Author: Ruochen Pi UID:500055496 Group 174 Update Time: 2022/4/7 Hardware and software specifications of the computer: Processor: Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz (8 CPUs), ~2.0GHz Memory: 16384MB RAM Available OS Memory: 16226MB RAM 1. input data In [3]: import pandas as pd import os import numpy as np print(os.listdir("./Input/train")) pd.set_option('display.max_columns', 10) ['train.csv'] In [4]: # train.csv including feature and label using for training model. data_train_df = pd.read_csv('./Input/train/train.csv') # 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', index col=0) In [5]: data train df.head() id v1 v2 v3 v4 ... v781 v782 v783 v784 label Out [5]: 0 0 ... 0 0 0 0 ... 0 0 2 0 0 0 0 0 1 0 4 0 0 0 1 ... 0 0 ... 8 5 rows × 786 columns In [6]: data_test df.head() v1 v2 v3 v4 v5 ... v780 v781 v782 v783 v784 Out[6]: id 0 0 0 0 0 ... 0 0 0 0 0 0 0 ... 0 0 0 0 0 0 0 0 0 0 0 0 ... 0 0 0 0 0 0 ... 0 0 5 rows × 784 columns In [7]: # Selecting input feature data_train_feature = data_train_df.loc[:, "v1":"v784"].to_numpy() # Selecting test feature data_test_feature = data_test_df.loc[:, "v1":"v784"].to_numpy() # Selecting output lable data train label = data train df.label.to numpy() In [8]: import matplotlib.pyplot as plt data_train_feature = data_train_feature.reshape((data_train_feature.shape[0], 28, 28)) plt.imshow(data train feature[0], cmap=plt.get cmap('gray')) plt.title("class " + str(data train label[0]) + ": Pullover") plt.show() class 2: Pullover 0 15 20 10 15 25 In [9]: data test df.head() Out[9]: v1 v2 v3 v4 v5 ... v780 v781 v782 v783 v784 0 0 0 0 ... 0 0 0 0 0 0 ... 0 0 0 0 ... 0 0 0 0 0 0 0 0 ... 0 0 0 0 5 rows × 784 columns 2. prediction In [10]: # prediction on the test data # normalization #def norm(data): min = data.min(axis=0) max = data.max(axis=0)norm_result = (data-min)/(max-min) # return norm result #train_norm = norm(data_train_feature) data_train_feature = data_train_feature.reshape((data_train_feature.shape[0],-1)) Train_Num=data_train_feature.shape[0] data_test_feature = data_test_feature.reshape((data_test_feature.shape[0],-1)) data all=np.vstack((data train feature, data test feature)) # In[35]: from sklearn.preprocessing import MinMaxScaler scaler = MinMaxScaler() scaler.fit(data all) all norm = scaler.transform(data all) train_norm = all_norm[:Train_Num,:] test_norm = all_norm[Train_Num:,:] In [11]: data train feature[0] 0, 0, 131, 184, 199, 229, 234, 217, array([0, 0, 0, Ο, 1, Out[11]: 212, 204, 208, 226, 227, 203, 185, 173, 44, 0, 4, 0, 0, 0, 0, 0, 0, 2, 0, 0, 214, 224, 116, 78, 149, 141, 148, 131, 121, 141, 141, 169, 212, 251, 136, 0, 10, 0, 0, 0, Ο, 0, 5, 0, 43, 220, 217, Ο, Ο, 1, 213, 104, 13, 6, 49, 36, 11, 37, 121, 179, 208, 227, 155, Ο, 0, 0, 0, 1, 0, 0, 0, 155, 0, 0, Ο, Ο, 233, 217, 226, 255, 252, 133, 64, 109, 127, 175, 240, 232, 209, 224, 204, 0, 0, 0, 0, 3, 0, 0, 0, 0, 3, 0, 212, 227, 223, 223, 217, 230, 241, 237, 210, 252, 229, 222, 213, 218, 221, 216, 0, 0, 7, 0, 0, 0, 0, 0, 13, 193, 223, 215, 218, 215, 224, 225, 219, 213, 209, 212, 217, 225, 225, 224, 217, 223, 198, 0, 0, 2, 0, 0, 0, 0, 197, 227, 211, 216, 215, 251, 236, 212, 250, 221, 213, 213, 207, 209, 208, 212, 214, 210, 232, 169, 0, 0, 0, 0, 0, 13, 214, 206, 216, 214, 246, 116, 14, 29, 0, 212, 215, 211, 214, 209, 211, 210, 209, 212, 203, 207, Ο, 0, 0, 0, 62, 223, 208, 222, 229, 195, 0, 137, 240, 200, 219, 214, 211, 207, 212, 216, 204, 0, 103, Ο, 0, 0, 0, 104, 223, 203, 224, 236, 70, 221, Ο, 191, 66, 19, 59, 35, 96, 227, 203, 207, 223, 221, 211, 204, 215, 203, 224, 131, 0, 0, 0, 0, 0, 169, 220, 204, 0, 230, 199, 204, 208, 210, 226, 221, 222, 229, 34, 0, 54, 223, 209, 217, 208, 216, 193, 0, 0, 0, 0, 0, 0, 206, 216, 205, 220, 223, 164, 221, 156, 149, 239, 217, 140, 244, 223, 139, 152, 230, 216, 219, 209, 211, 217, 0, 0, 0, 0, 0, 222, 213, 205, 222, 219, 150, 152, 201, 171, 162, 121, 120, 156, 151, 130, 162, 225, 214, 221, 211, 211, 225, 0, 0, 0, 0, 233, 209, 207, 231, 230, 173, 164, 176, 155, 163, 174, 141, 150, 147, 153, 157, 195, 221, 227, 216, 209, 238, 0, 0, 0, 11, 245, 208, 204, 230, 231, 137, 147, 144, 133, 125, 126, 126, 134, 129, 131, 138, 193, 208, 222, 218, 207, 215, 39, 0, 0, 0, 31, 246, 206, 199, 222, 251, 139, 73, 85, 77, 92, 111, 130, 150, 176, 187, 200, 218, 206, 222, 219, 207, 248, 55, 0, 0, 0, 56, 248, 203, 203, 225, 248, 210, 154, 210, 234, 236, 235, 235, 230, 224, 217, 212, 213, 207, 222, 220, 208, 245, 61, 0, 0, 0, 65, 246, 205, 200, 229, 235, 204, 229, 226, 208, 204, 204, 205, 204, 207, 205, 205, 211, 203, 219, 223, 209, 245, 72, 0, 0, 0, 0, 85, 244, 203, 196, 239, 233, 202, 206, 208, 210, 210, 211, 213, 208, 206, 212, 209, 210, 204, 218, 227, 210, 243, 79, 0, 0, 0, 119, 241, 202, 194, 242, 225, 206, 213, 213, 212, 212, 211, 212, 209, 205, 213, 212, 210, 204, 216, 229, 210, 240, 102, 0, 0, 145, 230, 200, 201, 240, 221, 208, 214, 214, 212, 212, 213, 213, 210, 204, 208, 212, 212, 204, 217, 229, 210, 234, 0, 0, 0, 171, 223, 211, 200, 239, 224, 206, 215, 216, 214, 213, 213, 213, 211, 207, 207, 211, 212, 207, 220, 224, 210, 235, 164, 0, 0, 0, 133, 237, 210, 204, 235, 226, 206, 214, 214, 213, 212, 214, 214, 213, 210, 208, 211, 212, 206, 219, 226, 210, 244, 103, 0, 0, 0, 32, 217, 204, 207, 239, 229, 207, 213, 213, 211, 209, 213, 214, 213, 210, 212, 212, 211, 200, 220, 226, 214, 221, 40, 0, 0, 0, 0, 33, 219, 209, 211, 204, 208, 219, 210, 214, 212, 211, 212, 212, 212, 209, 212, 216, 214, 209, 211, 236, 212, 231, 55, 0, 0, 21, 199, 215, 228, 149, 168, 224, 210, 211, 212, 214, 214, 213, 214, 212, 211, 207, 205, 212, 89, 233, 222, 197, 21, 0, 0, 0, 12, 21, 0, 217, 239, 217, 224, 220, 218, 215, 217, 222, 222, 226, 236, 219, 255, 51, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 48, 172, 181, 198, 206, 209, 210, 208, 202, 194, 186, 175, 143, 106, Ο, 0, 0], dtype=int64) 0, In [12]: train norm[0] , 0. , 0. , 0. , 0.00505051, array([0. , 0.59817352, 0.75720165, 0.78039216, , 0. 0.89803922, 0.91764706, 0.85098039, 0.83137255, 0.8 0.81568627, 0.88627451, 0.89019608, 0.79607843, 0.7254902 , 0.68924303, 0.17254902, 0. , 0.01818182, 0. , 0. , 0. , 0. 0.00784314, 0. 0.83921569, 0.87843137, 0.45490196, 0.30588235, 0.58431373, 0.55294118, 0.58039216, 0.51372549, 0.4745098 , 0.55294118, 0.55294118, 0.6627451 , 0.83137255, 0.98431373, 0.53333333,

 0.
 , 0.03921569, 0.
 , 0.
 , 0.
 , 0.
 , 0.
 , 0.
 , 0.
 , 0.
 , 0.00425532, 0.
 , 0.01960784, 0.
 , 0.16862745, 0.8627451, 0.85098039,

 0.83529412, 0.40784314, 0.05098039, 0.02352941, 0.19215686, 0.14117647, 0.04313725, 0.14509804, 0.4745098 , 0.70196078, 0.81568627, 0.89019608, 0.60784314, 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. 0.60784314, 0.91372549, 0.85098039, 0.88627451, 1. 0.98823529, 0.52156863, 0.25098039, 0.42745098, 0.49803922, 0.68627451, 0.94117647, 0.90980392, 0.81960784, 0.87843137,

 0.8
 , 0.
 , 0.
 , 0.01176471, 0.
 , 0.

 0.
 , 0.
 , 0.
 , 0.
 , 0.
 , 0.

 0.01176471, 0.
 , 0.
 , 0.83137255, 0.89019608,

 0.8745098 , 0.8745098 , 0.85098039, 0.90196078, 0.94509804, 0.92941176, 0.82352941, 0.98823529, 0.89803922, 0.87058824, 0.83529412, 0.85490196, 0.86666667, 0.84705882, 0. 0.75686275, 0.8745098 , 0.84313725, 0.85490196, 0.84313725, 0.87843137, 0.88235294, 0.85882353, 0.83529412, 0.81960784, 0.83137255, 0.85098039, 0.88235294, 0.88235294, 0.87843137, 0.84705882, 0.84313725, 0.98431373, 0.9254902, 0.83137255, 0.98039216, 0.86666667, 0.83529412, 0.83529412, 0.81176471, 0.81960784, 0.81568627, 0.83137255, 0.83921569, 0.82352941, , 0.05098039, 0.83921569, 0.80784314, 0.84705882, 0.83921569, 0.96470588, 0.45490196, 0.05490196, 0.11372549, 0. , 0.83137255, 0.84313725, 0.82745098, 0.83921569, 0.81960784, 0.82745098, 0.82352941, 0.81960784, 0.83137255, 0.79607843, 0.81176471, 0.87058824, 0.89803922, 0.76470588, 0. , 0.40392157, 0. , 0. , 0.5372549 , 0.94117647, 0.78431373, 0.85882353, 0.83921569, 0.82745098, 0.81176471, 0.83137255,

 0.84705882, 0.8
 , 0.866666667, 0.2745098 , 0.

 0.
 , 0.

 0.40784314, 0.8745098 , 0.79607843, 0.87843137, 0.9254902 , 0.74901961, 0.25882353, 0.0745098 , 0.23137255, 0.1372549 , 0.37647059, 0.89019608, 0.79607843, 0.81176471, 0.8745098, 0.86666667, 0.82745098, 0.8 , 0.84313725, 0.79607843,

 0.87843137, 0.51372549, 0.
 , 0.
 , 0.
 , 0.
 , 0.
 , 0.8627451 , 0.8627451 ,

 , 0.81568627, 0.82352941, 0.88627451, 0.8745098 , 0.81960784, 0.85098039, 0.81568627, 0.84705882, 0.75686275, , 0.80784314, 0.84705882, 0.80392157, 0.8627451 , 0.8745098 , 0.64313725, 0.86666667, 0.61176471, 0.58431373, 0.9372549 , 0.85098039, 0.54901961, 0.95686275, 0.8745098 , 0.54509804, 0.59607843, 0.90196078, 0.84705882, 0.85882353,

 0.81960784, 0.82745098, 0.85098039, 0.
 , 0.

 0.
 , 0.

 , 0.
 , 0.

 , 0.87058824, 0.83529412, 0.80392157, 0.87058824, 0.85882353, 0.58823529, 0.59607843, 0.78823529, 0.67058824, 0.63529412, 0.4745098, 0.47058824, 0.61176471, 0.59215686, 0.50980392, 0.63529412, 0.88235294, 0.83921569, 0.86666667, 0.82745098, 0.82745098, 0.90588235, 0.90196078, 0.67843137, 0.64313725, 0.69019608, 0.60784314, 0.63921569, 0.68235294, 0.55294118, 0.58823529, 0.57647059, 0.6 , 0.61568627, 0.76470588, 0.86666667, 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. , 0.25490196, 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. , 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
 , 0.85098039, 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. , 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. 0.70980392, 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. 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() The accuracy of KNN Running time of KNN 2.00 p = 1p = 286.2 1.75 86.0 1.50 85.8 Accuracy(%) 1.25 100 p = 2 85.6 85.4 0.75 0.50 85.2 0.25 85.0 n_neighbors n neighbors 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 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 0.838888888888889 Finish classifying, it takes 0.009974241256713867s 0.8294612794612795 Finish classifying, it takes 0.009477853775024414s 0.8420875420875421 Finish classifying, it takes 0.00803828239440918s 0.83737373737373 Finish classifying, it takes 0.007039546966552734s 0.82929292929293 Finish classifying, it takes 0.009000062942504883s In [20]: svmscore sum/5 0.8354208754208754 Out[20]: 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' : ['11','12']} #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 = 11, Accuracy = 83.00%, running time = 179.74s max_iter = 200, penalty = 11, Accuracy = 83.00%, running time = 210.94s max iter = 300, penalty = 11, Accuracy = 83.00%, running time = 203.95s max iter = 100, penalty = 12, Accuracy = 82.67%, running time = 43.65s max_iter = 200, penalty = 12, Accuracy = 82.67%, running time = 47.22s max iter = 300, penalty = 12, 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 = 11", "penalty = 12"]) 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 = 11", "penalty = 12"]) plt.xticks(para_grid_lr["max_iter"]) nlt title ("The running time of LR plt.ylabel("time(s)") plt.xlabel("max iter") plt.show() The running time of LR The accuracy of LR 83.00 penalty = I1 penalty = I2 200 82.95 175 82.90 150 82.85 time(s) 125 penalty = I1 penalty = I2 82.80 100 82.75 75 82.70 50 82.65 200 300 100 200 300 max iter max iter 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='12', tol=0.001) LRclassifier.fit(this train x, this train y) prediction = LRclassifier.predict(this test x) score = accuracy score(this test y, prediction) 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 0.83939393939394 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 0.845084175084175 Out[22]: 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, random 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() The accuracy of RF Running time of RF random_state = 0 random_state = 0 random_state = 2 random_state = 2 180 86 170 84 160 Accuracy(%) (%) 150 140 80 130 120 9 max_depth max_depth 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 0.8578114478114479 Out[24]: In [25]: best acc= max(knnscore sum/5,svmscore sum/5, lrscore sum/5, rfscore sum/5) best acc 0.8578114478114479 Out[25]: 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')

y_pred	d_best [1, 1,	1,	0, 2],	dtype=:	int64)	m) m class	ifiers	using :	input as	s data .	from te	st_inpu	t.csv	
# (500	00,)					ns = [' , sep="								