**2022-2 Machine Learning Team Project 1**

**201835516 Ganghoon Jeon**

**201835512 Yunyoung Jang**

**202035362 Sohyun Lee**

**202035369 Jongeun Lee**

|  |
| --- |
| Team project 1 |
| |  | | --- | | **Result** | |  |  |  | | --- | | **Full code** | | **import pandas as pd**  **import numpy as np**  **from sklearn.model\_selection import train\_test\_split**  **from sklearn import datasets**  **#DecisionTrees**  **from sklearn.tree import DecisionTreeClassifier**  **#LogisticRegression**  **from sklearn.linear\_model import LogisticRegression**  **#SVM**  **from sklearn.svm import SVC**  **from sklearn.metrics import classification\_report,confusion\_matrix**  **from sklearn.model\_selection import train\_test\_split**  **#Data Scalers**  **from sklearn.preprocessing import MinMaxScaler**  **from sklearn.preprocessing import StandardScaler**  **from sklearn.preprocessing import RobustScaler**  **from sklearn.preprocessing import Normalizer**  **from sklearn.preprocessing import MaxAbsScaler**  **#module used for Test and validation**  **from sklearn.model\_selection import KFold**  **from sklearn.metrics import accuracy\_score**  **#HyperParameter Setting**  **kFoldMaxCount = 4**  **tree\_depth = 7**  **#Data Load Phase**  **file\_path = "breast-cancer-wisconsin.data"**  **# data preprocessing: Setting target feature and other features**  **classes =['ID', 'ClumpThickness', 'UniformityOfCellSize', 'UniformityOfCellShape', 'MarginalAdhesion', 'SingleEpithelialCellSize','BareNuclei','BlandChromatin', 'NormalNucleoli', 'Mitoses','Class']**  **target = 'Class'**  **#**  **if(kFoldMaxCount <=2):**  **kFoldMaxCount=3**  **def best\_combi():**  **#Decimal Scaling**  **def decimalScaling(df):**  **temp = 10**  **for x in classes:**  **df[x] = pd.to\_numeric(df[x] )**  **for t in df[x]:**  **while(temp<float(t)):**  **temp=temp\*10**  **df[x] = df[x].div(temp)**  **#Log Normalization**  **def logNormalizationlScaling(df):**  **for x in classes:**  **df[x] = pd.to\_numeric(df[x] )**  **df[x] = np.log10(df[x])**  **features = data[data['BareNuclei'] !='?']**  **features.drop(inplace=True, columns =target)**  **label = data[[target]]**  **n\_iter = 0**  **score\_list = []**  **best\_combination = []**  **worst\_combination = []**  **for i in range(0, len(model\_list)):**  **best\_combination.append([])**  **best\_combination[i].append(-1)**  **worst\_combination.append([])**  **worst\_combination[i].append(2)**  **#List of Scalers, if append, add feature\_cached and feature\_cached\_name too.**  **decimalScaler = features.copy()**  **decimalScaling(decimalScaler)**  **#Robust Scaler**  **robustScaler = RobustScaler().fit\_transform(features.copy())**  **robustScaler = pd.DataFrame(robustScaler , columns=classes)**  **minmaxScaler = MinMaxScaler().fit\_transform(features.copy())**  **minmaxScaler = pd.DataFrame(minmaxScaler, columns=classes)**  **standardScaler = StandardScaler().fit\_transform(features.copy())**  **standardScaler = pd.DataFrame(standardScaler, columns=classes)**  **#Normalizer**  **normalizer = Normalizer().fit\_transform(features.copy())**  **normalizer= pd.DataFrame(normalizer , columns=classes)**  **#MaxAbs**  **maxabsScaler = MaxAbsScaler().fit\_transform(features.copy())**  **maxabsScaler= pd.DataFrame(maxabsScaler , columns=classes)**  **logNormalizationScaler = features.copy()**  **logNormalizationlScaling(logNormalizationScaler)**  **feature\_cached\_name = [ 'logNormalizationScaler','Standard Scaler', 'Robust Scaler', 'Normalizer', 'MaxAbsScaler', 'MinMaxScaler']**  **features\_cached = [ logNormalizationScaler,standardScaler, robustScaler,normalizer, maxabsScaler, minmaxScaler]**  **for k in range(2,kFoldMaxCount):**  **kf = KFold(n\_splits = k,shuffle= True)**  **for x in range(0,len(model\_list)):**  **for t in range(0,len(features\_cached)):**  **for train\_idx, test\_idx in kf.split(features\_cached[t], label):**  **n\_iter +=1**  **label\_train = label.iloc[train\_idx]**  **label\_test = label.iloc[test\_idx]**  **X\_train, X\_test = features.iloc[train\_idx], features.iloc[test\_idx]**  **y\_train, y\_test = label.iloc[train\_idx], label.iloc[test\_idx]**  **model\_list[x].fit(X\_train,y\_train.values.ravel())**  **preds = model\_list[x].predict(X\_test)**  **score= accuracy\_score(preds,y\_test)**  **if(score > best\_combination[x][0]):**  **best\_combination[x].clear()**  **best\_combination[x].append(score)**  **best\_combination[x].append(x)**  **best\_combination[x].append(t)**  **best\_combination[x].append(k)**  **if(score < worst\_combination[x][0]):**  **worst\_combination[x].clear()**  **worst\_combination[x].append(score)**  **worst\_combination[x].append(x)**  **worst\_combination[x].append(t)**  **worst\_combination[x].append(k)**  **result = [best\_combination,worst\_combination, feature\_cached\_name]**  **return result**  **#**  **model\_tree\_entropy= DecisionTreeClassifier(criterion='entropy', max\_depth= tree\_depth)**  **model\_tree\_gini= DecisionTreeClassifier(criterion='gini',max\_depth= tree\_depth)**  **model\_logistic = LogisticRegression(solver='lbfgs')**  **model\_svc = SVC(kernel = 'linear')**  **model\_list = [model\_tree\_entropy, model\_tree\_gini, model\_logistic,model\_svc]**  **#**  **data = pd.read\_csv(file\_path, delimiter=",",header= None, names = classes)**  **#Pre-processing Pjase : Set Pre-dropable Columns here. In this case, ID Would be dropped.**  **data.drop(inplace=True, columns =['ID'])**  **classes.remove('ID')**  **classes.remove(target)**  **data.drop\_duplicates(inplace=True)**  **data = data[data['BareNuclei'] !='?']**  **#print(data[data.duplicated()])**  **#print(data[data['BareNuclei'] =='?'])**  **# Setting Target and other dataset.**  **# Analyze the data**  **result = best\_combi()**  **bestNumber = result[0][0]**  **worstNumber = result[0][0]**  **for t in range(0,len(model\_list)):**  **print(f'At model {model\_list[result[0][t][1]]}, with Scaling with Scaler {result[2][result[0][t][2]]}, and At K-Fold with {result[0][t][3]}, the model got best accuracy as {result[0][t][0]}.')**  **print(f'At model {model\_list[result[1][t][1]]}, with Scaling with Scaler {result[2][result[1][t][2]]}, and At K-Fold with {result[1][t][3]}, the model got worst accuracy as {result[1][t][0]}.')**  **if(bestNumber[0] < result[0][t][0]):**  **bestNumber = result[0][t]**  **if(bestNumber[0]==1.0):**  **print('Warning: Accuracy with 1.0 appeared, it could be chance of overfit or wrong K value, or lack of dataset.')**  **if(worstNumber[0] > result[1][t][0]):**  **worstNumber = result[1][t]**  **print(f'At model {model\_list[bestNumber[1]]}, with Scaling with Scaler {result[2][bestNumber[2]]}, and At K-Fold with {bestNumber[3]}, the model got best accuracy compared with every condition as {bestNumber[0]}.')**  **print(f'At model {model\_list[worstNumber[1]]}, with Scaling with Scaler {result[2][worstNumber[2]]}, and At K-Fold with {worstNumber[3]}, the model got worst accuracy compared with every condition as {worstNumber[0]}.')**  **#print(f'So, the best Result is {bestNumber[0] } and when model is {model\_list[ result[1]]} and scaled with {result[2][result[2]]} on K-fold with k= {result[3]}')** |  |  |  |  | | --- | --- | --- | | **Team Contribution percentage & Their roles** | | | | **Name** | **Role Contributed** | **Percentage** | | **Ganghoon jeon** |  | **25%** | | **Yunyoung Jang** |  | **25%** | | **Sohyun Lee** |  | **25%** | | **Jongeun Lee** |  | **25%** | | **Common Role contributed** | **Programmed draft code** | | |