**PHW1**

201835486 이민서

1. **Prepare the data**

In the dataset, the data stored like this; 1000025,5,1,1,1,2,1,3,1,1,2(Sample code number, Clump Thickness, Uniformity of Cell Size, Uniformity of Cell Shape, Marginal Adhesion, Single Epithelial Cell Size, Bare Nuclei, Bland Chromatin, Normal Nucleoli, Mitoses).

텍스트이(가) 표시된 사진

자동 생성된 설명

To prepare data, read the data file with read\_csv and put columns names. Then, store the data in df data frame.

#read data  
df=pd.read\_csv('C://Users/lmslm/Desktop/breast-cancer-wisconsin.data',names=['ID','Clump Thickness','Uniformity of Cell Size','Uniformity of Cell Shape','Marginal Adhesion','Single Epithelia Cell Size','Bare Nuclei','Bland Chromatin','Normal Nucleoli','Mitoses','Classes'])

1. **Preprocessing**

In the dataset, there are 16 instances in Groups 1 to 6 that contain a single missing(i.e.,unavailable) attribute value, denoted by “?”. To handle missing value, we have to replace “?” to Nan.

# ? to nan  
df.replace({'?':np.nan},inplace=True)

To check the number of Nan, use this code.

# check the number of NaN  
print('<Before data preprocessing-Nan>')  
print(df.isna().sum())

텍스트이(가) 표시된 사진

자동 생성된 설명

Fill the Nan using bfill

# fill the Nan using bfill  
df.fillna(method='bfill',inplace=True)

텍스트이(가) 표시된 사진

자동 생성된 설명

Let’s split data to features and target. “Classes” is target.

X=df.drop(columns='Classes') # features  
y=df['Classes'] # target

Then, let’s check the target data distribution.

# count the number of each class  
print(df['Classes'].value\_counts()) # data imbalance



You can see the data is unbalanced. To solve data imbalance, use SMOTE.

from imblearn.over\_sampling import SMOTE

# Resolving the data imbalance  
smote = SMOTE(random\_state=0)  
X\_resampled,y\_resampled=smote.fit\_resample(X,y)  
print("After OverSampling, counts of label '2': {}".format(sum(y\_resampled==2)))  
print("After OverSampling, counts of label '4': {}".format(sum(y\_resampled==4)))

텍스트, 오렌지이(가) 표시된 사진

자동 생성된 설명

Now, data is balanced.

We don’t have categorical data, so we don’t need to encode the data.

Lastly, we have to scale. We will use StandardScaler(), MinMaxScaler(), RobustScaler(), MaxAbsScaler() to get the best accuracy.

1. **Modeling**

To get the best accuracy, we have to try all the combinations. Send scalers, models and the lit of k candidates(for K-fold cross validation) to find\_best.

# scalers,encoders,models  
scalers=[['standard',preprocessing.StandardScaler()],['minmax',preprocessing.MinMaxScaler()],['robust',preprocessing.RobustScaler()],['maxabs',preprocessing.MaxAbsScaler()]]  
models=[['decision',DecisionTreeClassifier()],['logistic\_regression',LogisticRegression()],['svm',SVC()]]  
kfold=[5,10,15,20]  
  
# get the best  
find\_best(X,y,scalers,encoders,models,kfold)

# find the combination that has the best accuracy  
def find\_best(X,y,scalers,encoders,models,kfold):  
 accuracy=[]  
 combination\_case = []  
 for scaler in range(len(scalers)):  
 X=scaling(scalers[scaler][1],X)  
 for model in range(len(models)):  
 for k in kfold:  
 hyperparameter\_tuning(X,y,models[model],k,accuracy)  
 combination\_case.append([scalers[scaler][0],models[model][0],k])  
  
 # show all the combination cases  
 print('====combination cases====')  
 for i in range(len(combination\_case)):  
 print(i+1,':',combination\_case[i])  
 # show all the accuracy  
 print('====accuracy====')  
 for i in range(len(accuracy)):  
 print(i+1,':',accuracy[i])  
 # show the best combination  
 print('<Best Combination>')  
 print('accuracy: ',max(accuracy),' scaler: ',combination\_case[accuracy.index(max(accuracy))][0],' model: ',combination\_case[accuracy.index(max(accuracy))][1],' K: ',combination\_case[accuracy.index(max(accuracy))][2])  
  
def hyperparameter\_tuning(X,y,model,k,accuracy):  
 print(X)  
 if model[0]=='decision':  
 #parameters  
 param={  
 'criterion': ['gini', 'entropy'],  
 'max\_depth': range(1, 10),  
 'min\_samples\_split': range(2, 10),  
 'min\_samples\_leaf': range(1, 5)  
 }  
 elif model[0]=='logistic\_regression':  
 #parameters  
 param={  
 'penalty':['none','l2'],  
 'C':np.logspace(0,4,10)  
 }  
 elif model[0]=='svm':  
 #parameters  
 param={  
 'C': scipy.stats.expon(scale=100),  
 'gamma': scipy.stats.expon(scale=.1),  
 'kernel': ['rbf'],  
 'class\_weight':['balanced', None]  
 }  
  
  
  
 cv = KFold(n\_splits=k,shuffle=True,random\_state=1)  
  
 for train\_ix,test\_ix in cv.split(X):  
 X\_train,X\_test=X.iloc[train\_ix],X.iloc[test\_ix]  
 y\_train,y\_test=y.iloc[train\_ix],y.iloc[test\_ix]  
  
 random = RandomizedSearchCV(estimator=model[1], param\_distributions=param, n\_iter=50, cv=3, verbose=2,  
 random\_state=42, n\_jobs=-1)  
 result=random.fit(X\_train,y\_train)  
  
 # get the best performing model if on the whole training set  
 best\_model=result.best\_estimator\_  
 # evaluate model on the hold out dataset  
 yhat=best\_model.predict(X\_test)  
 # evaluate the model  
 acc=accuracy\_score(y\_test,yhat)  
 # store accuracy  
 accuracy.append(acc)

def scaling(scaler,X):  
 scaled=scaler.fit\_transform(X)  
 scaled=pd.DataFrame(scaled)  
  
 return scaled

* Full Code

import pandas as pd  
import numpy as np  
import scipy  
from matplotlib import cm  
from sklearn.metrics import accuracy\_score  
from sklearn.preprocessing import LabelEncoder  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.metrics import classification\_report  
import seaborn as sns  
from sklearn import preprocessing  
import matplotlib.pyplot as plt  
from sklearn.model\_selection import RandomizedSearchCV  
from imblearn.over\_sampling import SMOTE  
from sklearn.linear\_model import LogisticRegression  
from sklearn.svm import SVC  
from sklearn.model\_selection import KFold  
import warnings  
warnings.filterwarnings(action="ignore")  
  
# find the combination that has the best accuracy  
def find\_best(X,y,scalers,encoders,models,kfold):  
 accuracy=[]  
 combination\_case = []  
 for scaler in range(len(scalers)):  
 X=scaling(scalers[scaler][1],X)  
 for model in range(len(models)):  
 for k in kfold:  
 hyperparameter\_tuning(X,y,models[model],k,accuracy)  
 combination\_case.append([scalers[scaler][0],models[model][0],k])  
   
 # show all the combination cases  
 print('====combination cases====')  
 for i in range(len(combination\_case)):  
 print(i+1,':',combination\_case[i])  
 # show all the accuracy  
 print('====accuracy====')  
 for i in range(len(accuracy)):  
 print(i+1,':',accuracy[i])  
 # show the best combination  
 print('<Best Combination>')  
 print('accuracy: ',max(accuracy),' scaler: ',combination\_case[accuracy.index(max(accuracy))][0],' model: ',combination\_case[accuracy.index(max(accuracy))][1],' K: ',combination\_case[accuracy.index(max(accuracy))][2])  
  
def hyperparameter\_tuning(X,y,model,k,accuracy):  
 print(X)  
 if model[0]=='decision':  
 #parameters  
 param={  
 'criterion': ['gini', 'entropy'],  
 'max\_depth': range(1, 10),  
 'min\_samples\_split': range(2, 10),  
 'min\_samples\_leaf': range(1, 5)  
 }  
 elif model[0]=='logistic\_regression':  
 #parameters  
 param={  
 'penalty':['none','l2'],  
 'C':np.logspace(0,4,10)  
 }  
 elif model[0]=='svm':  
 #parameters  
 param={  
 'C': scipy.stats.expon(scale=100),  
 'gamma': scipy.stats.expon(scale=.1),  
 'kernel': ['rbf'],  
 'class\_weight':['balanced', None]  
 }  
  
  
  
 cv = KFold(n\_splits=k,shuffle=True,random\_state=1)  
  
 for train\_ix,test\_ix in cv.split(X):  
 X\_train,X\_test=X.iloc[train\_ix],X.iloc[test\_ix]  
 y\_train,y\_test=y.iloc[train\_ix],y.iloc[test\_ix]  
  
 random = RandomizedSearchCV(estimator=model[1], param\_distributions=param, n\_iter=50, cv=3, verbose=2,  
 random\_state=42, n\_jobs=-1)  
 result=random.fit(X\_train,y\_train)  
  
 # get the best performing model if on the whole training set  
 best\_model=result.best\_estimator\_  
  
 # evaluate model on the hold out dataset  
 yhat=best\_model.predict(X\_test)  
 # evaluate the model  
 acc=accuracy\_score(y\_test,yhat)  
 # store accuracy  
 accuracy.append(acc)  
  
def scaling(scaler,X):  
 scaled=scaler.fit\_transform(X)  
 scaled=pd.DataFrame(scaled)  
  
 return scaled  
  
#read data  
df=pd.read\_csv('C://Users/lmslm/Desktop/breast-cancer-wisconsin.data',names=['ID','Clump Thickness','Uniformity of Cell Size','Uniformity of Cell Shape','Marginal Adhesion','Single Epithelia Cell Size','Bare Nuclei','Bland Chromatin','Normal Nucleoli','Mitoses','Classes'])  
print(df)  
  
# ? to nan  
df.replace({'?':np.nan},inplace=True)  
  
# check the number of NaN  
print('<Before data preprocessing-Nan>')  
print(df.isna().sum())  
  
# fill the Nan using bfill  
df.fillna(method='bfill',inplace=True)  
  
# check the number of NaN  
print('<After data preprocessing-Nan>')  
print(df.isna().sum())  
  
# split data to features and target  
X=df.drop(columns='Classes') # features  
y=df['Classes'] # target  
  
# count the number of each class  
print(df['Classes'].value\_counts()) # data imbalance  
  
# Resolving the data imbalance  
smote = SMOTE(random\_state=0)  
X\_resampled,y\_resampled=smote.fit\_resample(X,y)  
print("After OverSampling, counts of label '2': {}".format(sum(y\_resampled==2)))  
print("After OverSampling, counts of label '4': {}".format(sum(y\_resampled==4)))  
  
# scalers,encoders,models  
scalers=[['standard',preprocessing.StandardScaler()],['minmax',preprocessing.MinMaxScaler()],['robust',preprocessing.RobustScaler()],['maxabs',preprocessing.MaxAbsScaler()]]  
models=[['decision',DecisionTreeClassifier()],['logistic\_regression',LogisticRegression()],['svm',SVC()]]  
kfold=[5,10,15,20]  
  
# get the best  
find\_best(X,y,scalers,encoders,models,kfold)

* Output

Ex)

* Show the train/test dataset

텍스트, 실외, 명판이(가) 표시된 사진

자동 생성된 설명

텍스트, 실외, 명판이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

* 텍스트이(가) 표시된 사진

  자동 생성된 설명Randomized Searching
* Show all the combination cases

텍스트, 명판이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

* Show all the accuracy

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

* Show the best combination

