**PHW-1**

**201735871 장건**

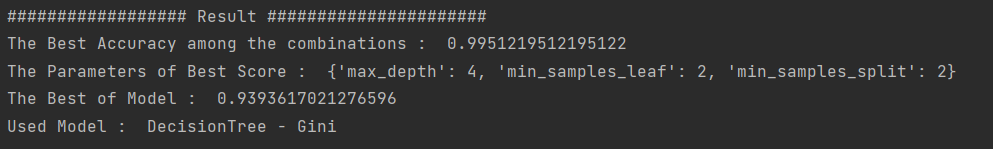
**Source Code:**

import numpy as np  
import pandas as pd  
from sklearn import preprocessing  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.model\_selection import train\_test\_split, GridSearchCV  
from sklearn.linear\_model import LogisticRegression  
from sklearn.svm import SVC  
  
# Standard Scaling Module  
def StandardScaling(data):  
 scaler = preprocessing.StandardScaler()  
 scaled\_df = scaler.fit\_transform(data)  
 scaled\_df = pd.DataFrame(scaled\_df)  
 return scaled\_df  
  
# MinMax Scaling Module  
def MinMaxScaling(data):  
 scaler = preprocessing.MinMaxScaler()  
 scaled\_df = scaler.fit\_transform(data)  
 scaled\_df = pd.DataFrame(scaled\_df)  
 return scaled\_df  
  
# Robust Scaling Module  
def RobustScaling(data):  
 scaler = preprocessing.RobustScaler()  
 scaled\_df = scaler.fit\_transform(data)  
 scaled\_df = pd.DataFrame(scaled\_df)  
 return scaled\_df  
  
# Method to Split Train and Test dataset.  
def SplitData(X, Y, testsize):  
 X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=testsize, shuffle=False, random\_state=0)  
 return X\_train, X\_test, Y\_train, Y\_test  
  
# Preprocessing  
def Preprocessing(df, Encode, Scaling): # Encode = 0 -> Use Ordinal Encoder. If Encode = 2, Use Label Encoder.  
 df.replace({"?": np.nan}, inplace=True) # Scaling = 0 -> Standard Scaler, Scaling = 1 -> MinMax Scaler, Scaling = 3 -> Robust Scaler  
 df.dropna(axis=0, inplace=True)  
 df = df.drop(columns=['ID'])  
 encode1 = preprocessing.OrdinalEncoder()  
 encode2 = preprocessing.LabelEncoder()  
 if Encode == 0 and Scaling == 0 :  
 encode1.fit(df['Bare\_N'][:, np.newaxis])  
 df['Bare\_N'] = encode1.transform(df['Bare\_N'][:, np.newaxis]).reshape(-1)  
 preprocessed\_data\_list.append(StandardScaling(df))  
 if Encode == 0 and Scaling == 1 :  
 encode1.fit(df['Bare\_N'][:, np.newaxis])  
 df['Bare\_N'] = encode1.transform(df['Bare\_N'][:, np.newaxis]).reshape(-1)  
 preprocessed\_data\_list.append(MinMaxScaling(df))  
 if Encode == 0 and Scaling == 2 :  
 encode1.fit(df['Bare\_N'][:, np.newaxis])  
 df['Bare\_N'] = encode1.transform(df['Bare\_N'][:, np.newaxis]).reshape(-1)  
 preprocessed\_data\_list.append(RobustScaling(df))  
 if Encode == 1 and Scaling == 0 :  
 encode2.fit(df['Bare\_N'][:, np.newaxis])  
 df['Bare\_N'] = encode2.transform(df['Bare\_N'][:, np.newaxis]).reshape(-1)  
 preprocessed\_data\_list.append(StandardScaling(df))  
 if Encode == 1 and Scaling == 1 :  
 encode2.fit(df['Bare\_N'][:, np.newaxis])  
 df['Bare\_N'] = encode2.transform(df['Bare\_N'][:, np.newaxis]).reshape(-1)  
 preprocessed\_data\_list.append(MinMaxScaling(df))  
 if Encode == 1 and Scaling == 2 :  
 encode2.fit(df['Bare\_N'][:, np.newaxis])  
 df['Bare\_N'] = encode2.transform(df['Bare\_N'][:, np.newaxis]).reshape(-1)  
 preprocessed\_data\_list.append(RobustScaling(df))  
  
def Make\_Combination(df):  
 Preprocessing(df, 0, 0)  
 Preprocessing(df, 0, 1)  
 Preprocessing(df, 0, 2)  
 Preprocessing(df, 1, 0)  
 Preprocessing(df, 1, 1)  
 Preprocessing(df, 1, 2)  
  
######### Model Building ###########  
def DecisionTree\_Entropy(CrossVal):  
 Tree = DecisionTreeClassifier(criterion='entropy')  
 Tree\_Entropy\_grid = {  
 'max\_depth': [2, 3, 4, 5, 6],  
 'min\_samples\_split': [2, 3, 4],  
 'min\_samples\_leaf': [1, 2, 3, 4]  
 }  
 gsTree\_Entropy = GridSearchCV(Tree, param\_grid=Tree\_Entropy\_grid, cv=CrossVal, refit=True)  
 gsTree\_Entropy.fit(X\_train, Y\_train)  
 TestSetScore\_list.append(gsTree\_Entropy.score(X\_test, Y\_test))  
 Best\_Parameters\_list.append(gsTree\_Entropy.best\_params\_)  
 Best\_Accuracy\_list.append(gsTree\_Entropy.best\_score\_)  
 Model\_list.append('Decision Tree - entropy')  
 print('DecisionTreeClassifier(Entropy) TestSet Score:', gsTree\_Entropy.score(X\_test, Y\_test))  
 print('DecisionTreeClassifier(Entropy) Best Parameter:', gsTree\_Entropy.best\_params\_)  
 print('DecisionTreeClassifier(Entropy) Best Accuracy: {0:.4f}'.format(gsTree\_Entropy.best\_score\_))  
  
def DecisionTree\_Gini(CrossVal):  
 Tree = DecisionTreeClassifier(criterion='gini')  
 Tree\_Gini\_grid = {  
 'max\_depth': [2, 3, 4, 5, 6],  
 'min\_samples\_split': [2, 3, 4],  
 'min\_samples\_leaf': [1, 2, 3, 4]  
 }  
 gsTree\_Gini = GridSearchCV(Tree, param\_grid=Tree\_Gini\_grid, cv=CrossVal, refit=True)  
 gsTree\_Gini.fit(X\_train, Y\_train)  
 TestSetScore\_list.append(gsTree\_Gini.score(X\_test, Y\_test))  
 Best\_Parameters\_list.append(gsTree\_Gini.best\_params\_)  
 Best\_Accuracy\_list.append(gsTree\_Gini.best\_score\_)  
 Model\_list.append('DecisionTree - Gini')  
 print('DecisionTreeClassifier(Gini) TestSet Score:', gsTree\_Gini.score(X\_test, Y\_test))  
 print('DecisionTreeClassifier(Gini) Best Parameter:', gsTree\_Gini.best\_params\_)  
 print('DecisionTreeClassifier(Gini) Best Accuracy: {0:.4f}'.format(gsTree\_Gini.best\_score\_))  
  
def LogisticReg(CrossVal):  
 reg = LogisticRegression()  
 reg\_grid = {  
 'C': [0.001, 0.01, 0.1, 1, 10, 100],  
 'penalty': ['l2']  
 }  
 regCV = GridSearchCV(reg, param\_grid=reg\_grid, cv=CrossVal, refit=True)  
 regCV.fit(X\_train, Y\_train)  
 TestSetScore\_list.append(regCV.score(X\_test, Y\_test))  
 Best\_Parameters\_list.append(regCV.best\_params\_)  
 Best\_Accuracy\_list.append(regCV.best\_score\_)  
 Model\_list.append('Logistic Regression')  
 print('Logistic Regression TestSet Score:', regCV.score(X\_test, Y\_test))  
 print('Logistic Regression Best Parameter:', regCV.best\_params\_)  
 print('Logistic Regression Best Accuracy: {0:.4f}'.format(regCV.best\_score\_))  
  
def SVM(CrossVal):  
 svm = SVC()  
 svm\_grid = {  
 'C': [0.1, 1, 10],  
 'gamma': [ 1, 0.1, 0.01],  
 'kernel': ['linear', 'rbf']  
 }  
 svmCV = GridSearchCV(svm, param\_grid=svm\_grid, cv=CrossVal, refit=True)  
 svmCV.fit(X\_train, Y\_train)  
 TestSetScore\_list.append(svmCV.score(X\_test, Y\_test))  
 Best\_Parameters\_list.append(svmCV.best\_params\_)  
 Best\_Accuracy\_list.append(svmCV.best\_score\_)  
 Model\_list.append('Support Vector Machine')  
 print('Support Vector Machine TestSet Score:', svmCV.score(X\_test, Y\_test))  
 print('Support Vector Machine Best Parameter:', svmCV.best\_params\_)  
 print('Support Vector Machine Best Accuracy: {0:.4f}'.format(svmCV.best\_score\_))  
  
# Load Data  
cols = ['ID', 'Thickness', 'Cell\_size', 'Cell\_shape', 'Adhesion', 'Epi\_Cell\_size', 'Bare\_N', 'Bland', 'Normal\_N',  
 'Mitoses', 'Class']  
data = pd.read\_csv('breast\_cancer\_wisconsin.data', sep=',', encoding='cp949', names=cols)  
data = pd.DataFrame(data)  
  
# Define Global Variable  
preprocessed\_data\_list = []  
Make\_Combination(data)  
TestSetScore\_list = [] # Store scores in the list  
Best\_Parameters\_list = [] # Store Model Best parameters in the list  
Best\_Accuracy\_list = [] # Store Model Best accuracy in the list  
Model\_list = [] # Store which model was used  
Dataset\_Num = 0  
  
# For each Preprocessed Datasets, Fit four models and store the scores.  
for i in preprocessed\_data\_list:  
 data = preprocessed\_data\_list[Dataset\_Num]  
 data = data.astype('int')  
 print('\n###########', Dataset\_Num + 1, 'th Dataset ############')  
 X = data.iloc[:,0:-1] # When you use in the other dataset, Change the location of the target.  
 Y = data.iloc[:,-1]  
 X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, shuffle=False, random\_state=0)  
 DecisionTree\_Entropy(10) # K-Fold Cross Validation  
 DecisionTree\_Gini(10) # K-Fold Cross Validation  
 SVM(10) # K-Fold Cross Validation  
 LogisticReg(10) # K-Fold Cross Validation  
 Dataset\_Num = Dataset\_Num + 1  
  
# Get Final Result (According to Best Score, print same index of lists.)  
print('\n################## Result ######################')  
print('The Best Score among the combinations : ', max(TestSetScore\_list))  
print('The Parameters of Best Score : ', Best\_Parameters\_list[TestSetScore\_list.index(max(TestSetScore\_list))])  
print('The Best Accuracy : ', Best\_Accuracy\_list[TestSetScore\_list.index(max(TestSetScore\_list))])  
print('Used Model : ', Model\_list[TestSetScore\_list.index(max(TestSetScore\_list))])

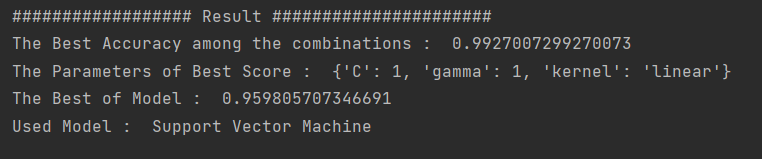
**OutPut :**

**# I tried a lot of combination of test size and Cross Validation too. And these are the good results. Not only on the captured screen, most of the scores selected with the best accuracy are very high.**

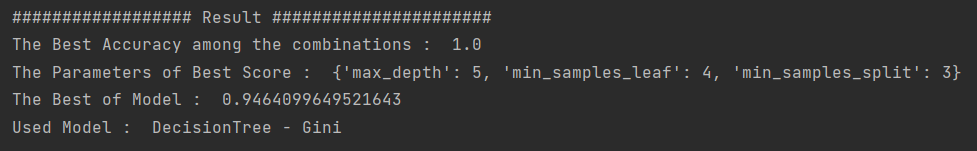
**Test size = 0.3, Cross Validation = 10**



**Test size = 0.2, Cross Validation = 9**



**Test size = 0.1, Cross Validation = 9**



**Accuracy of the test set is 1.0.(100%)**