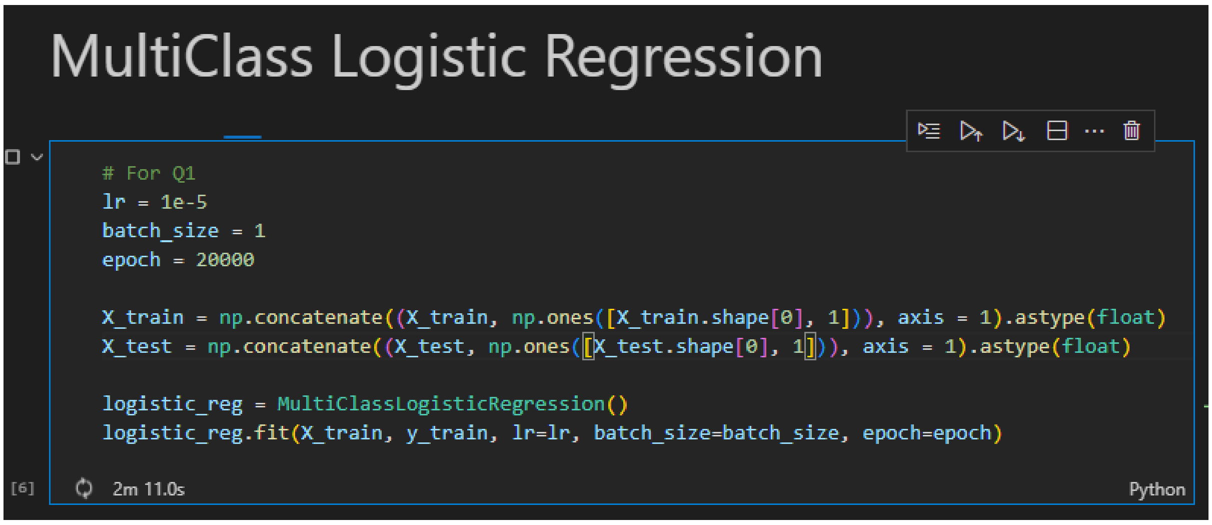
NYCU Pattern Recognition, Homework 2

0810749, 張君實

Part. 1, Coding (70%):

Logistic Regression Model

1. **(0%) Show the learning rate, epoch, and batch size that you used.**
   learning rate: 1e-5
   epoch: 20000
   batch size: 1

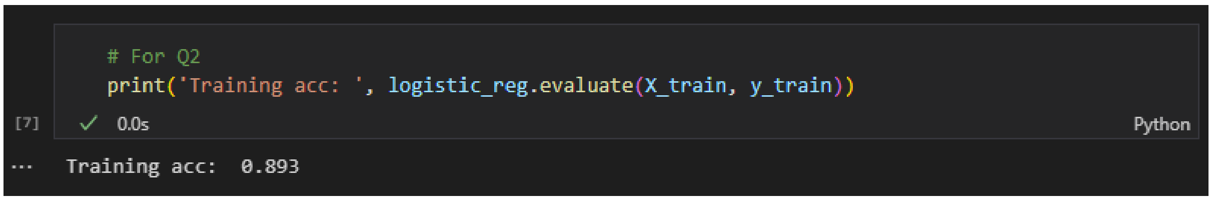


MultiClass Logistic Regression

```
# For Q1
lr = 1e-5
batch_size = 1
epoch = 20000

X_train = np.concatenate((X_train, np.ones([X_train.shape[0], 1])), axis = 1).astype(float)
X_test = np.concatenate((X_test, np.ones([X_test.shape[0], 1])), axis = 1).astype(float)

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

[6] 2m 11.0s Python

2. **(5%) What's your training accuracy?**
   Training accuracy: 0.893

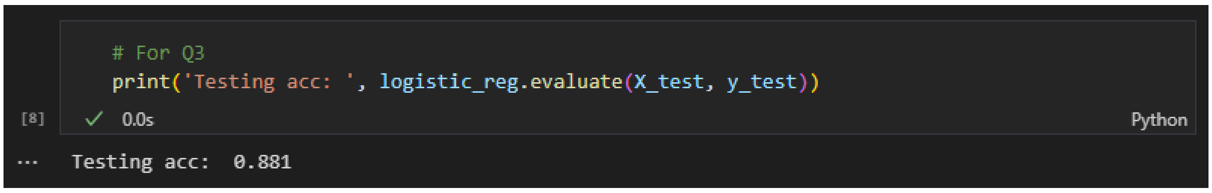


```
# For Q2
print('Training acc: ', logistic_reg.evaluate(X_train, y_train))
```

[7] ✓ 0.0s Python

```
... Training acc:  0.893
```

3. **(5%) What's your testing accuracy?**
   Testing accuracy: 0.881

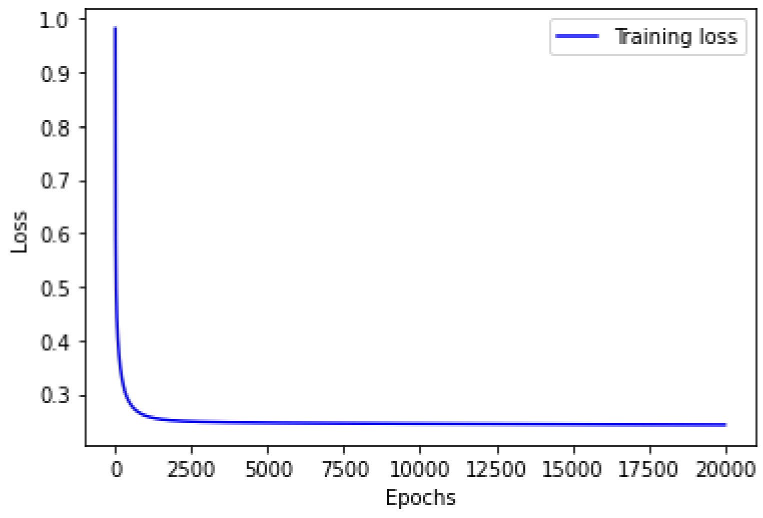


```
# For Q3
print('Testing acc: ', logistic_reg.evaluate(X_test, y_test))
```

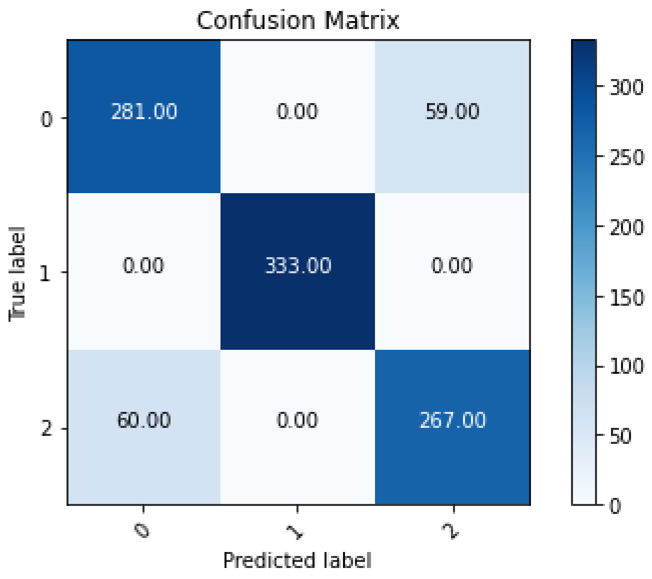
[8] ✓ 0.0s Python

```
... Testing acc:  0.881
```

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



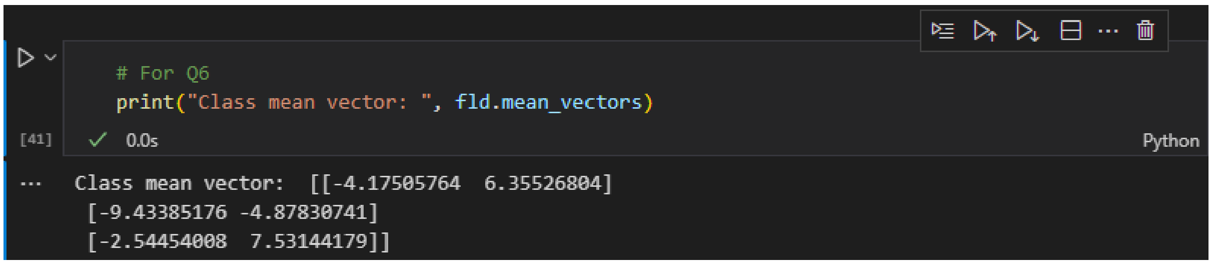
5. **(5%) Show the confusion matrix on testing data.**



Fisher's Linear Discriminant (FLD) Model

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

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

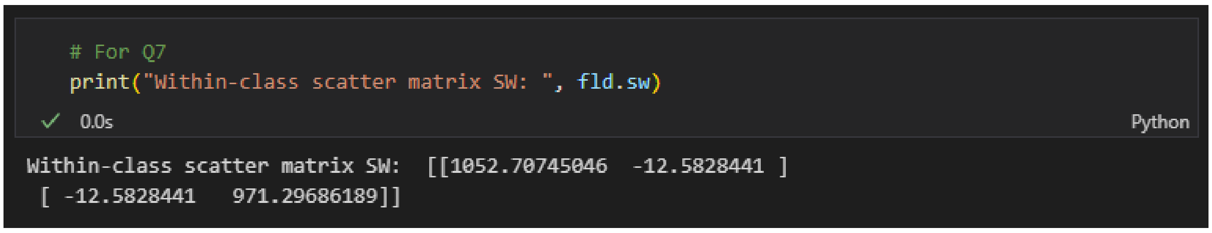


```
# For Q6
print("Class mean vector: ", fld.mean_vectors)
```

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

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

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

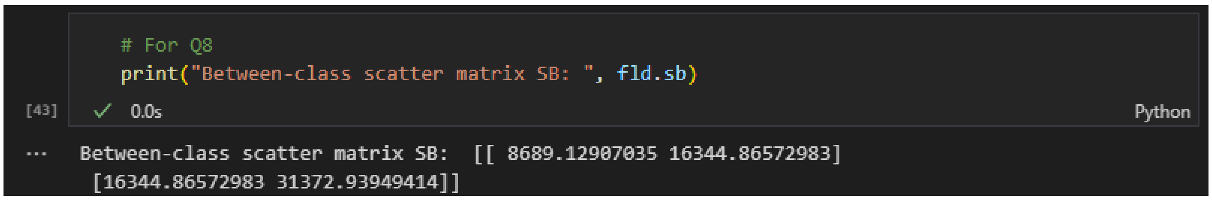


```
# For Q7
print("Within-class scatter matrix SW: ", fld.sw)
```

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

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

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

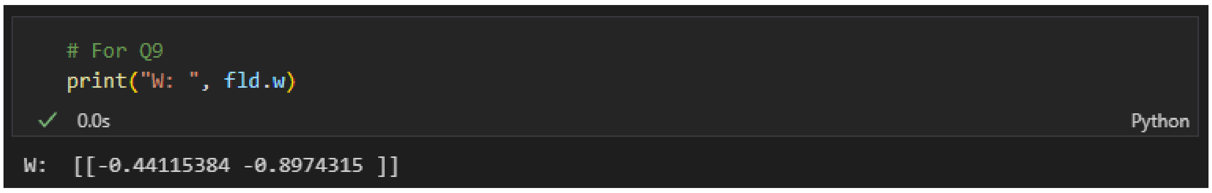


```
# For Q8
print("Between-class scatter matrix SB: ", fld.sb)
```

```
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.**

W: [[-0.44115384 -0.8974315]]

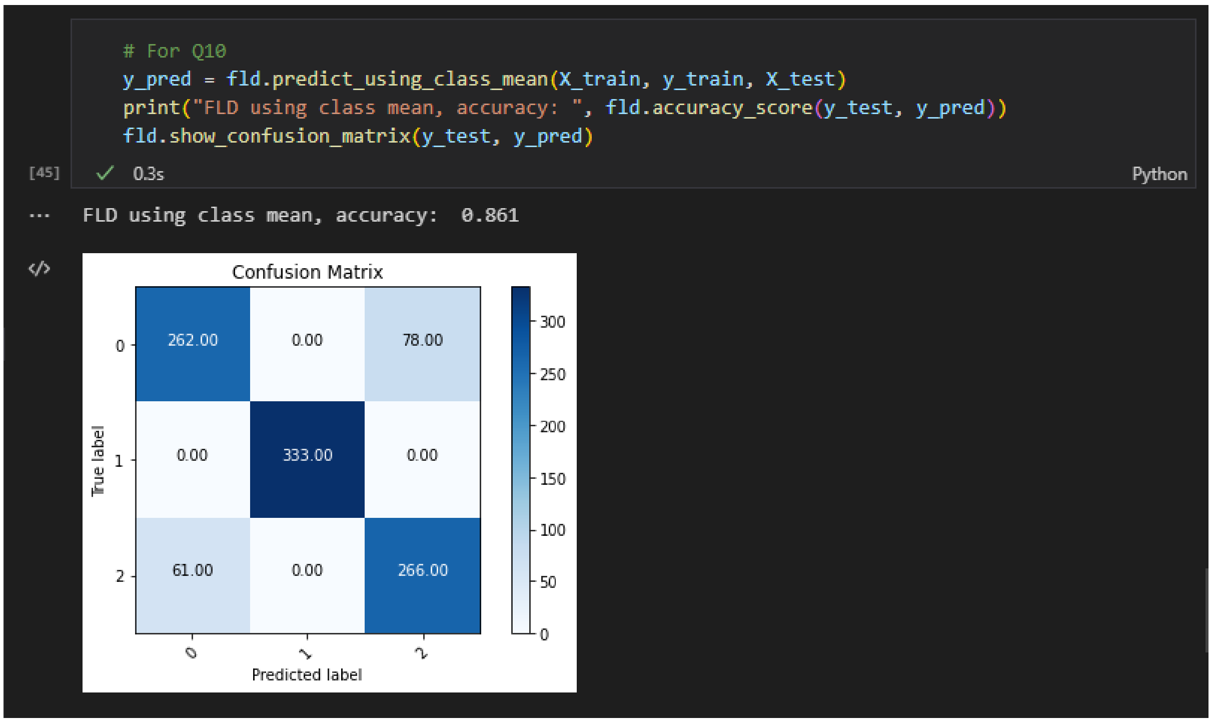


```
# For Q9
print("W: ", fld.w)
```

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

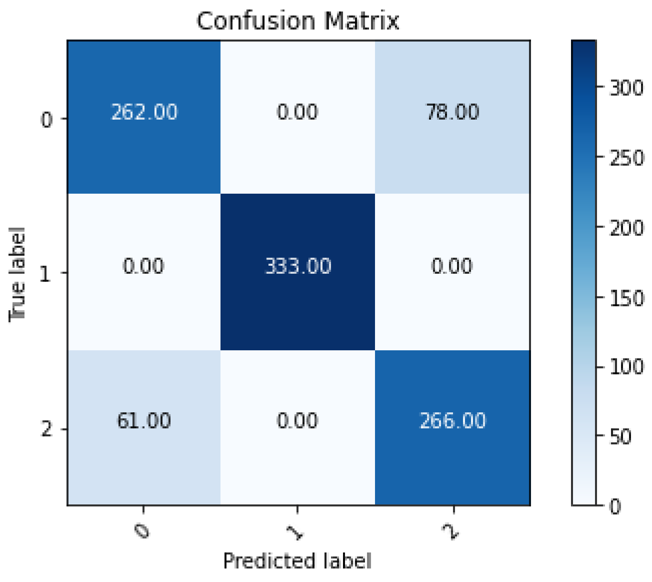
10. **(8%) Project the <u>testing data</u> to get the prediction using the shortest distance to the class mean. Report the accuracy score and draw the confusion matrix on <u>testing data</u>.**

FLD using class mean, accuracy: 0.861



```
# 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:  0.861
```

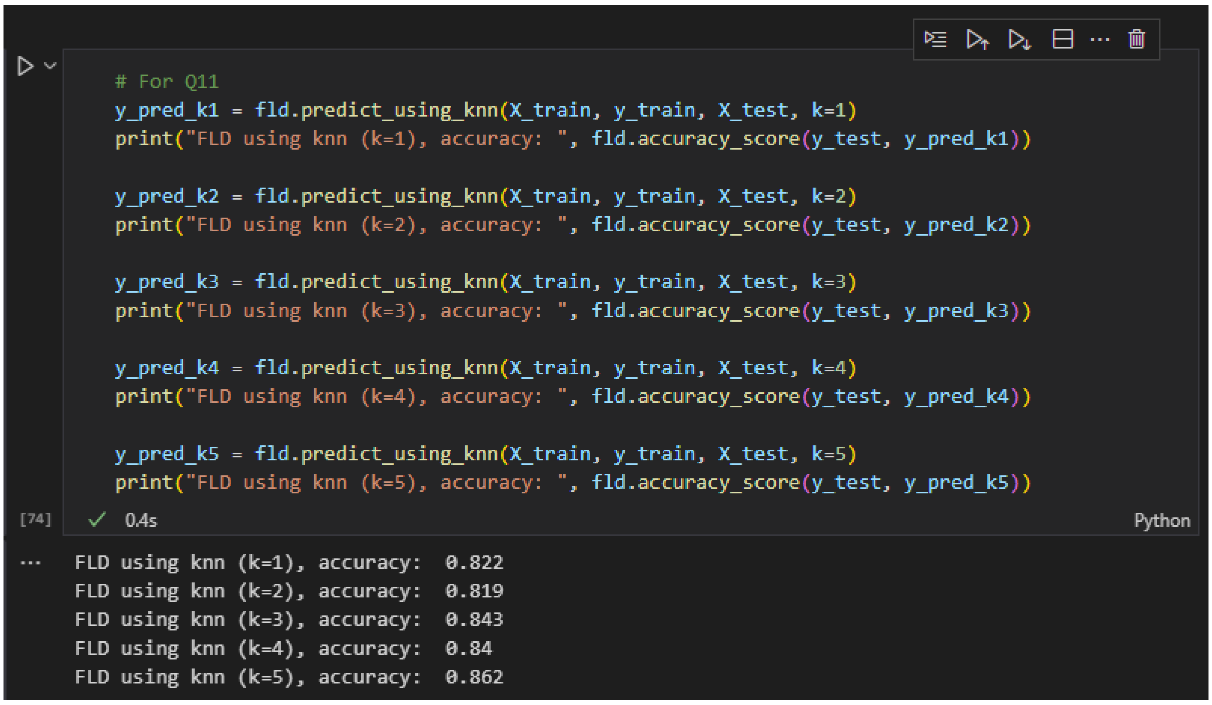


	0	1	2
0	262.00	0.00	78.00
1	0.00	333.00	0.00
2	61.00	0.00	266.00

11. (5%) 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.

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



```
# 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))
```

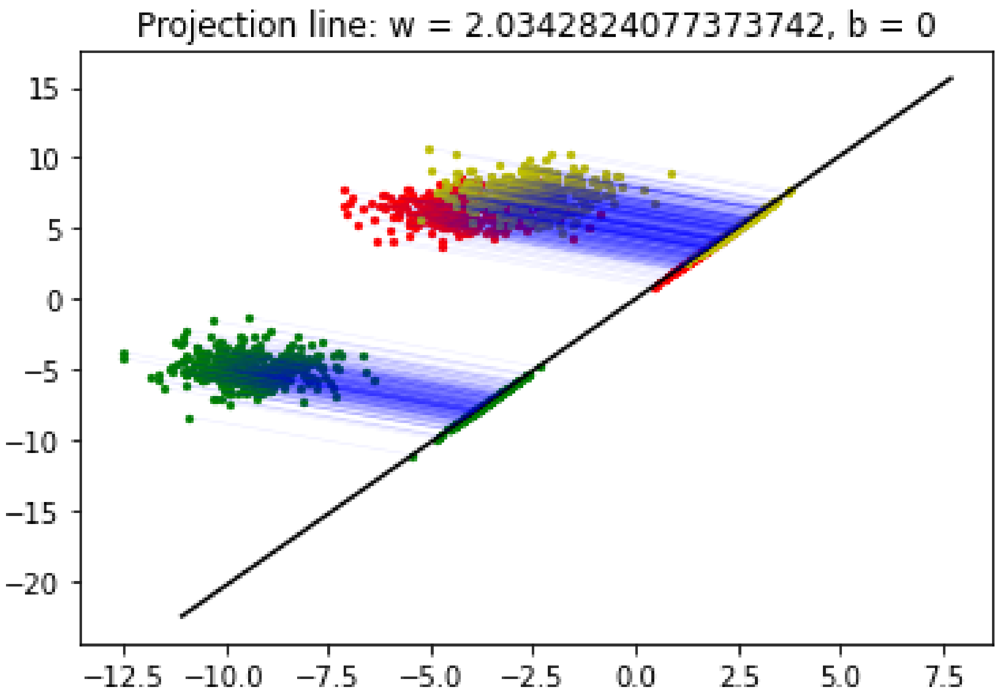
```
FLD using knn (k=1), accuracy:  0.822
FLD using knn (k=2), accuracy:  0.819
FLD using knn (k=3), accuracy:  0.843
FLD using knn (k=4), accuracy:  0.84
FLD using knn (k=5), accuracy:  0.862
```

12. (4%)

1) Plot the best projection line on the <u>training data</u> and <u>show the slope and intercept on the title</u> ***(you can choose any value of intercept for better visualization)***

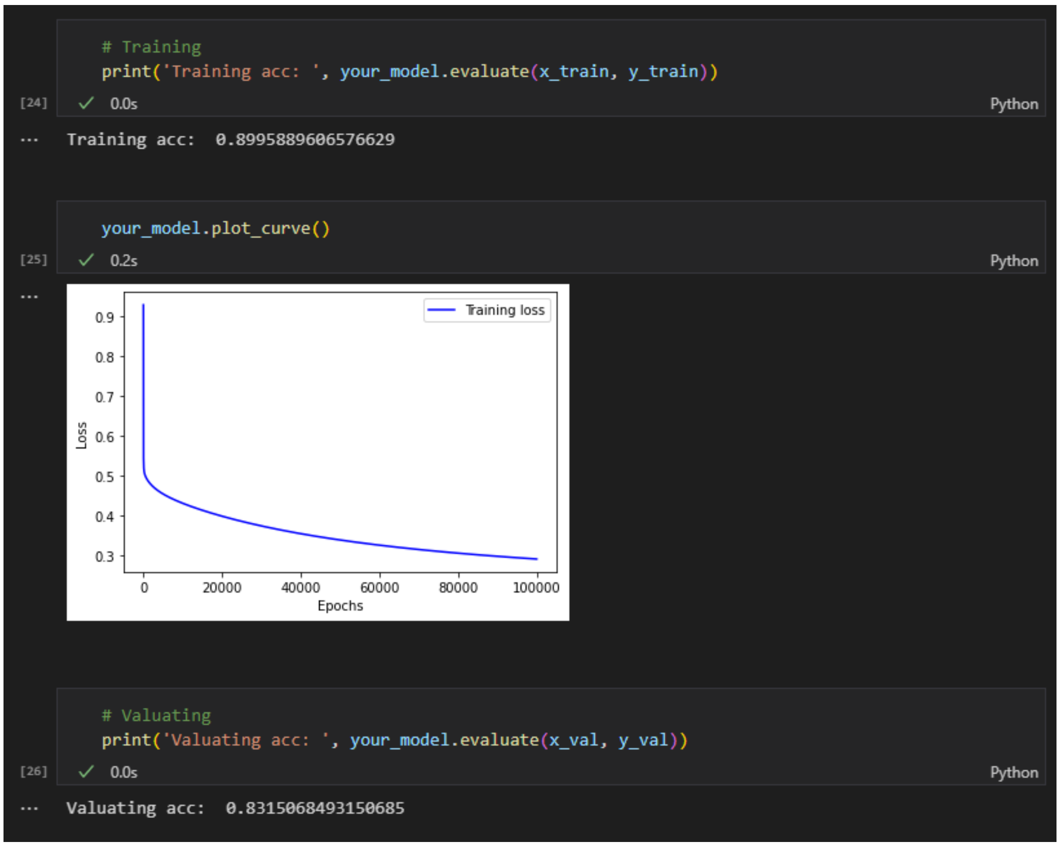
2) colorize the training data with each class

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 eventually used the FLD model with KNN algorithm as my model. It is because the performance of a regression model is poor in this dataset, in which 2 classes overlap severely. The evaluation acc even can not achieve 85% accuracy.

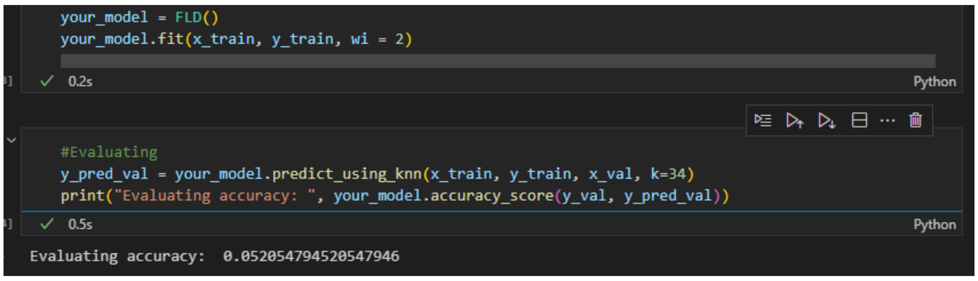


```
[24]
# Training
print('Training acc: ', your_model.evaluate(x_train, y_train))
✓ 0.0s                                                                 Python

Training acc:  0.8995889606576629
```

```
[25]
your_model.plot_curve()
✓ 0.2s                                                                 Python
```

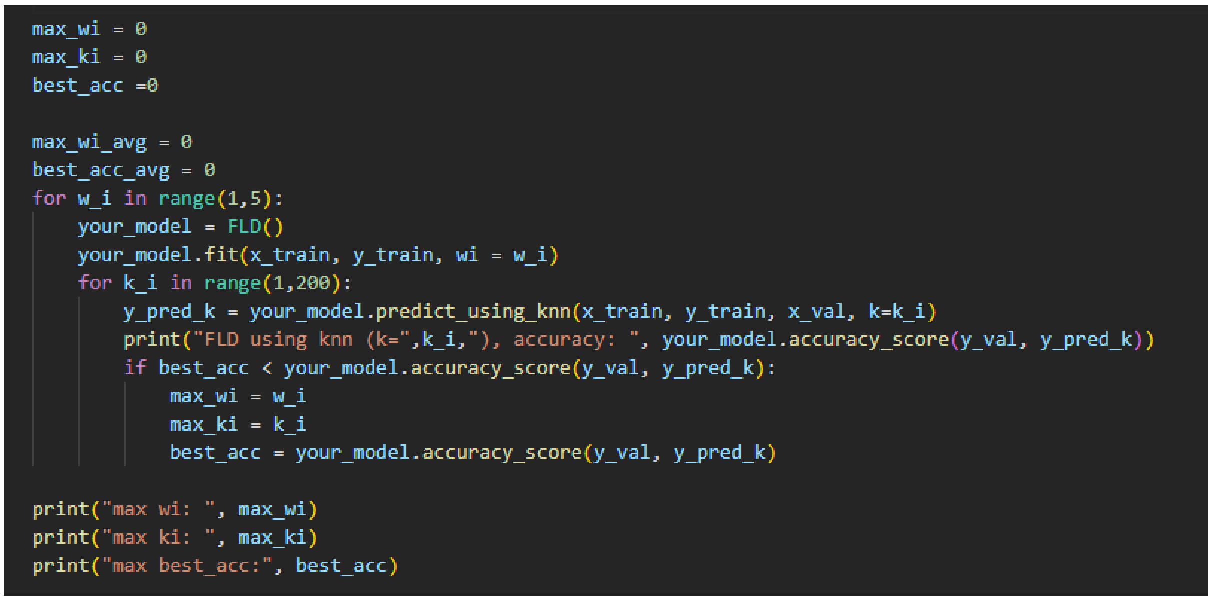
```
[26]
# Valuating
print('Valuating acc: ', your_model.evaluate(x_val, y_val))
✓ 0.0s                                                                 Python

Valuating acc:  0.8315068493150685
```

As for the feature processing, I don't do anything. It is because when I add new features in the dataset, it would make the model overfitting, which causes a poor performance on predicting.

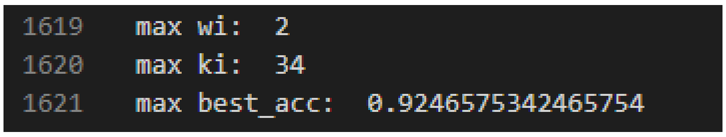


```
your_model = FLD()
your_model.fit(x_train, y_train, wi = 2)
✓ 0.2s                                                                 Python

#Evaluating
y_pred_val = your_model.predict_using_knn(x_train, y_train, x_val, k=34)
print("Evaluating accuracy: ", your_model.accuracy_score(y_val, y_pred_val))
✓ 0.5s                                                                 Python

Evaluating accuracy:  0.952054794520547946
```

Last but not least, I try to adjust the number of eigenvectors in FLD, and adjust the number of the k in KNN. Looking at the following picture, I swept a lot of the hyperparameter combinations. At last, I found that when the number of eigenvectors =2 , and k =34, the accuracy can achieve 92.46%, so I chose these hyperparameters for my model.



```python
max_wi = 0
max_ki = 0
best_acc =0

max_wi_avg = 0
best_acc_avg = 0
for w_i in range(1,5):
    your_model = FLD()
    your_model.fit(x_train, y_train, wi = w_i)
    for k_i in range(1,200):
        y_pred_k = your_model.predict_using_knn(x_train, y_train, x_val, k=k_i)
        print("FLD using knn (k=",k_i,"), accuracy: ", your_model.accuracy_score(y_val, y_pred_k))
        if best_acc < your_model.accuracy_score(y_val, y_pred_k):
            max_wi = w_i
            max_ki = k_i
            best_acc = your_model.accuracy_score(y_val, y_pred_k)

print("max wi: ", max_wi)
print("max ki: ", max_ki)
print("max best_acc:", best_acc)
```



```
max wi:  2
max ki:  34
max best_acc:  0.9246575342465754
```

Part. 2, Questions (30%):

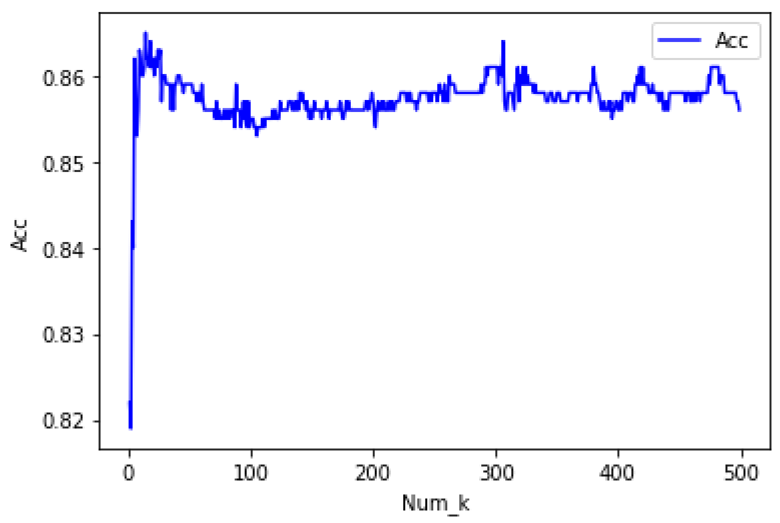
(6%) 1. Discuss and analyze the performance

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) According to our observation in Q10 and Q11, we could find that the performance of the Q10 and Q11 are almost the same. But in my viewpoint, I still think that KNN algorithm is better than shortest distance to the class mean in this dataset. In Q12, we could see that the red points are very close to the yellow points. Thus, when a prediction point is also close to the yellow points and red points, if we use KNN algorithm, the class of prediction points could be determined by the neighborhood points, and it is similar to the vote. Through the process of voting, KNN could deal with the condition of overlapping classes better. Thus, I think that KNN algorithm is slightly better than shortest distance to the class mean in this dataset.

(b) I tried to plot the different values k in the Q11. In the following picture, we could see the range of k is from 1 to 500, and we could notice that a larger k is better than a small k on average because the accuracy of a big k is over 85% . However, the best k in Q11 could not be too big or too small. In the Q11, we could see that the best k is 14, and the corresponding accuracy is 0.865. Thus, when we try to tune the k, we should try to find a k, which is not too big or too small.



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

The sigmoid function is often used for the binary classification task, and the softmax function is often used for the multiple classification task.

The formula of the sigmoid function is

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

The formula of the softmax function is

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

If we use the softmax function in binary classification tasks, the softmax will become the sigmoid function. In other words, sigmoid function is a case of softmax function when class number is 2.

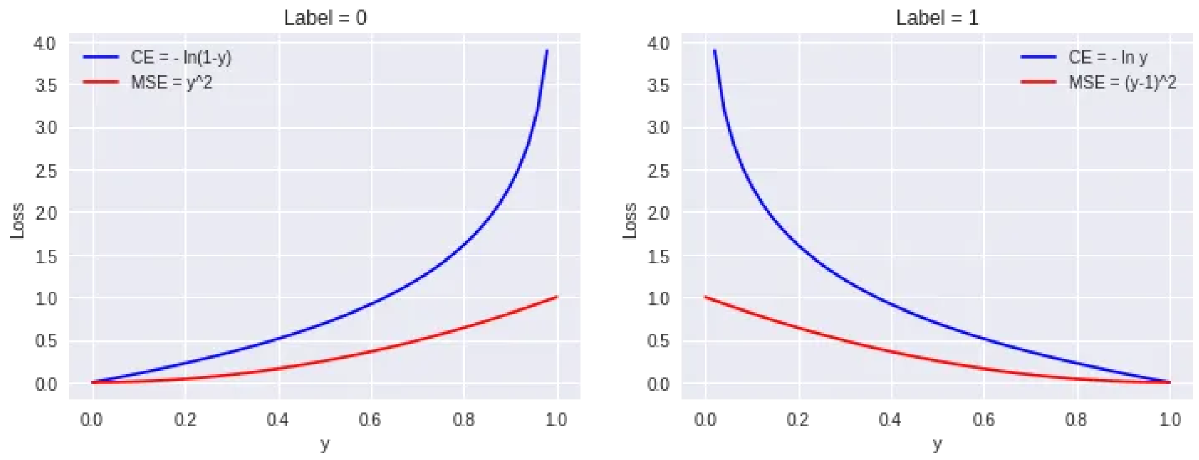
$$\Pr(Y_i = 0) = \frac{e^{\beta_0 \cdot \mathbf{X}_i}}{\sum_{0\le c\le K} e^{\beta_c \cdot \mathbf{X}_i}} = \frac{e^{\beta_0 \cdot \mathbf{X}_i}}{e^{\beta_0 \cdot \mathbf{X}_i} + e^{\beta_1 \cdot \mathbf{X}_i}} = \frac{e^{(\beta_0 - \beta_1) \cdot \mathbf{X}_i}}{e^{(\beta_0 - \beta_1) \cdot \mathbf{X}_i} + 1} = \frac{e^{-\beta \cdot \mathbf{X}_i}}{1 + e^{-\beta \cdot \mathbf{X}_i}}$$

$$\Pr(Y_i = 1) = \frac{e^{\beta_1 \cdot \mathbf{X}_i}}{\sum_{0\le c\le K} e^{\beta_c \cdot \mathbf{X}_i}} = \frac{e^{\beta_1 \cdot \mathbf{X}_i}}{e^{\beta_0 \cdot \mathbf{X}_i} + e^{\beta_1 \cdot \mathbf{X}_i}} = \frac{1}{e^{(\beta_0 - \beta_1) \cdot \mathbf{X}_i} + 1} = \frac{1}{1 + e^{-\beta \cdot \mathbf{X}_i}}$$

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

Cross entropy is a loss function that compares the similarity of 2 probability distributions, and the range is from 0 to 1. We don't use cross entropy in regression tasks because the predicted value is not always in the interval [0,1]. We use cross entropy in classification tasks.

MSE is a loss function that compares the distance between the true value and predicted value. If we use MSE in classification tasks, it may be possible to make the model converge. However, looking at the following graph, we could find that when the model predicts the wrong class, CE could give a bigger loss to the model, which means the gradient of CE is bigger than MSE. Thus, we often use CE in classification tasks.



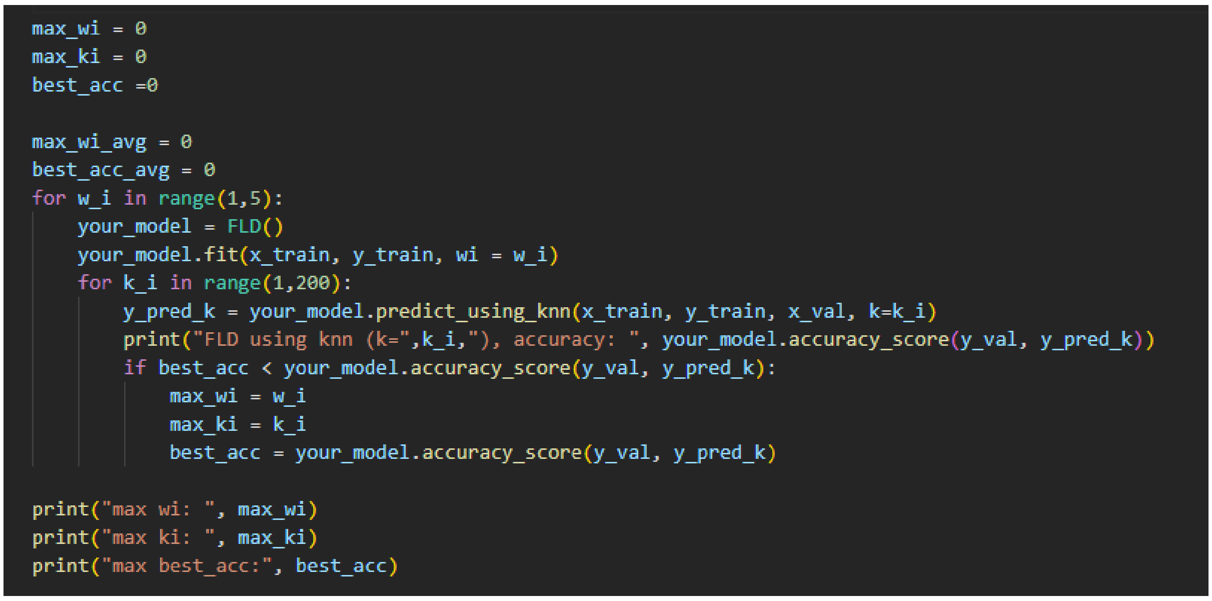
Additionally, in the classification tasks, when we use CE after the softmax function, the gradient could become a very simple formula. That is y_pred - y. This is another reason why we use CE in classification tasks.

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

Yes, we could adjust the number of eigenvectors in FLD. In this lab, I choose 2 eigenvectors in FLD.

Additionally, we also can change the number of k in KNN. In this lab, I choose k=34.

As for how to tune the number of eigenvectors and the k, we could try and error, and choose the best hyperparameter combinations on the validation dataset.



```
max_wi = 0
max_ki = 0
best_acc =0

max_wi_avg = 0
best_acc_avg = 0
for w_i in range(1,5):
    your_model = FLD()
    your_model.fit(x_train, y_train, wi = w_i)
    for k_i in range(1,200):
        y_pred_k = your_model.predict_using_knn(x_train, y_train, x_val, k=k_i)
        print("FLD using knn (k=",k_i,"), accuracy: ", your_model.accuracy_score(y_val, y_pred_k))
        if best_acc < your_model.accuracy_score(y_val, y_pred_k):
            max_wi = w_i
            max_ki = k_i
            best_acc = your_model.accuracy_score(y_val, y_pred_k)

print("max wi: ", max_wi)
print("max ki: ", max_ki)
print("max best_acc:", best_acc)
```

As for creating the new features or nonlinear features, it seems that it would make the model overfitting easily. Thus, I don't recommend adding new features in the dataset, although maybe it seems to be useful.

(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.)

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

$$\frac{\partial}{\partial x}\left(\big(y-\sigma(x)\big)^2\right)$$

If $6(x) = \frac{1}{1+e^{-x}}$

$$\frac{\partial\, 6(x)}{\partial x} = \frac{-e^{-x}}{-(1+e^{-x})^2} = \frac{e^{-x}}{(1+e^{-x})^2}$$

$$= \frac{1}{1+e^{-x}} \frac{e^{-x}}{1+e^{-x}}$$

$$= \frac{1}{1+e^{-x}} \left(1 - \frac{1}{1+e^{-x}}\right)$$

$$= 6(x)\left(1 - 6(x)\right)$$

$$\frac{\partial}{\partial x}\left(y \ln(6(x)) + (1-y)\ln(1-6(x))\right)$$

$$= \frac{6'(x)}{6(x)} y - (1-y)\frac{6'(x)}{1-6(x)}$$

$$= \frac{6'(x)}{6(x)(1-6(x))}\left(y(1-6(x)) - (1-y)6(x)\right)$$

$$= \frac{6'(x)}{6(x)(1-6(x))}(y - 6(x))$$

$$= \frac{6(x)(1-6(x))}{6(x)(1-6(x))}(y - 6(x))$$

If $6(x)$ is sigmoid function

$$= y - 6(x)$$

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

$$= -2\left(y-\sigma(x)\right)\sigma'(x)$$

$$= -2\left(y-\sigma(x)\right)\underbrace{\sigma(x)\left(1-\sigma(x)\right)}_{\uparrow}$$

If sigmoid function