```
#project
import sklearn
import csv
import numpy as np
import math
import copy
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
import numpy as np
from collections import Counter
import itertools
from sklearn.metrics import confusion_matrix
np.set_printoptions(threshold=np.inf)
from sklearn import preprocessing
from sklearn.preprocessing import scale
from sklearn import svm, datasets
from sklearn import model_selection
from sklearn import metrics
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
import pandas as pd
pd.set_option('display.max_columns', None)
from sklearn.neighbors import KNeighborsClassifier
from sklearn import neighbors
from sklearn.metrics import classification_report
from sklearn.decomposition import PCA
from sklearn.neural_network import MLPClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import BernoulliNB

with open("/Users/cjr/Desktop/EE559/project/localization/D_Train1.csv","r") as
Train_Feature:
    Train_Feature_Reader = csv.reader(Train_Feature, delimiter=',')
    header_row = next(Train_Feature_Reader)
    FeatureTrainList = []
    for row in Train_Feature_Reader:
        FeatureTrainList = FeatureTrainList +[row]
```

```
Train_Feature.close()

FeatureListNolabel = copy.deepcopy(FeatureTrainList)
for row in FeatureListNolabel:
	del row[0]

Trainsetdata = np.array(FeatureListNolabel).astype("float")


data_Class = []
for row in FeatureTrainList:
	data_Class.append(row[0])

Class_Train = np.array(data_Class).astype("float")



with	open("/Users/cjr/Desktop/EE559/project/localization/D_Test1.csv","r")	as
Test_Feature:
	Test_Feature_Reader = csv.reader(Test_Feature, delimiter=',')
	header_row = next(Test_Feature_Reader)#[1]
	FeatureTestList = []
	for row in Test_Feature_Reader:
		FeatureTestList = FeatureTestList +[row]

Test_Feature.close()

FeatureListNolabel2 = copy.deepcopy(FeatureTestList)
for row in FeatureListNolabel2:
	del row[0]

Testsetdata = np.array(FeatureListNolabel2).astype("float")

data_Class = []
for row in FeatureTestList:
	data_Class.append(row[0])

Class_Test = np.array(data_Class).astype("float")

scaler = StandardScaler()
scaler.fit(Trainsetdata)
```

```
train_data = scaler.transform(Trainsetdata)
test_data = scaler.transform(Testsetdata)

pca = PCA(n_components=5) #[2]
train_data_pca = pca.fit_transform(train_data[:, 0:train_data.shape[0]])
test_data_pca = pca.transform(test_data[:, 0:test_data.shape[0]])

def perceptron():
    model = Perceptron(n_iter=1000, tol=0.0001, random_state = None) #[3]
    model.fit(train_data, Class_Train)
    print('Final Weights:')
    print(model.coef_)
    Trainlabelpred = model.predict(train_data)

    trainsetacc = accuracy_score(Class_Train, Trainlabelpred)
    print('train accurracy = ',trainsetacc)

    Testlabelpred = model.predict(test_data)

    Testsetacc = accuracy_score(Class_Test, Testlabelpred)
    print('test accurracy = ',Testsetacc)

    Weight1 = np.zeros((4,7))
    Weight0 = np.zeros((4,7))
    Sample = np.zeros((4,7))
    Accurracy = 0

    for i in range(100):
        Sample = np.random.randn(4,7)
        model3 = Perceptron(n_iter=1000, tol=0.0001, random_state = None)
        model3.fit(train_data, Class_Train, coef_init = Sample)
        Trainlabelpred1 = model3.predict(train_data)
        trainsetacc1 = accuracy_score(Class_Train, Trainlabelpred1)
        if(Accurracy < trainsetacc1):
            Accurracy = trainsetacc1
            Weight1 = model3.coef_
            Weight0 = Sample
    print('run 100 times:')
    print('Final Weights:')
    print(Weight1)
```

```
    model3.fit(train_data, Class_Train, coef_init = Weight0)
    Testlabelpred = model3.predict(test_data)
    Test_acc = accuracy_score(Class_Test, Testlabelpred)
    print('train accurracy (100 times) = ',Accurracy)
    print('test accurracy (100 times) = ',Test_acc)


def NaiveBayes():
    print('GaussianNB:\n')
    clf = GaussianNB()
    clf.fit(train_data, Class_Train)
    pred = clf.predict(test_data)

    accuracy = clf.score(test_data, Class_Test)
    print('accuracy on test data=', accuracy)
    print('confusion matrix :\n',confusion_matrix(Class_Test, pred))

    clf = GaussianNB()
    clf.fit(train_data_pca, Class_Train)
    pred = clf.predict(test_data_pca)

    accuracy = clf.score(test_data_pca, Class_Test)
    print('accuracy on test data with PCA =', accuracy)
    print('confusion matrix :\n',confusion_matrix(Class_Test, pred))

    print('BernoulliNB:\n')
    clf = BernoulliNB()
    clf.fit(train_data, Class_Train)
    pred = clf.predict(test_data)

    accuracy = clf.score(test_data, Class_Test)
    print('accuracy on test data=', accuracy)
    print('confusion matrix :\n',confusion_matrix(Class_Test, pred))

    clf = BernoulliNB()
    clf.fit(train_data_pca, Class_Train)
    pred = clf.predict(test_data_pca)

    accuracy = clf.score(test_data_pca, Class_Test)
    print('accuracy on test data with PCA =', accuracy)
```

```
    print('confusion matrix :\n',confusion_matrix(Class_Test, pred))

def SVM_linear():
    #SVM linear kernel (all features vs PCA)
    #Model
    model = SVC(C=1, kernel='linear',gamma=1) #[4]
    #model = SVC(C=100, kernel='linear',gamma=1)
    model.fit(train_data, Class_Train)

    Acc= accuracy_score(Class_Train, model.predict(train_data))
    print("accuracy on train dataset=",Acc)

    acc = accuracy_score(Class_Test, model.predict(test_data))
    print('Accuracy on testset = ', acc)
    print('confusion matrix :\n',confusion_matrix(Class_Test, model.predict(test_data)))

    model = SVC(C=1, kernel='linear',gamma=1)
    #model = SVC(C=100, kernel='linear',gamma=1)
    model.fit(train_data_pca, Class_Train)

    Acc= accuracy_score(Class_Train, model.predict(train_data_pca))
    print("accuracy on train dataset with PCA=",Acc)

    acc = accuracy_score(Class_Test, model.predict(test_data_pca))
    print('Accuracy on testset with PCA= ', acc)
    print('confusion matrix :\n',confusion_matrix(Class_Test,
model.predict(test_data_pca)))

def SVM_rbf():
    x = train_data
    y = Class_Train

    C = np.logspace(-3, 3, 50)
    gamma = np.logspace(-3, 3, 50)

    ACC = np.zeros((50,50))
    DEV = np.zeros((50,50))

    SKF = StratifiedKFold(n_splits = 5, shuffle = True)

    for i, r in enumerate(gamma):
```

```
        for j, c in enumerate(C):
            Total_Acc=[]
            for train_index, dev_index in SKF.split(x, y):

                Classifier = SVC(C = c, kernel = 'rbf', gamma = r, )
                feature_train, feature_dev = x[train_index], x[dev_index]
                label_train, label_dev = y[train_index], y[dev_index]
                Classifier.fit(feature_train, label_train)
                label_pred=Classifier.predict(feature_dev)
                acc=accuracy_score(label_dev, label_pred)
                Total_Acc.append(acc)

            ACC[i,j] = np.mean(Total_Acc)
            DEV[i,j] = np.std(Total_Acc)

    i, j = np.argwhere(ACC == np.max(ACC))[0]

    print('Using SVM rbf kernel with cross validation: ')
    print('For the best pair, gamma =', gamma[i])
    print('C =', C[j])
    print('with mean cross-validation accuracy = ', ACC[i,j])
    print('and standard deviation = ', DEV[i,j])


    Classifier = SVC(C = C[j], kernel = 'rbf', gamma = gamma[i],
decision_function_shape = 'ovr')
    Classifier.fit(train_data, Class_Train)
    acc = accuracy_score(Class_Test, Classifier.predict(test_data))
    print('Accuracy on testset = ', acc)
    print('confusion matrix :\n',confusion_matrix(Class_Test,
Classifier.predict(test_data)))


    x = train_data_pca
    y = Class_Train

    C = np.logspace(-3, 3, 50)
    gamma = np.logspace(-3, 3, 50)

    ACC = np.zeros((50,50))
    DEV = np.zeros((50,50))
```

```
    SKF = StratifiedKFold(n_splits = 5, shuffle = True)

    for i, r in enumerate(gamma):
        for j, c in enumerate(C):
            Total_Acc=[]
            for train_index, dev_index in SKF.split(x, y):

                Classifier = SVC(C = c, kernel = 'rbf', gamma = r, )
                feature_train, feature_dev = x[train_index], x[dev_index]
                label_train, label_dev = y[train_index], y[dev_index]
                Classifier.fit(feature_train, label_train)
                label_pred=Classifier.predict(feature_dev)
                acc=accuracy_score(label_dev, label_pred)
                Total_Acc.append(acc)

            ACC[i,j] = np.mean(Total_Acc)
            DEV[i,j] = np.std(Total_Acc)

    i, j = np.argwhere(ACC == np.max(ACC))[0]

    print('Using SVM rbf kernel with cross validation&PCA: ')
    print('For the best pair, gamma =', gamma[i])
    print('C =', C[j])
    print('with mean cross-validation accuracy = ', ACC[i,j])
    print('and standard deviation = ', DEV[i,j])


    Classifier = SVC(C = C[j], kernel = 'rbf', gamma = gamma[i],
decision_function_shape = 'ovr')
    Classifier.fit(train_data_pca, Class_Train)
    acc = accuracy_score(Class_Test, Classifier.predict(test_data_pca))
    print('Accuracy on testset = ', acc)
    print('confusion matrix :\n',confusion_matrix(Class_Test,
Classifier.predict(test_data_pca)))

def fiveNN():
    x = train_data
    y = Class_Train
    z = test_data
    Total_Acc = []
```

```
    for i in range(0, 10):
        SKF = StratifiedKFold(n_splits=5, shuffle=True)
        aacc = []
        for train_index, dev_index in SKF.split(x, y):
            X_cv_train, X_cv_dev = x[train_index], x[dev_index]
            y_cv_train, y_cv_dev = y[train_index], y[dev_index]
            model =
neighbors.KNeighborsClassifier(n_neighbors=5,weights="uniform",algorithm="auto")#[
5]
            model.fit(X_cv_train, y_cv_train)
            acc = accuracy_score(y_cv_dev, model.predict(X_cv_dev))
            aacc.append(acc)
        Total_Acc.append(np.mean(aacc))
    print('train dataset accuracy for 5NN: ', np.mean(Total_Acc))
    model = neighbors.KNeighborsClassifier(n_neighbors=5, algorithm="auto")

    model.fit(x, y)
    test_acc = accuracy_score(Class_Test, model.predict(z))
    print('test dataset accuracy for 5NN: ',test_acc)
    print('confusion matrix :\n',confusion_matrix(Class_Test, model.predict(z)))


    x = train_data_pca
    y = Class_Train
    z = test_data_pca
    Total_Acc = []
    for i in range(0, 10):
        SKF = StratifiedKFold(n_splits=5, shuffle=True)
        aacc = []
        for train_index, dev_index in SKF.split(x, y):
            X_cv_train, X_cv_dev = x[train_index], x[dev_index]
            y_cv_train, y_cv_dev = y[train_index], y[dev_index]
            model =
neighbors.KNeighborsClassifier(n_neighbors=5,weights="uniform",algorithm="auto")
            model.fit(X_cv_train, y_cv_train)
            acc = accuracy_score(y_cv_dev, model.predict(X_cv_dev))
            aacc.append(acc)
        Total_Acc.append(np.mean(aacc))
    print('train dataset accuracy for 5NN with PCA: ', np.mean(Total_Acc))
    model = neighbors.KNeighborsClassifier(n_neighbors=5, algorithm="auto")
```

```
    model.fit(x, y)
    test_acc = accuracy_score(Class_Test, model.predict(z))
    print('test dataset accuracy for 5NN with PCA: ',test_acc)
    print('confusion matrix :\n',confusion_matrix(Class_Test, model.predict(z)))

def kNN():
    x = train_data
    y = Class_Train
    z = test_data
    Total_Acc = []
    ks = []
    for k in range(3, 12):
        for i in range(0, 10):
            SKF = StratifiedKFold(n_splits=5, shuffle=True)
            aacc = []
            for train_index, dev_index in skf.split(x, y):
                X_cv_train, X_cv_dev = x[train_index], x[dev_index]
                y_cv_train, y_cv_dev = y[train_index], y[dev_index]
                model                                                    =
neighbors.KNeighborsClassifier(n_neighbors=k,weights="uniform",algorithm="auto")
                model.fit(X_cv_train, y_cv_train)
                acc = accuracy_score(y_cv_dev, model.predict(X_cv_dev))
                aacc.append(acc)
            Total_Acc.append(np.mean(aacc))
        ks.append(np.mean(Total_Acc))

    j = np.argmax(ks)
    print('Best accuracy on train dataset: ', ks[j], 'with k =', j + 3)#[6]
    model1 = neighbors.KNeighborsClassifier(n_neighbors=j + 3,algorithm="auto")

    model1.fit(x, y)
    test_acc = accuracy_score(Class_Test, model1.predict(z))
    print('accuracy on test dataset: ', test_acc)
    print('confusion matrix :\n',confusion_matrix(Class_Test, model1.predict(z)))


    print('knn & PCA:')
    x = train_data_pca
    y = Class_Train
    z = test_data_pca
    Total_Acc = []
```

```
    ks = []
    for k in range(3, 12):
        for i in range(0, 10):
            SKF = StratifiedKFold(n_splits=5, shuffle=True)
            aacc = []
            for train_index, dev_index in skf.split(x, y):
                X_cv_train, X_cv_dev = x[train_index], x[dev_index]
                y_cv_train, y_cv_dev = y[train_index], y[dev_index]
                model =
neighbors.KNeighborsClassifier(n_neighbors=k,weights="uniform",algorithm="auto")
                model.fit(X_cv_train, y_cv_train)
                acc = accuracy_score(y_cv_dev, model.predict(X_cv_dev))
                aacc.append(acc)
            Total_Acc.append(np.mean(aacc))
        ks.append(np.mean(Total_Acc))

    j = np.argmax(ks)
    print('Best accuracy on train dataset: ', ks[j], 'with k =', j + 3)
    model1 = neighbors.KNeighborsClassifier(n_neighbors=j + 3,algorithm="auto")

    model1.fit(x, y)
    test_acc = accuracy_score(Class_Test, model1.predict(z))
    print('accuracy on test dataset: ', test_acc)
    print('confusion matrix :\n',confusion_matrix(Class_Test, model1.predict(z)))

def NN():
    print('NN:')
    x = train_data
    y = Class_Train
    z = test_data
    Total_Acc = []
    layers = [(100),(10,10),(10, 10, 10), (10, 10, 10, 10),(10, 10, 10, 10, 10)]
    for i in range(len(layers)):
        SKF = StratifiedKFold(n_splits=5, shuffle=True)
        aacc = []
        for train_index, dev_index in SKF.split(x, y):
            X_cv_train, X_cv_dev = x[train_index], x[dev_index]
            y_cv_train, y_cv_dev = y[train_index], y[dev_index]
            model = MLPClassifier(hidden_layer_sizes=layers[i], solver="lbfgs")#[7]
            model.fit(X_cv_train, y_cv_train)
            acc = accuracy_score(y_cv_dev, model.predict(X_cv_dev))
```

```
            aacc.append(acc)

        Total_Acc.append(np.mean(aacc))
        print('hidden layer =',layers[i],'accuracy on train dataset(avr) = ', np.mean(aacc))

        j = np.argmax(Total_Acc)
        print('Best accuracy = ', Total_Acc[j], 'with hidden layer =', layers[j])

        model1 = MLPClassifier(hidden_layer_sizes=layers[j], solver="lbfgs")
        model1.fit(x, y)

        test_acc = accuracy_score(Class_Test, model1.predict(z))
        print('accuracy on test dataset: ',test_acc)
        print('confusion matrix :\n',confusion_matrix(Class_Test, model1.predict(z)))
        print('\n')



print('NN with PCA:')
x = train_data_pca
y = Class_Train
z = test_data_pca
Total_Acc = []
layers = [(100),(10,10),(10, 10, 10), (10, 10, 10, 10),(10, 10, 10, 10, 10)]
for i in range(len(layers)):
        SKF = StratifiedKFold(n_splits=5, shuffle=True)
        aacc = []
        for train_index, dev_index in SKF.split(x, y):
                X_cv_train, X_cv_dev = x[train_index], x[dev_index]
                y_cv_train, y_cv_dev = y[train_index], y[dev_index]
                model = MLPClassifier(hidden_layer_sizes=layers[i], solver="lbfgs")
                model.fit(X_cv_train, y_cv_train)
                acc = accuracy_score(y_cv_dev, model.predict(X_cv_dev))
                aacc.append(acc)

        Total_Acc.append(np.mean(aacc))
        print('hidden layer =',layers[i],'accuracy on train dataset(avr) = ', np.mean(aacc))

        j = np.argmax(Total_Acc)
        print('Best accuracy = ', Total_Acc[j], 'with hidden layer =', layers[j])
```

```
        model1 = MLPClassifier(hidden_layer_sizes=layers[j], solver="lbfgs")
        model1.fit(x, y)

        test_acc = accuracy_score(Class_Test, model1.predict(z))
        print('accuracy on test dataset: ',test_acc)
        print('confusion matrix :\n',confusion_matrix(Class_Test, model1.predict(z)))
        print('\n')



def main():
    #perceptron()
    NaiveBayes()
    #SVM_linear()
    #SVM_rbf()
    #fiveNN()
    #kNN()
    #NN()
if __name__ == '__main__':
    main()
```


#Reference:
#[1] cite from 'https://fishc.com.cn/thread-106393-1-1.html'
#[2] cite from 'https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html'
#[3] cite from HW6 code
#[4] cite from HW8 code
#[5] cite from 'https://blog.csdn.net/weixin_41990278/article/details/93169529'
#[6] cite from 'https://github.com/liu578/news_popularity_classifier/blob/master/code.py'
#[7] cite from 'https://blog.csdn.net/weixin_38278334/article/details/83023958'