```
0.89019608, 0.84705882, 0.81960784, 0.93333333, 0.        ,
0.        , 0.        , 0.        , 0.        , 0.04313725,
0.96078431, 0.81568627, 0.8       , 0.90196078, 0.90588235,
0.5372549 , 0.57647059, 0.56470588, 0.52156863, 0.49019608,
0.49411765, 0.49411765, 0.5254902 , 0.50588235, 0.51372549,
0.54117647, 0.75686275, 0.81568627, 0.87058824, 0.85490196,
0.81176471, 0.84313725, 0.15294118, 0.        , 0.        ,
0.        , 0.        , 0.12156863, 0.96470588, 0.80784314,
0.78039216, 0.87058824, 0.98431373, 0.54509804, 0.28627451,
0.33333333, 0.30196078, 0.36078431, 0.43529412, 0.50980392,
0.58823529, 0.69019608, 0.73333333, 0.78431373, 0.85490196,
0.80784314, 0.87058824, 0.85882353, 0.81176471, 0.97254902,
0.21568627, 0.        , 0.        , 0.        , 0.        ,
0.21960784, 0.97254902, 0.79607843, 0.79607843, 0.88235294,
0.97254902, 0.82352941, 0.60392157, 0.82352941, 0.91764706,
0.9254902 , 0.92156863, 0.92156863, 0.90196078, 0.87843137,
0.85098039, 0.83137255, 0.83529412, 0.81176471, 0.87058824,
0.8627451 , 0.81568627, 0.96078431, 0.23921569, 0.        ,
0.        , 0.        , 0.        , 0.25490196, 0.96470588,
0.80392157, 0.78431373, 0.89803922, 0.92156863, 0.8       ,
0.89803922, 0.88627451, 0.81568627, 0.8       , 0.8       ,
0.80392157, 0.8       , 0.81176471, 0.80392157, 0.80392157,
0.82745098, 0.79607843, 0.85882353, 0.8745098 , 0.81960784,
0.96078431, 0.28235294, 0.        , 0.        , 0.        ,
0.        , 0.33333333, 0.95686275, 0.79607843, 0.76862745,
0.9372549 , 0.91372549, 0.79215686, 0.80784314, 0.81568627,
0.82352941, 0.82352941, 0.82745098, 0.83529412, 0.81568627,
0.80784314, 0.83137255, 0.81960784, 0.82352941, 0.8       ,
0.85490196, 0.89019608, 0.82352941, 0.95294118, 0.30980392,
0.        , 0.        , 0.        , 0.        , 0.46666667,
0.94509804, 0.79215686, 0.76078431, 0.94901961, 0.88235294,
0.80784314, 0.83529412, 0.83529412, 0.83137255, 0.83137255,
0.82745098, 0.83137255, 0.81960784, 0.80392157, 0.83529412,
0.83137255, 0.82352941, 0.8       , 0.84705882, 0.89803922,
0.82352941, 0.94117647, 0.4       , 0.        , 0.        ,
0.        , 0.        , 0.56862745, 0.90196078, 0.78431373,
0.78823529, 0.94117647, 0.86666667, 0.81568627, 0.83921569,
0.83921569, 0.83137255, 0.83137255, 0.83529412, 0.83529412,
0.82352941, 0.8       , 0.81568627, 0.83137255, 0.83137255,
0.8       , 0.85098039, 0.89803922, 0.82352941, 0.91764706,
0.48627451, 0.        , 0.        , 0.        , 0.        ,
0.67058824, 0.8745098 , 0.82745098, 0.78431373, 0.9372549 ,
0.87843137, 0.80784314, 0.84313725, 0.84705882, 0.83921569,
0.83529412, 0.83529412, 0.83529412, 0.82745098, 0.81176471,
0.81176471, 0.82745098, 0.83137255, 0.81176471, 0.8627451 ,
0.87843137, 0.82352941, 0.92156863, 0.64313725, 0.        ,
0.        , 0.        , 0.        , 0.52156863, 0.92941176,
0.82352941, 0.8       , 0.92156863, 0.88627451, 0.80784314,
0.83921569, 0.83921569, 0.83529412, 0.83137255, 0.83921569,
0.83921569, 0.83529412, 0.82352941, 0.81568627, 0.82745098,
0.83137255, 0.80784314, 0.85882353, 0.88627451, 0.82352941,
0.95686275, 0.40392157, 0.        , 0.        , 0.        ,
0.        , 0.1254902 , 0.85098039, 0.8       , 0.81176471,
0.9372549 , 0.89803922, 0.81176471, 0.83529412, 0.83529412,
0.82745098, 0.81960784, 0.83529412, 0.83921569, 0.83529412,
0.82352941, 0.83137255, 0.83137255, 0.82745098, 0.78431373,
0.8627451 , 0.88627451, 0.83921569, 0.86666667, 0.15686275,
0.        , 0.        , 0.        , 0.        , 0.12941176,
0.85882353, 0.81960784, 0.82745098, 0.8       , 0.81568627,
0.85882353, 0.82352941, 0.83921569, 0.83137255, 0.82745098,
0.83137255, 0.83137255, 0.83137255, 0.81960784, 0.83137255,
0.84705882, 0.83921569, 0.81960784, 0.82745098, 0.9254902 ,
0.83137255, 0.90588235, 0.21568627, 0.        , 0.        ,
0.        , 0.        , 0.08235294, 0.78039216, 0.84313725,
0.89411765, 0.58431373, 0.65882353, 0.87843137, 0.82352941,
0.82745098, 0.83137255, 0.83921569, 0.83921569, 0.83529412,
0.83921569, 0.83137255, 0.82745098, 0.81176471, 0.80392157,
0.83137255, 0.34901961, 0.91372549, 0.87058824, 0.77254902,
0.08235294, 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.04705882, 0.08235294, 0.        ,
0.85098039, 0.9372549 , 0.85098039, 0.87843137, 0.8627451 ,
0.85490196, 0.84313725, 0.85098039, 0.87058824, 0.87058824,
0.88627451, 0.9254902 , 0.85882353, 1.        , 0.2       ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.18823529, 0.6745098 ,
0.70980392, 0.77647059, 0.80784314, 0.81960784, 0.82352941,
0.81568627, 0.79215686, 0.76078431, 0.72941176, 0.68627451,
0.56078431, 0.41568627, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        ])
```

In [25]:

```
# PCA (after normalization)
#train_norm = train_norm.reshape((train_norm.shape[0],-1))

#from sklearn.decomposition import PCA
#pca = PCA(n_components = 200)
#train_norm_pca = pca.fit_transform(train_norm)

#test_norm_pca = pca.fit_transform(test_norm)
```

In [29]:

```
# seperate data into training and validation set (train:test 8:2)
from sklearn.model_selection import train_test_split
X_train, X_valid, y_train, y_valid = train_test_split(train_norm_pca, data_train_label, test_size = 0.01, rando


print('X_train',X_train.shape)
print('X_valid',X_valid.shape)
print('y_train',y_train.shape)
print('y_valid',y_valid.shape)
```

```
X_train (29700, 200)
X_valid (300, 200)
y_train (29700,)
y_valid (300,)
```

3. classifier

In [15]:

```
#K-fold method
#from sklearn.model_selection import StratifiedKFold

#get accuracy
#from sklearn.metrics import accuracy_score

#made pic
#import matplotlib.pyplot as plt

#running time
#from time import time

#import numpy as np

#from sklearn import svm, datasets
#from sklearn.model_selection import GridSearchCV
```

3.1 KNN classifier

In [30]:

```
# use KNN classifier atfer paramenter tunning
from time import time
from sklearn.metrics import accuracy_score
import numpy as np
from sklearn.neighbors import KNeighborsClassifier

#K-fold method
#from sklearn.model_selection import StratifiedKFold

#set parameter
para_grid_knn = {'n_neighbors':[3,4,5,6,7,8],'p':[1,2]}
#10-fold method
#cvKFold_knn = StratifiedKFold(n_splits = 10, shuffle = True)

#store data
time_knn = []
acc_knn = []

#grid search
for p in para_grid_knn["p"]:
    acc_mean = []
    time_list = []
    for n_neighbors in para_grid_knn["n_neighbors"]:
            acc= []
            start = time()

            knn = KNeighborsClassifier(n_neighbors = n_neighbors, p = p)
            knn.fit(X_train, y_train)
            y_pred = knn.predict(X_valid)
            acc.append(accuracy_score(y_pred,y_valid))
            acc_mean.append(np.around(np.mean(acc)*100,2))
            time_list.append(time()-start)
            print("n_neighbors = %d, p = %d, Accuracy = %.2f%%, running time = %.2fs"%(n_neighbors,p,acc_mean[-
    acc_knn.append(acc_mean)
    time_knn.append(time_list)
```

```
n_neighbors = 3, p = 1, Accuracy = 86.33%, running time = 1.96s
n_neighbors = 4, p = 1, Accuracy = 86.00%, running time = 1.94s
n_neighbors = 5, p = 1, Accuracy = 85.33%, running time = 1.92s
n_neighbors = 6, p = 1, Accuracy = 85.00%, running time = 1.93s
n_neighbors = 7, p = 1, Accuracy = 85.67%, running time = 1.95s
n_neighbors = 8, p = 1, Accuracy = 85.00%, running time = 1.92s
n_neighbors = 3, p = 2, Accuracy = 86.00%, running time = 0.22s
n_neighbors = 4, p = 2, Accuracy = 86.00%, running time = 0.25s
n_neighbors = 5, p = 2, Accuracy = 85.33%, running time = 0.26s
n_neighbors = 6, p = 2, Accuracy = 86.33%, running time = 0.24s
n_neighbors = 7, p = 2, Accuracy = 85.33%, running time = 0.26s
n_neighbors = 8, p = 2, Accuracy = 85.67%, running time = 0.24s
```

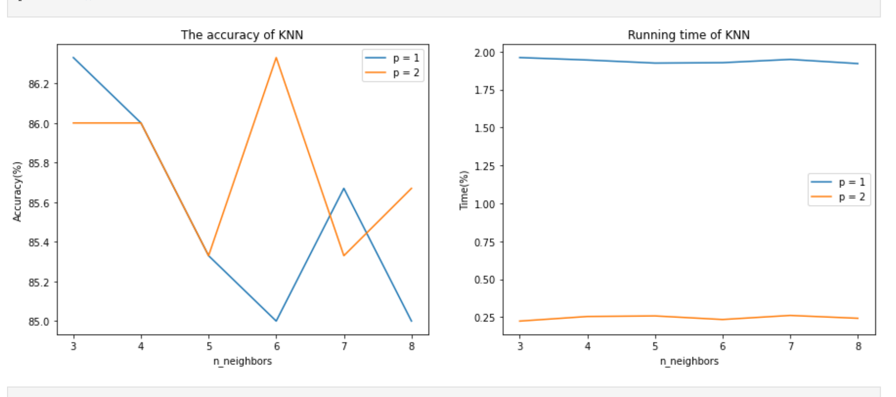
In [31]:

```
plt.figure(figsize = (15,12))

plt.subplot(2,2,1)
plt.plot(para_grid_knn["n_neighbors"],acc_knn[0])
plt.plot(para_grid_knn["n_neighbors"],acc_knn[1])
plt.legend(["p = 1","p = 2"])
plt.xticks(para_grid_knn["n_neighbors"])
plt.title("The accuracy of KNN")
plt.ylabel("Accuracy(%)")
plt.xlabel("n_neighbors")

plt.subplot(2,2,2)
plt.plot(para_grid_knn["n_neighbors"],time_knn[0])
plt.plot(para_grid_knn["n_neighbors"],time_knn[1])
plt.legend(["p = 1","p = 2"])
plt.xticks(para_grid_knn["n_neighbors"])
plt.title("Running time of KNN")
plt.ylabel("Time(%)")
plt.xlabel("n_neighbors")

plt.show()
```



In [17]:

```
import time
###KNN with 5 fold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
#from sklearn.metrics import classification_report
from sklearn.model_selection import KFold
KNN_score=[]
KNN_time=[]
kfold = KFold(n_splits=5)
knnscore_sum=0

for train_index, test_index in kfold.split(X_train, y_train):
    this_train_x, this_train_y = X_train[train_index], y_train[train_index]
    this_test_x, this_test_y = X_train[test_index], y_train[test_index]
    KNNclassifier = KNeighborsClassifier(n_neighbors = 6, p = 1)
    KNNclassifier.fit(this_train_x, this_train_y)
    prediction = KNNclassifier.predict(this_test_x)
    score = accuracy_score(this_test_y, prediction)
    print(score)
    KNN_score.append(score)
    start=time.time()
    prediction = KNNclassifier.predict(test_norm)
    end=time.time()
    print('Finish classifying, it takes '+str(end-start)+'s')
    KNN_time.append(end-start)
    knnscore_sum = knnscore_sum + score
```

```
0.8511784511784511
Finish classifying, it takes 159.8613338470459s
0.8430976430976431
Finish classifying, it takes 181.5887725353241s
0.8577441077441077
Finish classifying, it takes 154.08210062980652s
0.847979797979798
Finish classifying, it takes 155.83111786842346s
0.8501683501683501
Finish classifying, it takes 175.1950855255127s
```

In [18]:

```
knnscore_sum/5
```

Out[18]:

```
0.8500336700336699
```

3.2 SVM classifier

In [19]:

```
###SVM with 5 fold
from sklearn import svm

from sklearn.model_selection import KFold
SVM_score=[]
SVM_time=[]
kfold = KFold(n_splits=5)
svmscore_sum=0

for train_index, test_index in kfold.split(X_train, y_train):
    this_train_x, this_train_y = X_train[train_index], y_train[train_index]
    this_test_x, this_test_y = X_train[test_index], y_train[test_index]
    SVMclassifier = svm.LinearSVC(max_iter=5000,dual=False)
    SVMclassifier.fit(this_train_x, this_train_y)
    prediction = SVMclassifier.predict(this_test_x)
    score = accuracy_score(this_test_y, prediction)
    print(score)
    SVM_score.append(score)
    start=time.time()
    prediction = SVMclassifier.predict(test_norm)
    end=time.time()
    print('Finish classifying, it takes '+str(end-start)+'s')
    SVM_time.append(end-start)
    svmscore_sum = svmscore_sum + score
```

```
0.8388888888888889
Finish classifying, it takes 0.009974241256713867s
0.8294612794612795
Finish classifying, it takes 0.009477853775024414s
0.8420875420875421
Finish classifying, it takes 0.00803828239440918s
0.8373737373737373
Finish classifying, it takes 0.007039546966552734s
0.8292929292929293
Finish classifying, it takes 0.009000062942504883s
```

In [20]:

```
svmscore_sum/5
```

Out[20]:

```
0.8354208754208754
```

3.3 logistic regression

In [35]:

```
#LR parameter tunning
from sklearn.linear_model import LogisticRegression


#set parameter
para_grid_lr = {'max_iter': [100,200,300],'penalty' : ['l1','l2']}

#store data
time_lr = []
acc_lr = []

#grid search

for penalty in para_grid_lr["penalty"]:
    acc_mean = []
    time_list = []
    for max_iter in para_grid_lr["max_iter"]:
            acc= []
            start = time()

            #classifier
            lr = LogisticRegression(max_iter = max_iter, penalty = penalty,C = 1, solver = 'liblinear')
            lr.fit(X_train, y_train)

            #get acc and time
            y_pred = lr.predict(X_valid)
            acc.append(accuracy_score(y_pred,y_valid))
            acc_mean.append(np.mean(acc)*100)
            time_list.append(time()-start)
            print("max_iter = %d, penalty = %s, Accuracy = %.2f%%, running time = %.2fs"%(max_iter,penalty,acc_
    acc_lr.append(acc_mean)
    time_lr.append(time_list)
```

```
max_iter = 100, penalty = l1, Accuracy = 83.00%, running time = 179.74s
max_iter = 200, penalty = l1, Accuracy = 83.00%, running time = 210.94s
max_iter = 300, penalty = l1, Accuracy = 83.00%, running time = 203.95s
max_iter = 100, penalty = l2, Accuracy = 82.67%, running time = 43.65s
max_iter = 200, penalty = l2, Accuracy = 82.67%, running time = 47.22s
max_iter = 300, penalty = l2, Accuracy = 82.67%, running time = 45.09s
```

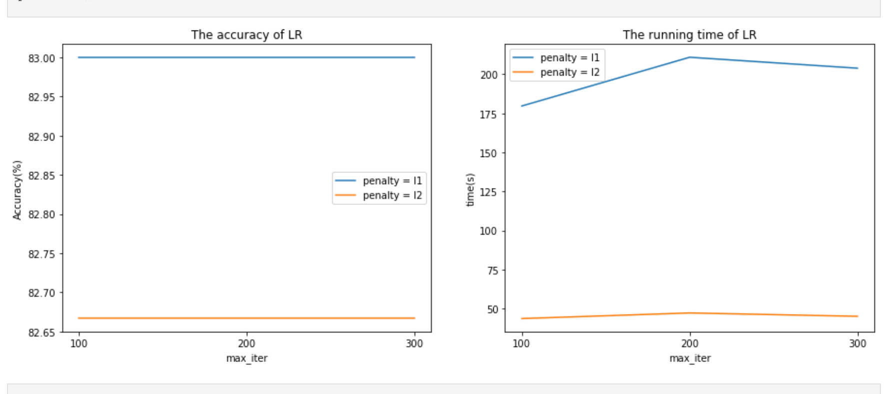
In [36]:

```
plt.figure(figsize = (15,12))

plt.subplot(2,2,1)
plt.plot(para_grid_lr["max_iter"],acc_lr[0])
plt.plot(para_grid_lr["max_iter"],acc_lr[1])
plt.legend(["penalty = l1","penalty = l2"])
plt.xticks(para_grid_lr["max_iter"])
plt.title("The accuracy of LR")
plt.ylabel("Accuracy(%)")
plt.xlabel("max_iter")

plt.subplot(2,2,2)
plt.plot(para_grid_lr["max_iter"],time_lr[0])
plt.plot(para_grid_lr["max_iter"],time_lr[1])
plt.legend(["penalty = l1","penalty = l2"])
plt.xticks(para_grid_lr["max_iter"])
plt.title("The running time of LR")
plt.ylabel("time(s)")
plt.xlabel("max_iter")

plt.show()
```



In [21]:

```
###logistic regression(LR) with 5 fold
from sklearn.linear_model import LogisticRegression
#
import time
from sklearn.metrics import accuracy_score
#from sklearn.metrics import classification_report
from sklearn.model_selection import KFold

LR_score=[]
LR_time=[]
kfold = KFold(n_splits=5)
lrscore_sum=0
for train_index, test_index in kfold.split(X_train, y_train):
    this_train_x, this_train_y = X_train[train_index], y_train[train_index]
    this_test_x, this_test_y = X_train[test_index], y_train[test_index]

    LRclassifier = LogisticRegression(solver='liblinear',
                                      max_iter=300,penalty='l2',tol=0.001)
    LRclassifier.fit(this_train_x,this_train_y)
    prediction = LRclassifier.predict(this_test_x)

    score = accuracy_score(this_test_y, prediction)
    print(score)
    LR_score.append(score)
    start=time.time()
    prediction = LRclassifier.predict(test_norm)
    end=time.time()
    print('Finish classifying, it takes '+str(end-start)+'s')
    LR_time.append(end-start)
    lrscore_sum = lrscore_sum+score
```

```
0.8486531986531987
Finish classifying, it takes 0.007008552551269531s
0.8393939393939394
Finish classifying, it takes 0.011997222900390625s
0.8501683501683501
Finish classifying, it takes 0.0070073604583740234s
0.8459595959595959
Finish classifying, it takes 0.007952690124511719s
0.8412457912457912
Finish classifying, it takes 0.009002447128295898s
```

In [22]:

```
lrscore_sum/5
```

Out[22]:

```
0.845084175084175
```

3.4 Random forest

In [39]:

```
# Random forest classifier
from sklearn.ensemble import RandomForestClassifier



#set parameter
para_grid_rf = {'max_depth':[7,9,11],'random_state':[0,2]}

#store data
time_rf = []
acc_rf = []

#grid searc
for random_state in para_grid_rf["random_state"]:
    acc_mean = []
    time_list = []
    for max_depth in para_grid_rf["max_depth"]:
            acc= []
            start = time()

            #classifier
            rf = RandomForestClassifier( max_depth= max_depth,random_state =random_state, n_estimators=500)
            rf.fit(X_train, y_train)

            #get acc and time
            y_pred = lr.predict(X_valid)
            acc.append(accuracy_score(y_pred,y_valid))
            acc_mean.append(np.around(np.mean(acc)*100,4))
            time_list.append(time()-start)
            print("max_depth = %d, random_state = %d, Accuracy = %.2f%%, running time = %.2fs"%(max_depth,rando
    acc_rf.append(acc_mean)
    time_rf.append(time_list)
```

```
max_depth = 7, random_state = 0, Accuracy = 82.67%, running time = 121.19s
max_depth = 9, random_state = 0, Accuracy = 82.67%, running time = 158.98s
max_depth = 11, random_state = 0, Accuracy = 82.67%, running time = 184.65s
max_depth = 7, random_state = 2, Accuracy = 82.67%, running time = 123.55s
max_depth = 9, random_state = 2, Accuracy = 82.67%, running time = 169.79s
max_depth = 11, random_state = 2, Accuracy = 82.67%, running time = 166.82s
```

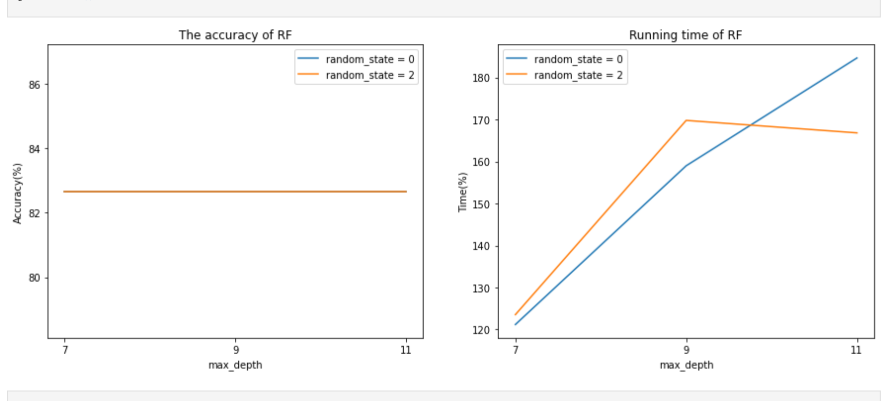
In [40]:

```
plt.figure(figsize = (15,12))

plt.subplot(2,2,1)
plt.plot(para_grid_rf["max_depth"],acc_rf[0])
plt.plot(para_grid_rf["max_depth"],acc_rf[1])
plt.legend(["random_state = 0","random_state = 2"])
plt.xticks(para_grid_rf["max_depth"])
plt.title("The accuracy of RF")
plt.ylabel("Accuracy(%)")
plt.xlabel("max_depth")

plt.subplot(2,2,2)
plt.plot(para_grid_rf["max_depth"],time_rf[0])
plt.plot(para_grid_rf["max_depth"],time_rf[1])
plt.legend(["random_state = 0","random_state = 2"])
plt.xticks(para_grid_rf["max_depth"])
plt.title("Running time of RF")
plt.ylabel("Time(%)")
plt.xlabel("max_depth")

plt.show()
```



In [23]:

```
###Roandom Forset(RF) with 5 fold
from sklearn.ensemble import RandomForestClassifier
RF_score=[]
RF_time=[]
kfold = KFold(n_splits=5)
rfscore_sum=0
for train_index, test_index in kfold.split(X_train, y_train):
    this_train_x, this_train_y = X_train[train_index], y_train[train_index]
    this_test_x, this_test_y = X_train[test_index], y_train[test_index]

    RFclassifier = RandomForestClassifier(max_depth=11, min_samples_leaf=2,
                                          n_estimators=500,min_samples_split=10,
                                          random_state=0)
    RFclassifier.fit(this_train_x,this_train_y)
    prediction = RFclassifier.predict(this_test_x)

    score = accuracy_score(this_test_y, prediction)
    print(score)
    RF_score.append(score)
    start=time.time()
    prediction = RFclassifier.predict(test_norm)
    end=time.time()
    print('Finish classifying, it takes '+str(end-start)+'s')
    RF_time.append(end-start)
    rfscore_sum = rfscore_sum+score
```

```
0.8606060606060606
Finish classifying, it takes 0.8252971172332764s
0.8488215488215488
Finish classifying, it takes 0.5999958515167236s
0.8653198653198653
Finish classifying, it takes 0.6478431224822998s
0.858922558922559
Finish classifying, it takes 0.5890023708343506s
0.8553872053872054
Finish classifying, it takes 0.6040041446685791s
```

In [24]:

```
rfscore_sum/5
```

Out[24]:

```
0.8578114478114479
```

In [25]:

```
best_acc= max(knnscore_sum/5,svmscore_sum/5, lrscore_sum/5, rfscore_sum/5)
best_acc
```

Out[25]:

```
0.8578114478114479
```

4. output

In [26]:

```
# test_input.csv includes 5000 samples used for label prediction. Test samples do not have labels.
data_test_df = pd.read_csv('./Input/test/test_input.csv')
```

In [27]:

```
data_test_df.head()
```

Out[27]:

	id	v1	v2	v3	v4	...	v780	v781	v782	v783	v784
0	0	0	0	0	0	...	0	0	0	0	0
1	1	0	0	0	0	...	0	0	0	0	0
2	2	0	0	0	0	...	0	0	0	0	0
3	3	0	0	0	0	...	0	0	0	0	0
4	4	0	0	0	0	...	0	0	0	0	0

5 rows × 785 columns

In [28]:

```
y_pred_best = RFclassifier.predict(test_norm)
y_pred_best
```

Out[28]:

```
array([1, 1, 1, ..., 5, 0, 2], dtype=int64)
```

In [29]:

```
# assume output is the predicted labels from classifiers using input as data from test_input.csv
# (5000,)


output_df = pd.DataFrame(y_pred_best, columns = ['label'])
output_df.to_csv('./Output/test_output.csv', sep=",", float_format='%d', index_label="id")
```