|  |  |
| --- | --- |
| PHW | 201635811 민우혁 |

|  |
| --- |
| PHW code |
| from sklearn.model\_selection import train\_test\_split  from sklearn.tree import DecisionTreeClassifier  from sklearn.linear\_model import LogisticRegression  import pandas as pd  import numpy as np  from sklearn import preprocessing  import warnings ; warnings.filterwarnings('ignore')  import seaborn as sns  from sklearn.metrics import accuracy\_score, roc\_auc\_score, f1\_score  from sklearn import preprocessing  from sklearn.model\_selection import cross\_val\_score, StratifiedKFold  skf = StratifiedKFold(n\_splits=10)  from sklearn.linear\_model import LogisticRegression  from sklearn.svm import SVC  from sklearn.model\_selection import KFold  from sklearn.model\_selection import cross\_val\_score  from sklearn.model\_selection import GridSearchCV  import sys  ################  #make result table  score\_sample = {'Scaler':["Sample"], 'Encoder':["Sample"], 'Model':["Sample"],'Best\_para':["Sample"], "Score":[1]}  score\_results = pd.DataFrame(score\_sample)    #for scale and encorde  class PreprocessPipeline():      def \_\_init\_\_(self, num\_process, cat\_process, verbose=False):          #super(PreprocessPipeline, self).\_\_init\_\_()          self.num\_process = num\_process          self.cat\_process = cat\_process          #for each type          if num\_process == 'standard':              self.scaler = preprocessing.StandardScaler()          elif num\_process == 'minmax':              self.scaler = preprocessing.MinMaxScaler()          elif num\_process == 'maxabs':              self.scaler = preprocessing.MaxAbsScaler()          elif num\_process == 'robust':              self.scaler = preprocessing.RobustScaler()          else:              raise ValueError("Supported 'num\_process' : 'standard','minmax','maxabs','robust'")          if cat\_process == 'onehot':              self.encoder = preprocessing.OneHotEncoder(sparse=False, handle\_unknown='ignore')          elif cat\_process == 'ordinal':              self.encoder = preprocessing.OrdinalEncoder()          else:              raise ValueError("Supported 'cat\_process' : 'onehot', ordinal'")          self.verbose=verbose            #do Preprocess      def process(self, X, Xt):          X\_cats = X.select\_dtypes(np.object).copy()          X\_nums = X.select\_dtypes(exclude=np.object).copy()          Xt\_cats = Xt.select\_dtypes(np.object).copy()          Xt\_nums = Xt.select\_dtypes(exclude=np.object).copy()          if self.verbose:              print(f"Categorica Colums : {list(X\_cats)}")              print(f"Numeric Columns : {list(X\_nums)}")          if self.verbose:              print(f"Categorical cols process method : {self.cat\_process.upper()}")          X\_cats = self.encoder.fit\_transform(X\_cats)          Xt\_cats = self.encoder.transform(Xt\_cats)          if self.verbose:              print(f"Numeric columns process method : {self.num\_process.upper()}")          X\_nums = self.scaler.fit\_transform(X\_nums)          Xt\_nums = self.scaler.transform(Xt\_nums)          X\_processed = np.concatenate([X\_nums, X\_cats], axis=-1)          Xt\_processed = np.concatenate([Xt\_nums, Xt\_cats], axis=-1)            return X\_processed, Xt\_processed  # do process on I want  class AutoProcess():      def \_\_init\_\_(self, verbose=False):            self.pp = PreprocessPipeline          self.verbose= verbose        def run(self, X, Y, Xt, Yt):          methods = []          scores = []          print(X.shape, Xt.shape)            for num\_process in ['standard','robust','minmax','maxabs']:              for cat\_process in ['onehot','ordinal']:                  if self.verbose:                      print("\n------------------------------------------------------\n")                      print(f"Numeric Process : {num\_process}")                      print(f"Categorical Process : {cat\_process}")                  methods.append([num\_process, cat\_process])                  pipeline = self.pp(num\_process=num\_process, cat\_process=cat\_process)                    X\_processed, Xt\_processed = pipeline.process(X, Xt)                  #Classifier part                  for model in ['gini','entropy','svc']:                      if self.verbose:                          print(f"\nClassifier model: {model}")                      if model =='gini':                          param\_grid = {'max\_depth' : [3,5,7,10],                                        "min\_samples\_leaf":[1, 2, 3, 4, 5, 6, 7, 8, 9],                                        "min\_samples\_split":[2, 3, 4, 5, 6, 7, 8, 9, 10]}                          clf = DecisionTreeClassifier()                      elif model =='svc':                          param\_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100],                                       'gamma': [0.001, 0.01, 0.1, 1, 10, 100] }                          clf = SVC()                      else:                          param\_grid = {'max\_depth' : [3,5,7,10],                                        "min\_samples\_leaf":[1, 2, 3, 4, 5, 6, 7, 8, 9],                                        "min\_samples\_split":[2, 3, 4, 5, 6, 7, 8, 9, 10]}                          clf = DecisionTreeClassifier(criterion = "entropy")                      grid\_search = GridSearchCV(clf,param\_grid,cv=5)                      grid\_search.fit(X\_processed,Y)                      score\_results.loc[len(score\_results)] = [num\_process, cat\_process, model, grid\_search.best\_params\_, grid\_search.best\_score\_]                      clf = clf.fit(X\_processed, Y)                      predict = clf.predict(Xt\_processed)                      score1 = accuracy\_score(Yt, predict)                      #print("Score: ", score1)                      score = cross\_val\_score(clf, Xt\_processed, Yt, cv=kfold, n\_jobs=1, scoring='accuracy')                      score2 = np.mean(score)                      #print("Score with using kfold: ", score2)                      score\_results.loc[len(score\_results)] = [num\_process, cat\_process, model, np.NaN ,score1]                      score\_results.loc[len(score\_results)] = [num\_process, cat\_process, model+" with kfold",np.NaN, score2]                    #logistic Regression part                  for model in ['logistic']:                      if self.verbose:                          print(f"\nRegression model: {model}")                        param\_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100],                                    'penalty': ['l1', 'l2']}                      grid\_search =GridSearchCV(LogisticRegression(), param\_grid, cv=5)                      grid\_search.fit(X\_processed, Y)                        #lr = LogisticRegression().fit(X\_processed, Y)                      #predict = lr.predict(Xt\_processed)                      #score = round(accuracy\_score(Yt, predict.round())\*100, 2)                      score\_results.loc[len(score\_results)] = [num\_process, cat\_process, model, grid\_search.best\_params\_, grid\_search.best\_score\_]          return  kfold = KFold(5, True, 1)    df = pd.read\_csv('E:\PythonWorkSpace\s2\lab1\\breast-cancer-wisconsin.data', sep=',',names=['ID','CT','UC Size','UC Shape','MA','SECS','BN','BC','NN','Mitoses','Class'],header=None)  #print(df)  #check null  print(df.isnull().sum())  #check data type  print(df.dtypes)  df=df[df.BN != '?']  df=df.astype({'BN':int})  ##print(df.dtypes)  df = df.drop(['ID'],axis =1)  print(df)  #Separate taget and feature  X =df.iloc[:,0:9]  Y = df.iloc[:,[9]]  print(X)  print(Y)  #split train and test  X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.33, random\_state=0)  autoprocess = AutoProcess(verbose=True)  autoprocess.run(X\_train, Y\_train, X\_test, Y\_test)  pd.set\_option('display.max\_row', 100)  print(score\_results.sort\_values(by=['Score'], axis=0,ascending=False))  sys.stdout = open('score result.txt', 'w')  print(score\_results.sort\_values(by=['Score'], axis=0,ascending=False))  sys.stdout.close() |
| Result |
|  |