PHW1

201636417 심우석

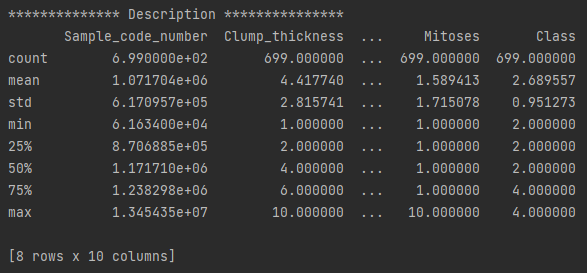
Source code:

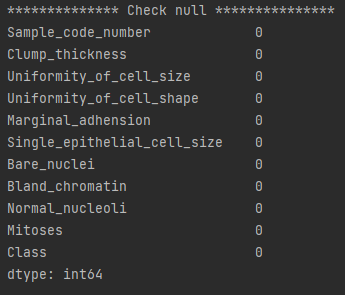
*# Import Class Libraries*import pandas as pd  
import numpy as np  
from scipy.stats import stats  
from sklearn.model\_selection import GridSearchCV, train\_test\_split  
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder, LabelEncoder  
from sklearn.preprocessing import StandardScaler, RobustScaler, MinMaxScaler, MaxAbsScaler  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.linear\_model import LogisticRegression  
from sklearn.svm import SVC  
from imblearn.over\_sampling import SMOTE  
from sklearn.feature\_selection import SelectKBest, f\_classif, f\_regression  
  
*##############################################################################################  
# FindBestAccruacy (X, y, scale\_col, encode\_col, scalers = None, encoders = None,  
# models = None, model\_param = None, cv = None, n\_jobs = None)  
# Description = When parameters are put in, the highest accuracy and the best model are output  
# Input = X: Data Feature  
# Y: Data Target  
# Scale\_col: columns to scaled  
# Encode\_col: columns to encode  
# Scalers: list of scalers  
# None: [StandardScaler(), RobustScaler(), MinMaxScaler(), MaxAbsScaler()]  
# If you want to scale other ways, then put the scaler in list.  
#  
# Encoders: list of encoders  
# None: [OrdinalEncoder(), OneHotEncoder(), LabelEncoder()]  
# If you want to encode other ways, then put the encoder in list.  
#  
# Models: list of models  
# None: [DecisionTreeClassifier(criterion=’entropy’), DecisionTreeClassifier(criterion=’gini’),  
# LogisticRegression(), SVC()]  
# If you want to fit other ways, then put the (Classification)model in list.  
#  
# Model\_param: list of model’s hyperparameter  
# DecisionTreeClassifier(criterion=’entropy’)’s None: [random\_state: None (int), max\_depth: None (int),  
# Max\_features: None (auto, sqrt, log2), max\_leaf\_node: None (int)]  
# DecisionTreeClassifier(criterion=’gini’)’s None: [random\_state: None (int), max\_depth: None (int),  
# Max\_features: None (auto, sqrt, log2), max\_leaf\_node: None (int)]  
# LocisticRegression()’s None: [penalty: None (l2), random\_state: None (int),  
# C: None (float), Solver: None (lbfgs, sag or saga), max\_iter: None (int)]  
# SVC()’s None: [kernel: None (linear, rbf, sigmoid), random\_state: None(int),  
# C: None (float), gamma: None (int)]  
# If you want to set other ways, then put the hyperparameter in list.  
#  
# Cv = K-Fold cross validation’s K  
# None: 5  
# N\_jobs = number of jobs to run in parallel. Training the estimator and computing score are  
# parallelized over the cross-validation splits  
# None: 1  
  
# Output = Best Model, Best Accuracy*def FindBestAccruacy(X, y, scale\_col, encode\_col, scalers=None, encoders=None,  
 models=None, model\_param=None, cv=None, n\_jobs=None):  
  
 *# Set Encoder* if encoders is None:  
 encode = [OrdinalEncoder(), OneHotEncoder(), LabelEncoder()]  
 else: encode = encoders  
  
 *# Set Scaler* if scalers is None:  
 scale = [StandardScaler(), MinMaxScaler(), MaxAbsScaler(), RobustScaler()]  
 else: scale = scalers  
  
 *# Set Model* if models is None:  
 model = [DecisionTreeClassifier(criterion='entropy'),  
 DecisionTreeClassifier(criterion='gini'),  
 LogisticRegression(), SVC()]  
 else: model = models  
  
 *# Set Hyperparameter* if model\_param is None:  
 *# DecisionTreeClassifier('Entropy')* parameter = [{'criterion':['entropy'], 'random\_state':[1, 2, 5, 10, 20], 'max\_depth':[4, 6, 8, 10],  
 'max\_features':["auto", "sqrt", "log2"], 'max\_leaf\_nodes':[2, 4, 6]},  
 *# DecisionTreeClassifier('Gini')* {'criterion':['gini'], 'random\_state':[1, 2, 5, 10, 20], 'max\_depth':[4, 6, 8, 10],  
 'max\_features':["auto", "sqrt", "log2"], 'max\_leaf\_nodes':[2, 4, 6]},  
 *# LogisticRegression()* {'random\_state':[1, 2, 5, 10, 20], 'penalty':['l2'], 'max\_iter':[30000, 50000, 100000],  
 'C':[0.01, 0.1, 1.0, 10.0, 100.0], 'solver':["newton-cg", "lbfgs", "sag", "saga"]},  
 *# SVC()* {'random\_state':[1, 2, 5, 10, 20], 'kernel':['linear', 'rbf', 'sigmoid'],  
 'C':[0.01, 0.1, 1.0, 10.0, 100.0], 'gamma':['scale', 'auto']}]  
 else: parameter = model\_param  
  
 *# Set CV(cross validation)* if cv is None:  
 setCV = 5  
 else: setCV = cv  
  
 *# Set n\_jobs* if n\_jobs is None:  
 N\_JOBS = -1  
 else: N\_JOBS = n\_jobs  
  
 best\_score = 0  
 best\_combination = {}  
 param = {}  
  
 *# SMOTE - Synthetic minority oversampling technique (Fixing the imbalanced data)* target = y  
 smote = SMOTE(random\_state=len(X))  
 X, y = smote.fit\_resample(X, y)  
  
 *####################################################################  
 # Iterate* for i in scale:  
 for j in encode:  
  
 *# Scaling* df\_scaled = pd.DataFrame(i.fit\_transform(X[scale\_col]))  
  
 *# Encoding* if encode\_col is not None:  
 if j == OrdinalEncoder():  
 df\_encoded = j.fit\_transform(X[encode\_col])  
 df\_prepro = pd.concat([df\_scaled, df\_encoded], axis=1)  
  
 else:  
 dum = pd.DataFrame(pd.get\_dummies(X[encode\_col]))  
 df\_prepro = pd.concat([df\_scaled, dum], axis=1)  
  
 else:  
 df\_prepro = df\_scaled  
  
 *# Feature Selection Using the Select KBest (K = 6)* selectK = SelectKBest(score\_func=f\_regression, k=6).fit(df\_prepro, y.values.ravel())  
 cols = selectK.get\_support(indices=True)  
 df\_selected = df\_prepro.iloc[:, cols]  
  
 for z in model:  
  
 *# Split train, testset* X\_train, X\_test, y\_train, y\_test = train\_test\_split(df\_selected, y)  
  
 *# Set hyperparameter* if model\_param is None:  
 if model[0] is z:  
 param = parameter[0]  
 elif model[1] is z:  
 param = parameter[1]  
 elif model[2] is z:  
 param = parameter[2]  
 elif model[3] is z:  
 param = parameter[3]  
  
 else: param = parameter  
  
 *# Modeling(Using the GridSearchCV)* grid\_search = GridSearchCV(estimator=z, param\_grid=param, n\_jobs=N\_JOBS, cv=setCV)  
 grid\_search.fit(X\_train, y\_train.values.ravel())  
 score = grid\_search.score(X\_test, y\_test)  
  
 *# Find Best Score* if best\_score == 0 or best\_score < score:  
 best\_score = score  
 best\_combination['scaler'] = i  
 best\_combination['encoder'] = j  
 best\_combination['model'] = z  
 best\_combination['parameter'] = grid\_search.best\_params\_  
  
 *# Print them* print("Best Score = {:0.6f}".format(best\_score), "")  
 print("Best Combination, Model {}, Encoder {}, Scaler {}".  
 format(best\_combination['model'], best\_combination['encoder'], best\_combination['scaler']))  
 print("Hyperparameter {}".format(best\_combination['parameter']))  
 return  
  
*#########################################################################################  
# Read the dataset  
# Dataset = The Wisconsin Cancer Dataset  
# Feature = 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  
# Target = Class  
  
# Number of Dataset = 699  
# Numerical value = 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  
# Categorical value = Class*feature\_label = ['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']  
target\_label = ['Class']  
  
df = pd.read\_csv("breast-cancer-wisconsin.data", header=None,  
 names=feature\_label+target\_label)  
  
*# Print Cancer data's information  
# print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Cancer \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
# print(df.head())  
  
# print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\* Description \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
# print(df.describe())  
  
# Check null value  
# print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\* Check null \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
# print(df.isna().sum())  
  
# Drop Sample code number(ID)*df = df.drop(["Sample\_code\_number"], axis=1)  
feature\_label.remove("Sample\_code\_number")  
  
*# Cleaning Dirty data*df = df.replace('?', np.NaN)  
df = df.fillna(method='ffill')  
  
*# Casting type Bare\_nuclei column*df['Bare\_nuclei'] = pd.to\_numeric(df['Bare\_nuclei'])  
  
*# Cleaning Dirty Data, Remove outliers using Z score (>, < 3)  
# Remove Outliers with z-score  
# Description = Use the z-score to handle outlier over mean +- 3SD  
# Input = dataframe's column  
# Output = index*def find\_outliers(col):  
 z = np.abs(stats.zscore(col))  
 idx\_outliers = np.where(z > 3, True, False)  
 return pd.Series(idx\_outliers, index=col.index)  
  
for n in range(10):  
 idx = None  
 idx = find\_outliers(df.iloc[:, n])  
 df = df.loc[idx == False]  
  
  
*# print("\n\*\*\*\*\*\*\*\* Removed Outlier \*\*\*\*\*\*")  
# print(df.describe())  
  
# Set X, y data*y\_data = df.loc[:, target\_label]  
X\_data = df.drop(target\_label, axis=1)  
  
*# Auto Find Best Accuracy*print("Auto Find Best Accuracy")  
FindBestAccruacy(X\_data, y\_data, scale\_col=feature\_label, encode\_col=None)  
  
*# Setting some values*print("\n\n\nSetting some values")  
FindBestAccruacy(X\_data, y\_data, scale\_col=feature\_label, encode\_col=None,  
 scalers=[StandardScaler(), MinMaxScaler()], encoders=[OneHotEncoder(), LabelEncoder()],  
 models=[DecisionTreeClassifier(criterion='entropy'), DecisionTreeClassifier(criterion='gini')],  
 model\_param={'random\_state':[1, 5, 10], 'max\_depth':[4, 6, 8],  
 'max\_features':["auto", "sqrt", "log2"], 'max\_leaf\_nodes':[2, 4, 6]})

Result:

텍스트이(가) 표시된 사진

자동 생성된 설명



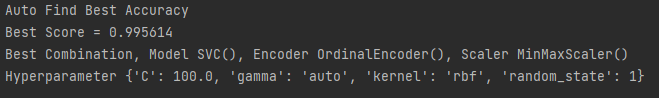


텍스트이(가) 표시된 사진

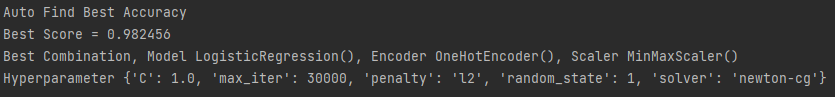
자동 생성된 설명

As a result of automatically finding the optimal combination after entering only the label values of the features to be scaled and encoded, X and y.

Try 1



Try 2



Try 3

텍스트이(가) 표시된 사진

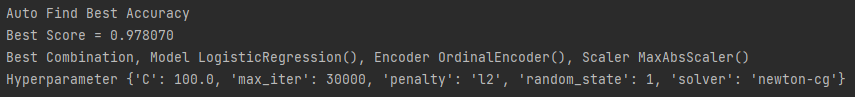
자동 생성된 설명

Try 4

텍스트이(가) 표시된 사진

자동 생성된 설명

Try 5



The combination that was chosen that most is Model – SVC(), Encoder – OnehotEncoder(), Scaler – Robust, MinMaxScaler().

The most accurate one is 99.561% !!!

data preprocessing was detailed, so good results were obtained generally.

The value that received the result after setting the specific values.

