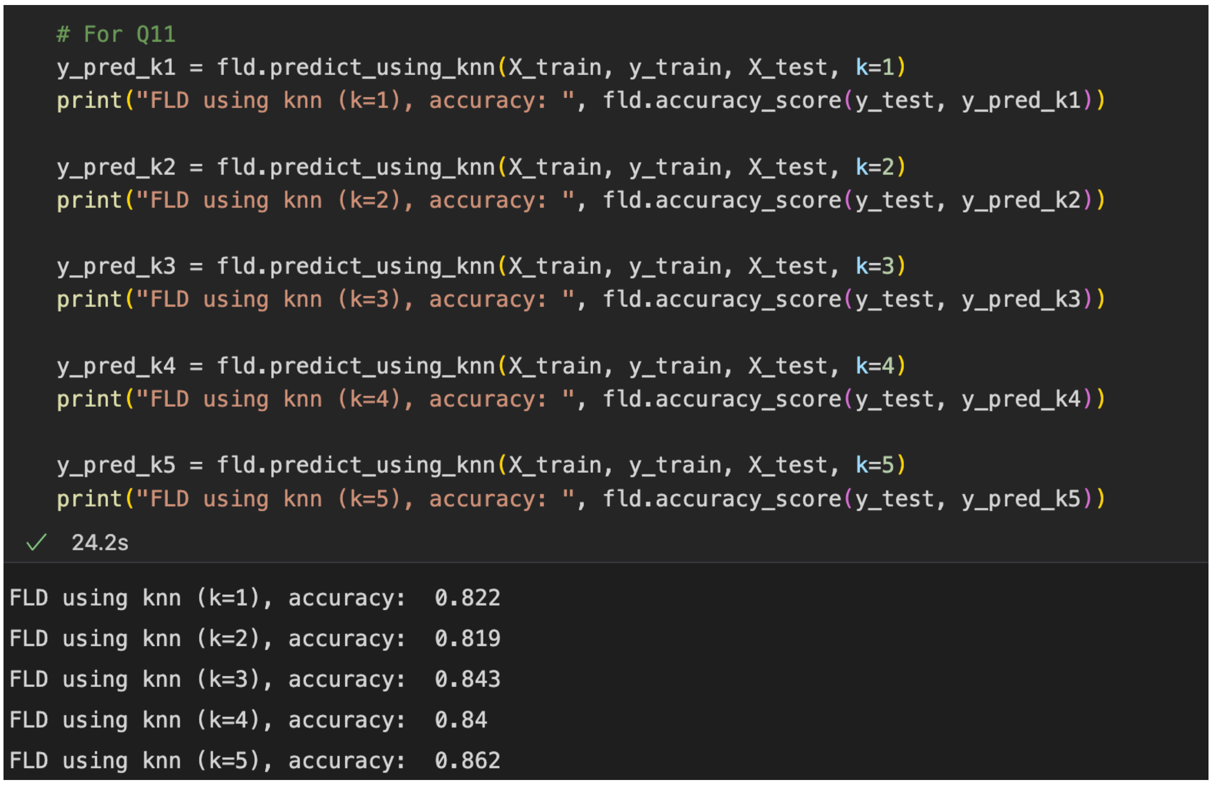


```
# For Q10
y_pred = fld.predict_using_class_mean(X_train, y_train, X_test)
print("FLD using class mean, accuracy: ", fld.accuracy_score(y_test, y_pred))

fld.show_confusion_matrix(y_test, y_pred)
```

```
FLD using class mean, accuracy:  0.886
```



11. (8%) Project the testing data to get the prediction using K-Nearest-Neighbor. Compare the accuracy score on the testing data with K values from 1 to 5.

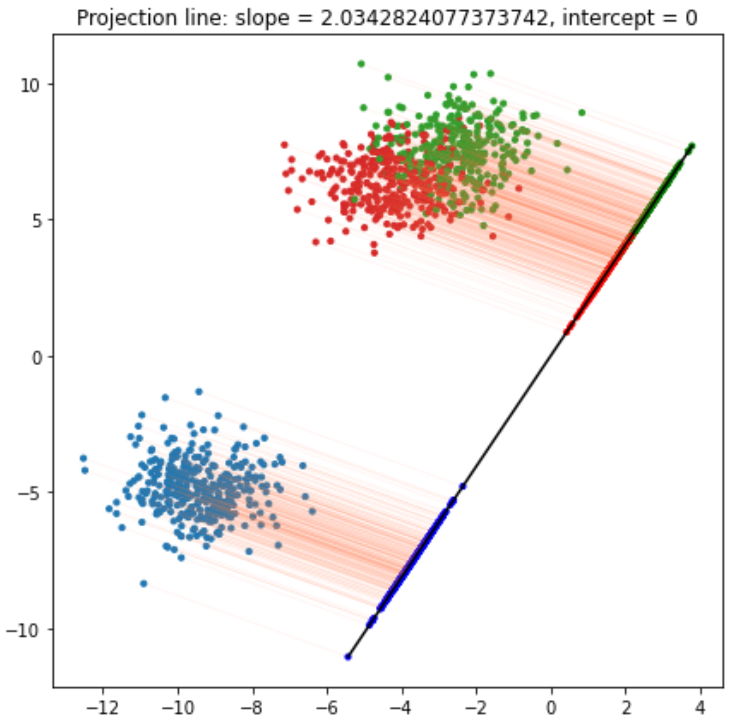


```
# For Q11
y_pred_k1 = fld.predict_using_knn(X_train, y_train, X_test, k=1)
print("FLD using knn (k=1), accuracy: ", fld.accuracy_score(y_test, y_pred_k1))

y_pred_k2 = fld.predict_using_knn(X_train, y_train, X_test, k=2)
print("FLD using knn (k=2), accuracy: ", fld.accuracy_score(y_test, y_pred_k2))

y_pred_k3 = fld.predict_using_knn(X_train, y_train, X_test, k=3)
print("FLD using knn (k=3), accuracy: ", fld.accuracy_score(y_test, y_pred_k3))

y_pred_k4 = fld.predict_using_knn(X_train, y_train, X_test, k=4)
print("FLD using knn (k=4), accuracy: ", fld.accuracy_score(y_test, y_pred_k4))

y_pred_k5 = fld.predict_using_knn(X_train, y_train, X_test, k=5)
print("FLD using knn (k=5), accuracy: ", fld.accuracy_score(y_test, y_pred_k5))
```

✓ 24.2s

```
FLD using knn (k=1), accuracy:  0.822
FLD using knn (k=2), accuracy:  0.819
FLD using knn (k=3), accuracy:  0.843
FLD using knn (k=4), accuracy:  0.84
FLD using knn (k=5), accuracy:  0.862
```

12. (4%)
1) Plot the best projection line on the training data and show the slope and intercept on the title **(you can choose any value of intercept for better visualization)**
2)colorize the training data with each class
3) project all training data points on your projection line. Your result should look like the below image (This image is for reference, not the answer)



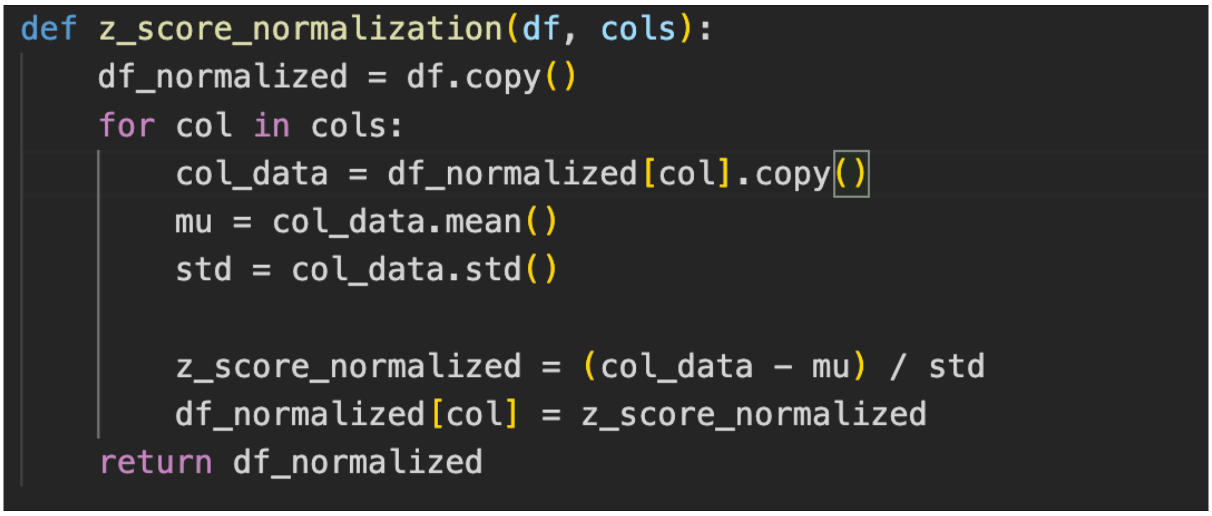
13. Explain how you chose your model and what feature processing you have done in detail. Otherwise, no points will be given.

    I use logistic classification as my model because it has a better performance in previous classification tasks and I think it has more learning ability and space than FDL.

    I first used the original data as the training data, after getting only about 60% correct rate, I chose to use the polynomial expansion method similar to the last HW1, after making the data z-score normalized, I expanded all the four feature. And then adjusting the batchsize to the maximum. After fintuning the epoch and the lr ,the result is a validation acc of more than 0.91.

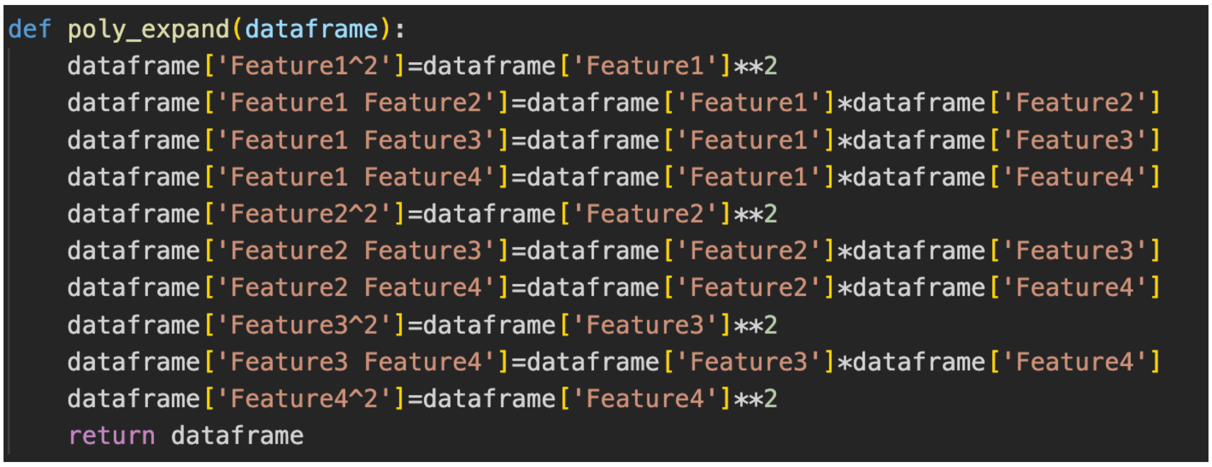
    Part of code is shown below:

    z-score normalization:



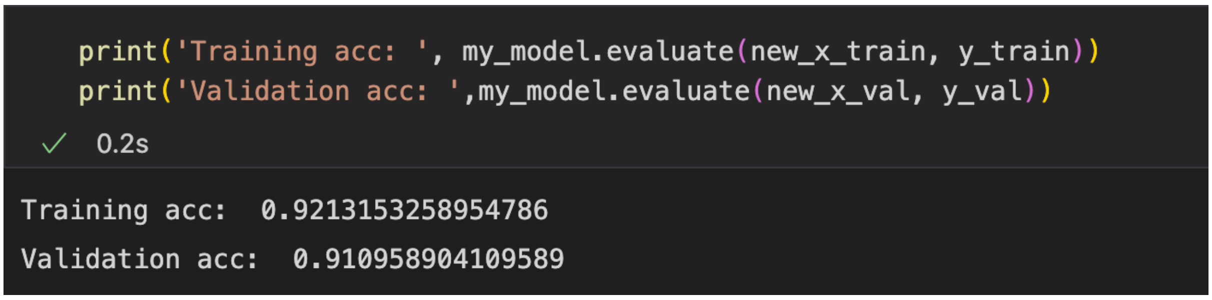
```python
def z_score_normalization(df, cols):
    df_normalized = df.copy()
    for col in cols:
        col_data = df_normalized[col].copy()
        mu = col_data.mean()
        std = col_data.std()

        z_score_normalized = (col_data - mu) / std
        df_normalized[col] = z_score_normalized
    return df_normalized
```

    Polynomial expansion:



```python
def poly_expand(dataframe):
    dataframe['Feature1^2']=dataframe['Feature1']**2
    dataframe['Feature1 Feature2']=dataframe['Feature1']*dataframe['Feature2']
    dataframe['Feature1 Feature3']=dataframe['Feature1']*dataframe['Feature3']
    dataframe['Feature1 Feature4']=dataframe['Feature1']*dataframe['Feature4']
    dataframe['Feature2^2']=dataframe['Feature2']**2
    dataframe['Feature2 Feature3']=dataframe['Feature2']*dataframe['Feature3']
    dataframe['Feature2 Feature4']=dataframe['Feature2']*dataframe['Feature4']
    dataframe['Feature3^2']=dataframe['Feature3']**2
    dataframe['Feature3 Feature4']=dataframe['Feature3']*dataframe['Feature4']
    dataframe['Feature4^2']=dataframe['Feature4']**2
    return dataframe
```

Final result:



```
print('Training acc: ', my_model.evaluate(new_x_train, y_train))
print('Validation acc: ',my_model.evaluate(new_x_val, y_val))
✓ 0.2s

Training acc:  0.9213153258954786
Validation acc:  0.910958904109589
```

Part. 2, Questions (30%):

(6%) 1. Discuss and analyze the performance

a) between Q10 and Q11, which approach is more suitable for this dataset. Why?

b) between different values of k in Q11. (Which is better, a larger or smaller k? Does this always hold?)

a)

From the results, it seems that shortest distance to class mean performs better. This may be because the data itself is already quite separate and has few overlap, considering the neighbors rather affects the classification performance.

In general, the shortest distance to class mean approach works well when the classes are well separated and have distinct means. However, if the classes overlap or have complex boundaries, this approach may not perform well. On the other hand, the K-Nearest-Neighbor approach can handle complex boundaries and overlapping classes, but it requires more computational resources and may not perform well if the number of training samples is large.

b)

From the results it seems that accuracy rate increases with increasing k value, which shows that the distribution of this dataset is sufficiently separated so that taking more neighbors into consideration can assist in the classification process to increase the performance.

The optimal value of k depends on the specific dataset and the problem at hand. In general, a larger value of k can lead to better performance if the classes are well separated and there is little noise in the data. However, if the classes overlap or there is significant noise, a smaller value of k may be more appropriate. It is important to note that the optimal value of k may not always be the same for all datasets, and it may need to be tuned using techniques such as cross-validation.

(6%) 2. Compare the sigmoid function and softmax function.

The sigmoid function and the softmax function are both commonly used activation functions in neural networks.

The sigmoid function is a non-linear activation function that maps any real-valued number to a value between 0 and 1. It is defined as:

$$\sigma(x) = 1 / (1 + e^{(-x)})$$

where x is the input to the function. The sigmoid function is useful for binary classification problems, where the output is a probability between 0 and 1, indicating the likelihood of the input belonging to a particular class.
On the other hand, the softmax function is also a non-linear activation function that maps any vector of real numbers to a probability distribution over the same set of numbers. It is defined as:

$$\text{softmax}(x_i) = e^{(x_i)} / \sum e^{(x_j)}$$

where x_i is the i_{th} element of the input vector x. The softmax function is commonly used for multi-class classification problems, where the output is a probability distribution over a set of classes.
In summary, while the sigmoid function is useful for binary classification problems, the softmax function is useful for multi-class classification problems. The sigmoid function maps to a single probability between 0 and 1, while the softmax function maps to a probability distribution over a set of numbers.

(6%) 3. Why do we use cross entropy for classification tasks and mean square error for regression tasks?

Cross-entropy is typically used for classification tasks because it measures the difference between two probability distributions. In classification tasks, the goal is to predict a probability distribution over a set of classes for each input. Cross-entropy loss compares the predicted probability distribution to the true probability distribution and penalizes the model for making incorrect predictions. The cross-entropy loss is high when the predicted probabilities are far from the true probabilities and low when they are close to each other.
On the other hand, MSE is typically used for regression tasks because it measures the average squared difference between the predicted values and the true values. In regression tasks, the goal is to predict a continuous value for each input. MSE loss penalizes the model for making large errors, as the squared term magnifies the difference between the predicted and true values. The MSE loss is high when the predicted values are far from the true values and low when they are close to each other.

(6%) 4. In Q13, we provide an imbalanced dataset. Are there any methods to improve Fisher Linear Discriminant's performance in handling such datasets?

One common approach is to use class weights to balance the contribution of each class to the loss function.

Class weights can be used to assign higher weights to the minority class and lower weights to the majority class, which can help the model learn to better distinguish between the classes. The weights can be used to adjust the loss function used in FLD to be more sensitive to the minority class.

Another approach is to use data augmentation techniques to increase the size of the minority class. This can be done by artificially creating new data points in the minority class by applying transformations such as rotation, translation, or scaling. This can help to balance the number of samples in each class, which can improve the model's performance.

Additionally, resampling techniques such as oversampling or undersampling can also be used to balance the dataset. Oversampling involves replicating the minority class samples, while undersampling involves removing samples from the majority class. Both techniques can be used to adjust the class distribution in the dataset and improve the performance of FLD.

In summary, class weighting, data augmentation, and resampling techniques can all be used to improve FLD's performance in handling imbalanced datasets. The choice of which technique to use depends on the specific characteristics of the dataset and the goals of the analysis.

(6%) 5. Calculate the results of the partial derivatives for the following equations. (The first one is binary cross-entropy loss, and the second one is mean square error loss followed by a sigmoid function.)

$$\frac{\partial}{\partial x}\underbrace{\left(y * \ln(\sigma(x)) + (1-y) * \ln(1-\sigma(x))\right)}_{}\ L$$

$$\Rightarrow \frac{\partial L}{\partial x} = \frac{\partial L}{\partial \sigma(x)} * \frac{\partial \sigma(x)}{\partial x}$$

① $$\frac{\partial L}{\partial \sigma(x)} = \frac{\partial}{\partial \sigma(x)}\left[y * \ln \sigma(x) + (1-y) * \ln(1-\sigma(x)) \right]$$

$$= \frac{y}{\sigma(x)} - \frac{1-y}{1-\sigma(x)}$$

② 若 $\sigma(x)$ 為 sigmoid function，則

$$\sigma(x) = (1+e^{-x})^{-1} \Rightarrow \sigma'(x) = (-1)^2(1-e^{-x})^{-2}e^{-x}$$

$$= \frac{e^{-x}}{(1-e^{-x})^2}$$

Ans: $$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial \sigma(x)}\frac{\partial \sigma(x)}{\partial x}$$

$$= \left[\frac{y}{\sigma(x)} - \frac{1-y}{1-\sigma(x)}\right]\left[\frac{e^{-x}}{(1-e^{-x})^2}\right]$$ #

其中 $\sigma(x)$ 為 sigmoid function

$$\frac{\partial}{\partial x}\Big(\underbrace{\left(y-\sigma(x)\right)^2}_{L}\Big)$$

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial \sigma(x)} \frac{\partial \sigma(x)}{\partial x}$$

$$\frac{\partial L}{\partial \sigma(x)} = (-1)\cdot 2 \cdot [y-\sigma(x)]$$

$$= -2[y-\sigma(x)]$$

$$\frac{\partial \sigma(x)}{\partial x} = \frac{e^{-x}}{(1-e^{-x})^2}$$

$$\Rightarrow \frac{\partial L}{\partial x} = -2[y-\sigma(x)]\frac{e^{-x}}{(1-e^{-x})^2} \#$$

$\sigma(x)$ 為 sigmoid function