MACHINE LEARNING AND DATA MINING 2

LABWORK 3: Classification I

GROUP MEMBERS:

Đào Hải Long BA9-041

Đặng Thái Sơn BA9-053

A/ K-nearest neighbor classification

I/ Iris dataset

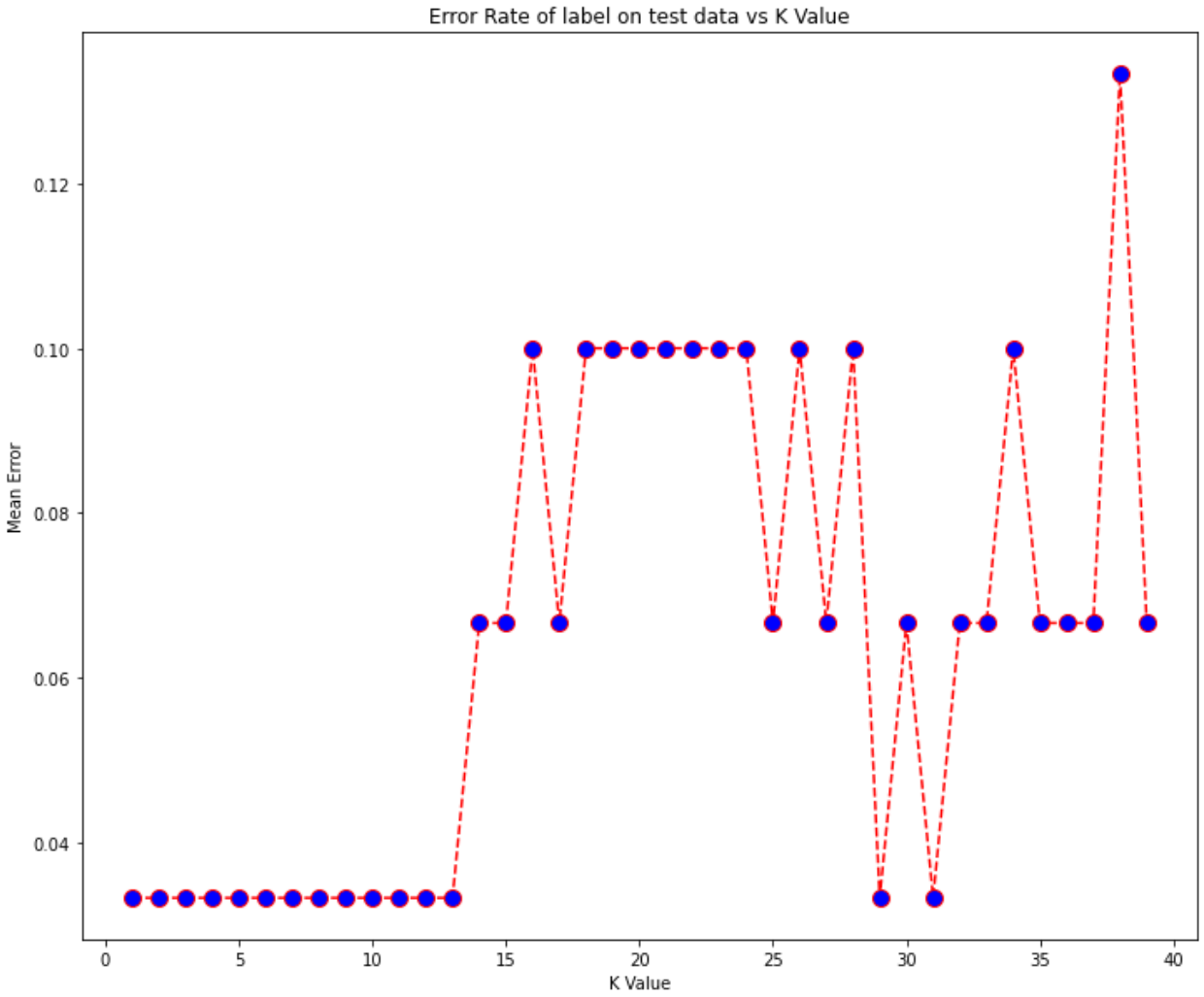
1. Implement KNN and calculate classification error

After implementing K-nearest neighbor (where K = 3) and testing on the Iris dataset, we get the following result with some important statistic such as the confusion matrix, precision, accuracy...

```
Confusion Matrix
 [[ 6  0  0]
 [ 0 13  0]
 [ 0  1 10]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       0.93      1.00      0.96        13
           2       1.00      0.91      0.95        11

    accuracy                           0.97        30
   macro avg       0.98      0.97      0.97        30
weighted avg       0.97      0.97      0.97        30
```

In terms of the classification error, we print out both the values and the graph:

```
[0.03333333333333333, 0.03333333333333333, 0.03333333333333333, 0.03333333333333333, 0.03333333333333333, 0.0333333333333333
3, 0.03333333333333333, 0.03333333333333333, 0.03333333333333333, 0.03333333333333333, 0.03333333333333333, 0.03333333333333
333, 0.03333333333333333, 0.06666666666666667, 0.06666666666666667, 0.1, 0.06666666666666667, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1,
0.1, 0.06666666666666667, 0.1, 0.06666666666666667, 0.1, 0.03333333333333333, 0.06666666666666667, 0.03333333333333333, 0.06
666666666666667, 0.06666666666666667, 0.1, 0.06666666666666667, 0.06666666666666667, 0.06666666666666667, 0.1333333333333333
3, 0.06666666666666667]
```



2. Vary the value of K

Applying the same K-nearest neighbor method with the increase in the K value of two units (K = 5), we get the following result:

```
Confusion Matrix
 [[10  0  0]
 [ 0  9  2]
 [ 0  0  9]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      0.82      0.90        11
           2       0.82      1.00      0.90         9

    accuracy                           0.93        30
   macro avg       0.94      0.94      0.93        30
weighted avg       0.95      0.93      0.93        30
```

Compared to the case where K = 3, the accuracy rate of this case is lower. To be more specific, when K = 3, the accuracy is 0.97, but it can only get the rate of 0.93 when K = 5.

3. Normalize the input dataset then use KNN

By normalizing the Iris dataset before implementing the KNN, the performance is much better: the accuracy can achieve the maximum:

```
Confusion Matrix
 [[10  0  0]
 [ 0 12  0]
 [ 0  0  8]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00        12
           2       1.00      1.00      1.00         8

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

4. Apply PCA then use KNN

Firstly, we apply the PCA with three components on the dataset:

```
array([[-2.26470281, -0.4800266 , -0.12770602],
       [-2.08096115,  0.67413356, -0.23460885],
       [-2.36422905,  0.34190802,  0.04420148],
       [-2.29938422,  0.59739451,  0.09129011],
       [-2.38984217, -0.64683538,  0.0157382 ]])
```

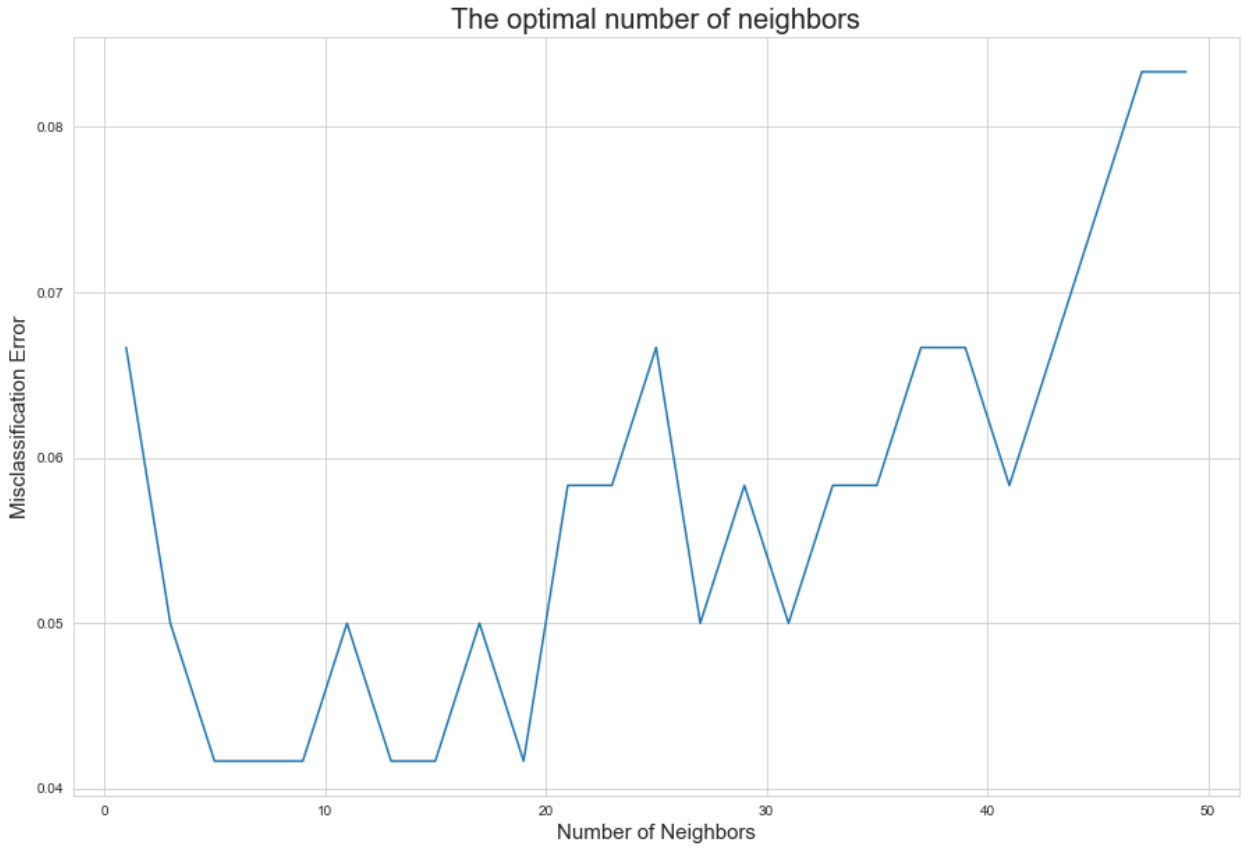
Then, we implement the KNN with K = 3:

```
Confusion Matrix
 [[ 8  1  0]
 [ 0 11  0]
 [ 0  0 10]]
              precision    recall  f1-score   support

           0       1.00      0.89      0.94         9
           1       0.92      1.00      0.96        11
           2       1.00      1.00      1.00        10

    accuracy                           0.97        30
   macro avg       0.97      0.96      0.97        30
weighted avg       0.97      0.97      0.97        30
```

There is not much difference compared to the case KNN (K = 3) without PCA. The accuracy is still remained the same at the rate of 0.97.

5. K-cross validation

Applying the K-cross validation with 10 folds, we get the following graph of the optimal number of neighbors:



Then, we implement the KNN with K = 3:

```
Confusion Matrix
 [[ 9  0  0]
 [ 0 12  0]
 [ 0  0  9]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         9
           1       1.00      1.00      1.00        12
           2       1.00      1.00      1.00         9

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

We can see that compared to the case when using only KNN with K = 3, the performance of KNN with the help of K-cross validation has been improved (from 0.97 to 1.00).

6. Leave-one-out

Implementing the leave-one-out method, we get the mean of the cross validation score of 0.975, which results in the classification error of 0.025.

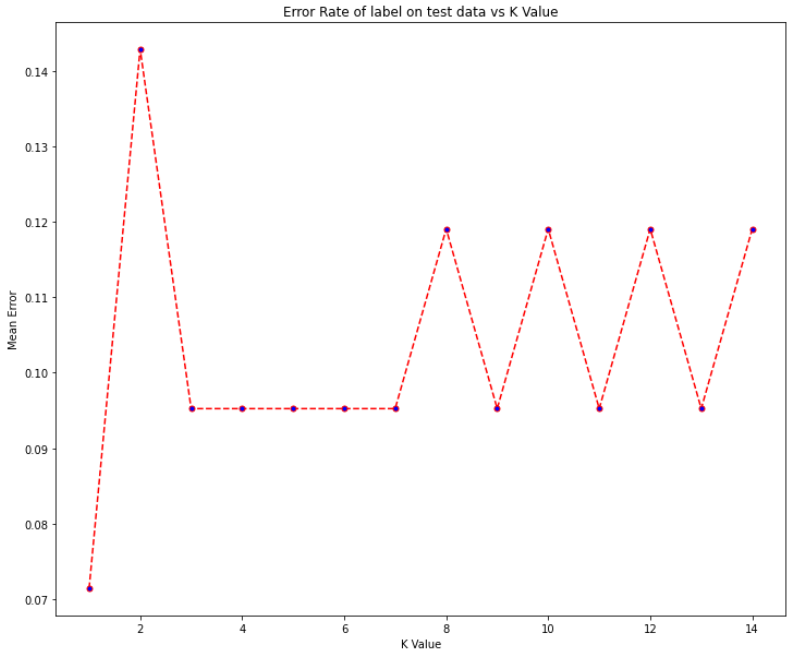
II/ Seeds dataset

1. Implement KNN and calculate classification error

After implementing K-nearest neighbor (where K = 7) and testing on the Seeds dataset, we get the following result with some important statistic such as the confusion matrix, precision, accuracy...

```
Confusion Matrix
 [[10  0  2]
 [ 1 14  0]
 [ 0  0 15]]
              precision    recall  f1-score   support

           0       0.91      0.83      0.87        12
           1       1.00      0.93      0.97        15
           2       0.88      1.00      0.94        15

    accuracy                           0.93        42
   macro avg       0.93      0.92      0.92        42
weighted avg       0.93      0.93      0.93        42
```

In terms of the classification error, we print the graph below:



2. Vary the value of K

Applying the same K-nearest neighbor method with the decrease in the K value of two units (K = 5), we get the following result:

```
Confusion Matrix
 [[11  1  1]
 [ 0 15  0]
 [ 0  0 14]]
              precision    recall  f1-score   support

           0       1.00      0.85      0.92        13
           1       0.94      1.00      0.97        15
           2       0.93      1.00      0.97        14

    accuracy                           0.95        42
   macro avg       0.96      0.95      0.95        42
weighted avg       0.96      0.95      0.95        42
```

Compared to the case where K = 7, the accuracy rate of this case is higher. To be more specific, when K = 7, the accuracy is only 0.93, but it can get the rate of 0.95 when K = 5.

3. Normalize the input dataset then use KNN

By normalizing the Seeds dataset before implementing the KNN, the performance cannot be improved: the accuracy reduce from 0.95 to 0.90:

```
Confusion Matrix
 [[14  1  1]
 [ 1 12  0]
 [ 1  0 12]]
              precision    recall  f1-score   support

           0       0.88      0.88      0.88        16
           1       0.92      0.92      0.92        13
           2       0.92      0.92      0.92        13

    accuracy                           0.90        42
   macro avg       0.91      0.91      0.91        42
weighted avg       0.90      0.90      0.90        42
```

4. Apply PCA then use KNN

Firstly, we apply the PCA with two components on the dataset:

```
array([[-0.2983016 ,  2.27121283],
       [ 0.15146837,  2.29712372],
       [ 0.27946559,  2.34430715],
       [-1.38397001,  2.21573759],
       [ 0.0394514 ,  2.07290297]])
```

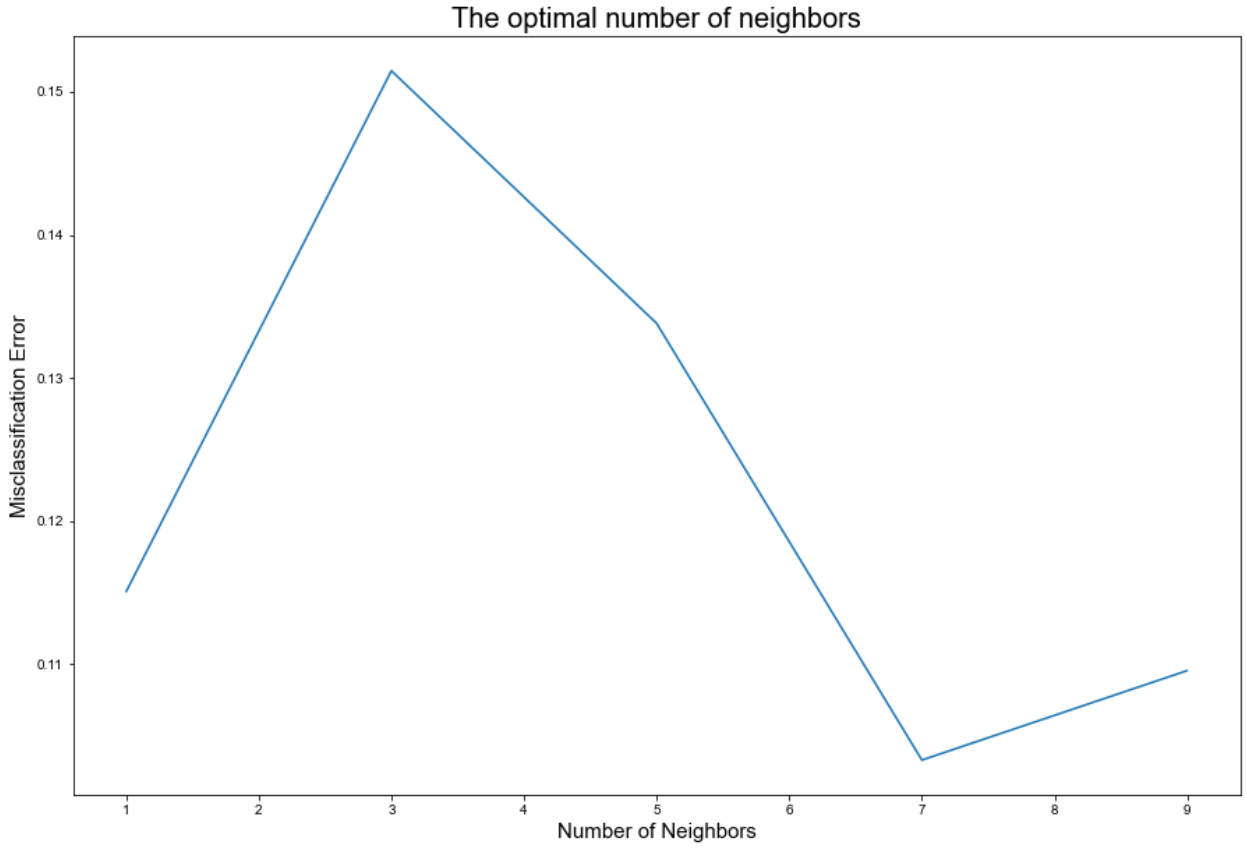
Then, we implement the KNN with K = 5:

```
Confusion Matrix
 [[10  1  0]
 [ 0 13  0]
 [ 0  0 18]]
              precision    recall  f1-score   support

           0       1.00      0.91      0.95        11
           1       0.93      1.00      0.96        13
           2       1.00      1.00      1.00        18

    accuracy                           0.98        42
   macro avg       0.98      0.97      0.97        42
weighted avg       0.98      0.98      0.98        42
```

The performance improves so much compared to the case KNN (K = 5) without PCA. The accuracy increases from 0.95 to 0.98 (nearly the maximum rate).

5. K-cross validation

Applying the K-cross validation with 10 folds, we get the following graph of the optimal number of neighbors:



Then, we implement the KNN with K = 5:

```
Confusion Matrix
 [[13  0  3]
 [ 1 10  0]
 [ 1  0 14]]
              precision    recall  f1-score   support

           0       0.87      0.81      0.84        16
           1       1.00      0.91      0.95        11
           2       0.82      0.93      0.87        15

    accuracy                           0.88        42
   macro avg       0.90      0.88      0.89        42
weighted avg       0.89      0.88      0.88        42
```

We can see that compared to the case when using only KNN with K = 5, the performance of KNN with the help of K-cross validation decreases somehow (from 0.95 to 0.88).

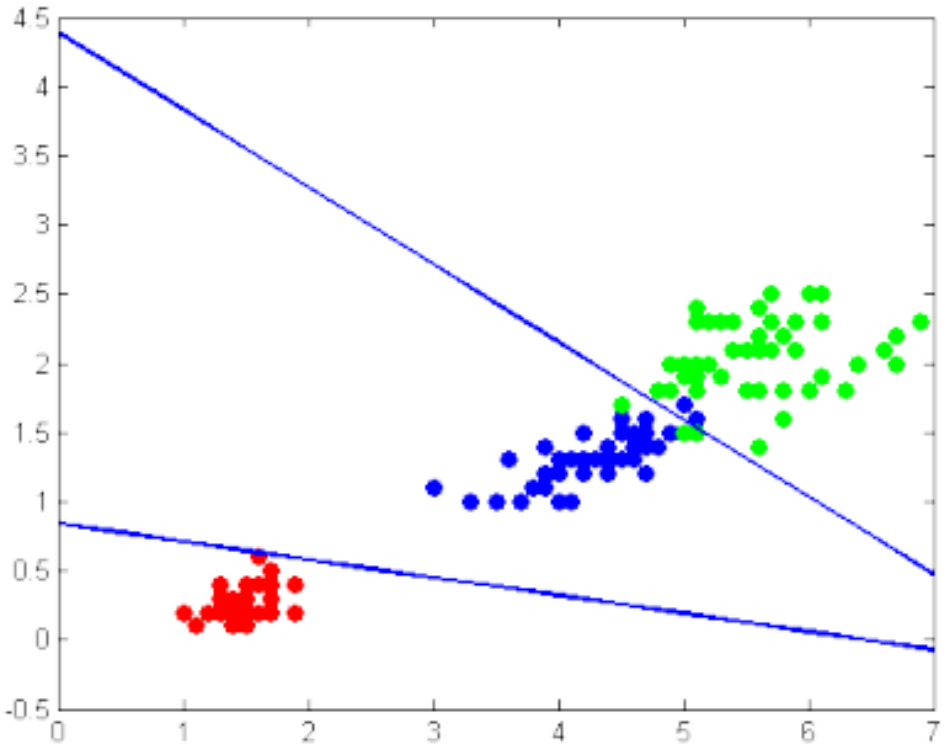
6. Leave-one-out

Implementing the leave-one-out method, we get the mean of the cross validation score of 0.952, which results in the classification error of 0.048.

B/ Perceptron classifier

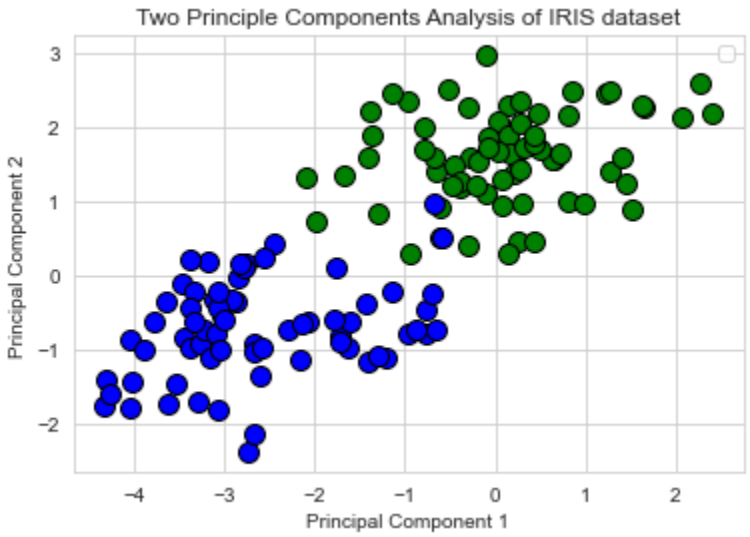
I/ Iris dataset

Dealing with Iris dataset is a multi-class problem because there are three classes in this dataset. Therefore, before using Perceptron classifier, we have to use PCA with two components in order to turn it into a 2-class problem:



II/ Seeds dataset

Dealing with Iris dataset is a multi-class. Therefore, before using Perceptron classifier, we have to use PCA with two components and we get the following result:



C/ Source code

Here is the Github repository to the source code of our labwork if you want to check:
https://github.com/DaoHaiLong/Machine-Learning-and-Data-Mining-II