201935006 고효진

import warnings  
warnings.filterwarnings("ignore")  
import pandas as pd  
from sklearn import preprocessing  
from sklearn.model\_selection import train\_test\_split  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.metrics import accuracy\_score  
from sklearn.feature\_selection import RFECV, VarianceThreshold  
from sklearn.model\_selection import KFold  
from sklearn.metrics import roc\_auc\_score  
from sklearn.svm import SVC  
from sklearn.model\_selection import cross\_val\_score  
from sklearn.model\_selection import GridSearchCV  
  
  
#load Data  
df = pd.read\_csv('breast-cancer-wisconsin.csv')  
df.columns = ['Sample code number','Clump Thickness', 'Uniformity of Cell Size',  
 'Uniformity of Cell Shape', 'Marginal Adhension','Single Epithelial Cell Size',  
 'Bare Nuclei', 'Bland Chromatin','Normal Nucleoli','Mitoses','Class']  
print(df.dtypes)  
print(df.isnull().any(0))  
  
def ordinal\_encode(df):  
 ordinalEncoder = preprocessing.OrdinalEncoder()  
 df= ordinalEncoder.fit\_transform(df)  
 df = pd.DataFrame(df)  
  
  
def oneHot\_encode(df):  
 onehotEncoder = preprocessing.OneHotEncoder()  
 df = onehotEncoder.fit\_transform(df)  
 df = pd.DataFrame(df)  
  
  
  
  
#  
# def ordinal\_scale(df, targetName):  
# ordinalEncoder = preprocessing.OrdinalEncoder()  
# X = pd.DataFrame(df[targetName])  
# ordinalEncoder.fit(X)  
# df[targetName] = pd.DataFrame(ordinalEncoder.transform(X))  
#  
# def oneHot\_scale(df, targetName):  
# oneHotEncoder = preprocessing.OneHotEncoder()  
# X = pd.DataFrame(df[targetName])  
# oneHotEncoder.fit(X)  
# df[targetName] = pd.DataFrame(oneHotEncoder.transform(X))  
  
# def label\_scale(df, targetName):  
# labelEncoder = preprocessing.LabelEncoder()  
# X = pd.DataFrame(df[targetName])  
# labelEncoder.fit(X)  
# df[targetName] = pd.DataFrame(labelEncoder.transform(X))  
  
  
  
  
  
  
  
  
def scale\_module(df, targetName):  
 #Encoding data  
 y\_ordinal=df[targetName]  
 X\_ordinal=df.drop([targetName],1)  
 ordinal\_encode(X\_ordinal)  
 X\_train\_ordinal, X\_test\_ordinal, y\_train\_ordinal, y\_test\_ordinal = train\_test\_split(X\_ordinal,y\_ordinal,random\_state=0)  
  
 y\_oneHot = df[targetName]  
 X\_oneHot = df.drop([targetName], 1)  
 oneHot\_encode(X\_oneHot)  
 X\_train\_oneHot, X\_test\_oneHot, y\_train\_oneHot, y\_test\_oneHot = train\_test\_split(X\_oneHot, y\_oneHot,  
 random\_state=0)  
  
 # Normalization with 4 Scaling methods  
 maxAbsScaler = preprocessing.MaxAbsScaler()  
 minmaxScaler = preprocessing.MinMaxScaler()  
 robustScaler = preprocessing.RobustScaler()  
 standardScaler = preprocessing.StandardScaler()  
  
 #Scaling ordinal encoded data  
 df\_maxAbs\_ordinal\_train = maxAbsScaler.fit\_transform(X\_train\_ordinal)  
 df\_maxAbs\_ordinal\_train = pd.DataFrame(df\_maxAbs\_ordinal\_train, columns=X\_train\_ordinal.columns)  
 df\_maxAbs\_ordinal\_test = maxAbsScaler.fit\_transform(X\_test\_ordinal)  
 df\_maxAbs\_ordinal\_test = pd.DataFrame(df\_maxAbs\_ordinal\_test, columns=X\_test\_ordinal.columns)  
  
 df\_minMax\_ordinal\_train = minmaxScaler.fit\_transform(X\_train\_ordinal)  
 df\_minMax\_ordinal\_train = pd.DataFrame(df\_minMax\_ordinal\_train, columns=X\_train\_ordinal.columns)  
 df\_minMax\_ordinal\_test = minmaxScaler.fit\_transform(X\_test\_ordinal)  
 df\_minMax\_ordinal\_test = pd.DataFrame(df\_minMax\_ordinal\_test, columns=X\_test\_ordinal.columns)  
  
 df\_robust\_ordinal\_train = robustScaler.fit\_transform(X\_train\_ordinal)  
 df\_robust\_ordinal\_train = pd.DataFrame(df\_robust\_ordinal\_train, columns=X\_train\_ordinal.columns)  
 df\_robust\_ordinal\_test = robustScaler.fit\_transform(X\_test\_ordinal)  
 df\_robust\_ordinal\_test = pd.DataFrame(df\_robust\_ordinal\_test, columns=X\_test\_ordinal.columns)  
  
 df\_standard\_ordinal\_train = standardScaler.fit\_transform(X\_train\_ordinal)  
 df\_standard\_ordinal\_train = pd.DataFrame(df\_standard\_ordinal\_train, columns=X\_train\_ordinal.columns)  
 df\_standard\_ordinal\_test = standardScaler.fit\_transform(X\_test\_ordinal)  
 df\_standard\_ordinal\_test = pd.DataFrame(df\_standard\_ordinal\_test, columns=X\_test\_ordinal.columns)  
  
 # Scaling oneHot encoded data  
 df\_maxAbs\_oneHot\_train = maxAbsScaler.fit\_transform(X\_train\_oneHot)  
 df\_maxAbs\_oneHot\_train = pd.DataFrame(df\_maxAbs\_oneHot\_train, columns=X\_train\_oneHot.columns)  
 df\_maxAbs\_oneHot\_test = maxAbsScaler.fit\_transform(X\_test\_oneHot)  
 df\_maxAbs\_oneHot\_test = pd.DataFrame(df\_maxAbs\_oneHot\_test, columns=X\_test\_oneHot.columns)  
  
 df\_minMax\_oneHot\_train = minmaxScaler.fit\_transform(X\_train\_oneHot)  
 df\_minMax\_oneHot\_train = pd.DataFrame(df\_minMax\_oneHot\_train, columns=X\_train\_oneHot.columns)  
 df\_minMax\_oneHot\_test = minmaxScaler.fit\_transform(X\_test\_oneHot)  
 df\_minMax\_oneHot\_test = pd.DataFrame(df\_minMax\_oneHot\_test, columns=X\_test\_oneHot.columns)  
  
 df\_robust\_oneHot\_train = robustScaler.fit\_transform(X\_train\_oneHot)  
 df\_robust\_oneHot\_train = pd.DataFrame(df\_robust\_oneHot\_train, columns=X\_train\_oneHot.columns)  
 df\_robust\_oneHot\_test = robustScaler.fit\_transform(X\_test\_oneHot)  
 df\_robust\_oneHot\_test = pd.DataFrame(df\_robust\_oneHot\_test, columns=X\_test\_oneHot.columns)  
  
 df\_standard\_oneHot\_train = standardScaler.fit\_transform(X\_train\_oneHot)  
 df\_standard\_oneHot\_train = pd.DataFrame(df\_standard\_oneHot\_train, columns=X\_train\_oneHot.columns)  
 df\_standard\_oneHot\_test = standardScaler.fit\_transform(X\_test\_oneHot)  
 df\_standard\_oneHot\_test = pd.DataFrame(df\_standard\_oneHot\_test, columns=X\_test\_oneHot.columns)  
  
 # Alogrithm  
 print("\n------------------------- Using maxAbs scaled dataset -------------------------")  
 max\_score\_maxAbs\_ordinal = algorithm\_module(df\_maxAbs\_ordinal\_train, df\_maxAbs\_ordinal\_test, y\_train\_ordinal, y\_test\_ordinal)  
 print("\n------------------------- Using minMax scaled dataset -------------------------")  
 max\_score\_minMax\_ordinal = algorithm\_module(df\_minMax\_ordinal\_train, df\_minMax\_ordinal\_test, y\_train\_ordinal, y\_test\_ordinal)  
 print("\n------------------------- Using robust scaled dataset -------------------------")  
 max\_score\_robust\_ordinal = algorithm\_module(df\_robust\_ordinal\_train, df\_robust\_ordinal\_test, y\_train\_ordinal, y\_test\_ordinal)  
 print("\n------------------------- Using standard scaled dataset -------------------------")  
 max\_score\_standard\_ordinal = algorithm\_module(df\_standard\_ordinal\_train, df\_standard\_ordinal\_test, y\_train\_ordinal, y\_test\_ordinal)  
  
 # Result  
 max\_score\_result\_oneHot = max(max\_score\_maxAbs\_oneHot, max\_score\_minMax\_oneHot, max\_score\_robust\_oneHot, max\_score\_standard\_oneHot)  
 print("\n\n============================== oneHot encoded Result ==============================")  
 print("Final maximum score for oneHot encoded: %.6f" % max\_score\_result\_oneHot)  
  
 print("\n------------------------- Using maxAbs scaled dataset -------------------------")  
 max\_score\_maxAbs\_oneHot = algorithm\_module(df\_maxAbs\_oneHot\_train, df\_maxAbs\_oneHot\_test, y\_train\_oneHot,  
 y\_test\_oneHot)  
 print("\n------------------------- Using minMax scaled dataset -------------------------")  
 max\_score\_minMax\_oneHot = algorithm\_module(df\_minMax\_oneHot\_train, df\_minMax\_oneHot\_test, y\_train\_oneHot,  
 y\_test\_oneHot)  
 print("\n------------------------- Using robust scaled dataset -------------------------")  
 max\_score\_robust\_oneHot = algorithm\_module(df\_robust\_oneHot\_train, df\_robust\_oneHot\_test, y\_train\_oneHot,  
 y\_test\_oneHot)  
 print("\n------------------------- Using standard scaled dataset -------------------------")  
 max\_score\_standard\_oneHot = algorithm\_module(df\_standard\_oneHot\_train, df\_standard\_oneHot\_test, y\_train\_oneHot,  
 y\_test\_oneHot)  
  
 # Result  
 max\_score\_result\_oneHot = max(max\_score\_maxAbs\_oneHot, max\_score\_minMax\_oneHot, max\_score\_robust\_oneHot,  
 max\_score\_standard\_oneHot)  
 print("\n\n============================== oneHot encoded Result ==============================")  
 print("Final maximum score for oneHot encoded: %.6f" % max\_score\_result\_oneHot)  
  
  
def algorithm\_module(X\_train, X\_test, y\_train, y\_test):  
 #Decision tree classifier  
 dt\_params = {"max\_depth": [2,3,4],  
 "max\_features": randint(1,10),  
 "min\_samples\_leaf":randint(1,10),  
 "criterion": ["gini",'entropy']}  
 tree\_clf = DecisionTreeClassifier()  
 fold5 = KFold(n\_splits=5, shuffle=True, random\_state=0)  
 fold10 = KFold(n\_splits=10, shuffle=True, random\_state=0)  
 grid\_cv = GridSearchCV(tree\_clf, param\_grid=dt\_params, scoring='accuracy', cv=fold5)  
 grid\_cv.fit(X\_train,y\_train)  
 print("Best score of Decision tree is(cv=5): {}".format(grid\_cv.best\_score\_))  
 print("Best parameter of Decision tree is(cv=5): {}".format(grid\_cv.best\_params\_))  
 fold5\_dt\_bestScore=grid\_cv.best\_score  
 grid\_cv = GridSearchCV(tree\_clf, param\_grid=dt\_params, scoring='accuracy', cv=fold10)  
 grid\_cv.fit(X\_train, y\_train)  
 print("Best score of Decision tree is(cv=10): {}".format(grid\_cv.best\_score\_))  
 print("Best parameter of Decision tree is(cv=10): {}".format(grid\_cv.best\_params\_))  
 fold10\_dt\_bestScore=grid\_cv.best\_score  
 max\_dt\_score = max(fold10\_dt\_bestScore,fold5\_dt\_bestScore)  
 #Logistic regression  
 log\_param = {  
 'C': [0.01, 0.1, 1.0, 10.0, 100.0],  
 'max\_iter': [100,1000]  
 }  
 log\_reg = LogisticRegssion()  
 grid\_cv = GridSearchCV(log\_reg, param\_grid=log\_param, scoring='accuracy', cv=fold5)  
 grid\_cv.fit(X\_train, y\_train)  
 print("Best score of Logistic Regression is(cv=5): {}".format(grid\_cv.best\_score\_))  
 print("Best parameter of Logistic Regression is(cv=5): {}".format(grid\_cv.best\_params\_))  
 fold5\_log\_bestScore = grid\_cv.best\_score  
 grid\_cv = GridSearchCV(log\_reg, param\_grid=log\_param, scoring='accuracy', cv=fold10)  
 grid\_cv.fit(X\_train, y\_train)  
 print("Best score of Logistic Regression is(cv=10): {}".format(grid\_cv.best\_score\_))  
 print("Best parameter of Logistic Regression is(cv=10): {}".format(grid\_cv.best\_params\_))  
 fold10\_log\_bestScore = grid\_cv.best\_score  
 max\_log\_score = max(fold10\_log\_bestScore,fold5\_log\_bestScore)  
  
 #SVM  
 svm\_clf = SVC()  
 svm\_params = {  
 'C': [0.01, 0.1, 1.0, 10.0, 100.0],  
 'gamma':[0.01,0.1,1.0,10.0]  
 }  
 grid\_cv = GridSearchCV(svm\_clf, param\_grid=svm\_params, scoring='accuracy', cv=fold5)  
 grid\_cv.fit(X\_train, y\_train)  
 print("Best score of SVM is(cv=5): {}".format(grid\_cv.best\_score\_))  
 print("Best parameter of SVM is(cv=5): {}".format(grid\_cv.best\_params\_))  
 fold5\_svm\_bestScore = grid\_cv.best\_score  
 grid\_cv = GridSearchCV(svm\_clf, param\_grid=svm\_params, scoring='accuracy', cv=fold10)  
 grid\_cv.fit(X\_train, y\_train)  
 print("Best score of SVM is(cv=10): {}".format(grid\_cv.best\_score\_))  
 print("Best parameter of SVM is(cv=10): {}".format(grid\_cv.best\_params\_))  
 fold10\_svm\_bestScore = grid\_cv.best\_score  
 max\_svm\_score = max(fold5\_svm\_bestScore, fold10\_svm\_bestScore)  
  
 max\_score = max(max\_dt\_score, score\_poly, max\_log\_score, max\_svm\_score)  
 return max\_score  
  
print(scale\_module(df, "Class"))