<https://dacon.io/competitions/official/9565/data/>

train.csv - 의료기관이 폐업했는지 여부를 포함하여 최근 2개년의 재무정보와 병원 기본정보

test.csv - 폐업 여부를 제외하고 train.csv와 동일

sample\_submission.csv - inst\_id와 open과 close를 예측하는 OC 두개의 열로 구성. OC의 값은 open 예측일 경우 1, close 예측일 경우 0.

inst\_id - 각 파일에서의 병원 고유 번호

OC – 영업/폐업 분류, 2018년 폐업은 2017년 폐업으로 간주함

sido – 병원의 광역 지역 정보

sgg – 병원의 시군구 자료

openDate – 병원 설립일

bedCount - 병원이 갖추고 있는 병상의 수

instkind – 병원, 의원, 요양병원, 한의원, 종합병원 등 병원의 종류

·        종합병원 : 입원환자 100명 이상 수용 가능

·        병원 : 입원 환자 30명 이상 100명 미만 수용 가능

·        의원 : 입원 환자 30명 이하 수용 가능

·        한방 병원(한의원) : 침술과 한약으로 치료하는 의료 기관.

revenue1 – 매출액, 2017(회계년도)년 데이터를 의미함

salescost1 – 매출원가, 2017(회계년도)년 데이터를 의미함

sga1 - 판매비와 관리비, 2017(회계년도)년 데이터를 의미함

salary1 – 급여, 2017(회계년도)년 데이터를 의미함

noi1 – 영업외수익, 2017(회계년도)년 데이터를 의미함

noe1 – 영업외비용, 2017(회계년도)년 데이터를 의미함

Interest1 – 이자비용, 2017(회계년도)년 데이터를 의미함

ctax1 – 법인세비용, 2017(회계년도)년 데이터를 의미함

Profit1 – 당기순이익, 2017(회계년도)년 데이터를 의미함

liquidAsset1 – 유동자산, 2017(회계년도)년 데이터를 의미함

quickAsset1 – 당좌자산, 2017(회계년도)년 데이터를 의미함

receivableS1 - 미수금(단기), 2017(회계년도)년 데이터를 의미함

inventoryAsset1 – 재고자산, 2017(회계년도)년 데이터를 의미함

nonCAsset1 – 비유동자산, 2017(회계년도)년 데이터를 의미함

tanAsset1 – 유형자산, 2017(회계년도)년 데이터를 의미함

OnonCAsset1 - 기타 비유동자산, 2017(회계년도)년 데이터를 의미함

receivableL1 – 장기미수금, 2017(회계년도)년 데이터를 의미함

debt1 – 부채총계, 2017(회계년도)년 데이터를 의미함

liquidLiabilities1 – 유동부채, 2017(회계년도)년 데이터를 의미함

shortLoan1 – 단기차입금, 2017(회계년도)년 데이터를 의미함

NCLiabilities1 – 비유동부채, 2017(회계년도)년 데이터를 의미함

longLoan1 – 장기차입금, 2017(회계년도)년 데이터를 의미함

netAsset1 – 순자산총계, 2017(회계년도)년 데이터를 의미함

surplus1 – 이익잉여금, 2017(회계년도)년 데이터를 의미함

revenue2 – 매출액, 2016(회계년도)년 데이터를 의미함

salescost2 – 매출원가, 2016(회계년도)년 데이터를 의미함

sga2 - 판매비와 관리비, 2016(회계년도)년 데이터를 의미함

salary2 – 급여, 2016(회계년도)년 데이터를 의미함

noi2 – 영업외수익, 2016(회계년도)년 데이터를 의미함

noe2 – 영업외비용, 2016(회계년도)년 데이터를 의미함

interest2 – 이자비용, 2016(회계년도)년 데이터를 의미함

ctax2 – 법인세비용, 2016(회계년도)년 데이터를 의미함

profit2 – 당기순이익, 2016(회계년도)년 데이터를 의미함

liquidAsset2 – 유동자산, 2016(회계년도)년 데이터를 의미함

quickAsset2 – 당좌자산, 2016(회계년도)년 데이터를 의미함

receivableS2 - 미수금(단기), 2016(회계년도)년 데이터를 의미함

inventoryAsset2 – 재고자산, 2016(회계년도)년 데이터를 의미함

nonCAsset2 – 비유동자산, 2016(회계년도)년 데이터를 의미함

tanAsset2 – 유형자산, 2016(회계년도)년 데이터를 의미함

OnonCAsset2 - 기타 비유동자산, 2016(회계년도)년 데이터를 의미함

receivableL2 – 장기미수금, 2016(회계년도)년 데이터를 의미함

Debt2 – 부채총계, 2016(회계년도)년 데이터를 의미함

liquidLiabilities2 – 유동부채, 2016(회계년도)년 데이터를 의미함

shortLoan2 – 단기차입금, 2016(회계년도)년 데이터를 의미함

NCLiabilities2 – 비유동부채, 2016(회계년도)년 데이터를 의미함

longLoan2 – 장기차입금, 2016(회계년도)년 데이터를 의미함

netAsset2 – 순자산총계, 2016(회계년도)년 데이터를 의미함

surplus2 – 이익잉여금, 2016(회계년도)년 데이터를 의미함

employee1 – 고용한 총 직원의 수, 2017(회계년도)년 데이터를 의미함

employee2 – 고용한 총 직원의 수, 2016(회계년도)년 데이터를 의미함

ownerChange – 대표자의 변동

# -\*- coding: utf-8 -\*-

"""

Created on Tue Sep 18 19:03:22 2018

@author: BTHANISH

"""

import os

import numpy as np

import pandas as pd

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import train\_test\_split, StratifiedKFold, GridSearchCV

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

from sklearn import svm

from sklearn.metrics import confusion\_matrix, accuracy\_score, roc\_auc\_score

import shap

import xgboost as xgb

import lightgbm as lgb

import seaborn as sns

import matplotlib.pyplot as plt

os.chdir('C:\\Users\\blued\\Desktop')

#Reading the train and test files

train\_prod\_df = pd.read\_csv('train.csv')

test\_prod\_df = pd.read\_csv('test.csv')

#Removing the comma in the employee1 and 2 columns in the test dataset and replace it with empty space and convert it to float format.

test\_prod\_df.employee1 = test\_prod\_df.employee1.astype('str').str.replace(",", "").astype('float')

test\_prod\_df.employee2 = test\_prod\_df.employee2.astype('str').str.replace(",", "").astype('float')

#Converting the employee1 and 2 column as float in the train set as done for the test dataset

train\_prod\_df.employee1 = train\_prod\_df.employee1.astype('float')

train\_prod\_df.employee2 = train\_prod\_df.employee2.astype('float')

train\_prod\_df.OC= train\_prod\_df.OC.astype('str').str.replace(" ","")

#Combining the train and test dataset

train\_test\_prod = train\_prod\_df.append(test\_prod\_df)

#Get the object and numeric columns seperately

factor\_columns = train\_test\_prod.select\_dtypes(include = ['object']).columns

numeric\_columns = train\_test\_prod.columns.difference(factor\_columns)

#After analysis realized that the bed counts of these two hospitals may have had wrong entries.

#Filling up the empty instkind and bedCount for hospital id 430 and 413

train\_test\_prod.loc[train\_test\_prod.inst\_id == 430, ['instkind']] = 'dental\_clinic'

train\_test\_prod.loc[train\_test\_prod.inst\_id == 430, ['bedCount']] = 0

train\_test\_prod.loc[train\_test\_prod.inst\_id == 413, ['bedCount']] = -999

#Fill the empty values in the object columns as "Not sure"

train\_test\_prod[factor\_columns] = train\_test\_prod[factor\_columns].fillna('Not\_sure')

#Fill all the empty values in the numeric columns as -999

train\_test\_prod[numeric\_columns] = train\_test\_prod[numeric\_columns].fillna(-999)

#Convert all the object columns to numeric since the ML algorithms don't accept object features directly

fac\_le = LabelEncoder()

train\_test\_prod[factor\_columns] = train\_test\_prod.loc[:,factor\_columns].apply(lambda x : fac\_le.fit\_transform(x))

#Splitting back data to train prod and test prod

train\_prod = train\_test\_prod.loc[train\_test\_prod.OC != 0,]

test\_prod = train\_test\_prod.loc[train\_test\_prod.OC == 0,]

train\_prod['OC'] = train\_prod['OC'] - 1

#Obtain the submission ID to create the submission file later

sub\_id = test\_prod.inst\_id

#Get the dependent and independent column

dep = 'OC'

indep = train\_prod.columns.difference([dep])

train\_prod\_X = train\_prod[indep]

train\_prod\_Y = train\_prod[dep]

test\_prod\_X = test\_prod[indep]

A value is trying to be set on a copy of a slice from a DataFrame.

Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy

############################################################################

############ Random Forest

############################################################################

estimators = 10

np.random.seed(100)

RF\_prod = RandomForestClassifier(n\_estimators = estimators)

RF\_prod\_model = RF\_prod.fit(train\_prod\_X, train\_prod\_Y)

RF\_prod\_prediction = RF\_prod.predict\_proba(test\_prod\_X)[:,1]

sub\_RF = pd.DataFrame({'inst\_id' : sub\_id , 'OC' : RF\_prod\_prediction })

sub\_RF = sub\_RF[['inst\_id', 'OC']]

############################################################################

############ GBM

############################################################################

estimators = 10

np.random.seed(100)

GBM\_prod = GradientBoostingClassifier(n\_estimators = estimators)

GBM\_prod\_model = GBM\_prod.fit(train\_prod\_X, train\_prod\_Y)

GBM\_prod\_prediction = GBM\_prod.predict\_proba(test\_prod\_X)[:,1]

sub\_GBM = pd.DataFrame({'inst\_id' : sub\_id , 'OC' : GBM\_prod\_prediction })

sub\_GBM = sub\_GBM[['inst\_id', 'OC']]

############################################################################

############ XGBOOST

############################################################################

dtrain\_prod = xgb.DMatrix(data = train\_prod\_X, label = train\_prod\_Y)

dtest\_prod = xgb.DMatrix(data = test\_prod\_X)

#Custom error function for the XGB model

threshold = 0.5

def eval\_error(preds, dtrain):

labels = dtrain.get\_label()

preds = (preds &gt; threshold ).astype('float')

return "accuracy", accuracy\_score(labels, preds)

param = {'objective' : 'binary:logistic',

'max\_depth' : 6,

'eta': 0.3,

'colsample\_bytree' : 1,

'subsample' : 1,

'silent' : 0

}

nrounds = 2

np.random.seed(100)

xgb\_model = xgb.train(param,

dtrain\_prod,

num\_boost\_round = nrounds ,

feval = eval\_error,

#maximize = True,

#early\_stopping\_rounds = 10,

)

XGB\_prediction = xgb\_model.predict(dtest\_prod)

sub\_XGB= pd.DataFrame({'inst\_id' : sub\_id , 'OC' : XGB\_prediction })

sub\_XGB= sub\_XGB[['inst\_id', 'OC']]

#PLotting the feature importance

xgb\_Imp = pd.DataFrame({'Features' : list(xgb\_model.get\_score().keys()),

'Importance' : list(xgb\_model.get\_score().values())}).sort\_values(['Importance'])

plt.figure()

sns.barplot(xgb\_Imp.Importance, xgb\_Imp.Features)

plt.show()

############################################################################

#Ensembling the three models

############################################################################

#Forming the ensemble dataset of the 3 models

ensemble = pd.DataFrame()

ensemble['inst\_id'] = sub\_XGB['inst\_id']

ensemble['XGB'] = sub\_XGB['OC']

ensemble['GBM'] = sub\_GBM['OC']

ensemble['RF'] = sub\_RF['OC']

# Taking the average of all 3 models

ensemble['ens'] = (ensemble['XGB'] + ensemble['GBM'] + ensemble['RF'])/3

ensemble['OC'] = (ensemble['ens'] &gt; 0.7).astype('int') #0.7 as the threshold and above that the hospital is closed.

#Printing to see all the hospitals that are classified as closed

print(ensemble.loc[ensemble['OC'] == 0, ])

ensemble = ensemble.loc[:, ['inst\_id', 'OC']]

#ensemble.to\_csv('ens\_XGB\_7\_RF\_4\_GBM\_2\_39.csv', index = False)

inst\_id XGB GBM RF ens OC

1 5 0.657157 0.443177 0.8 0.633445 0

2 6 0.409096 0.666447 0.6 0.558514 0

10 24 0.409096 0.925726 0.7 0.678274 0

13 30 0.409096 0.230173 0.8 0.479756 0

22 64 0.409096 0.936367 0.1 0.481821 0

40 123 0.409096 0.842282 0.7 0.650459 0

78 229 0.409096 0.539125 0.6 0.516074 0

84 258 0.409096 0.442562 0.6 0.483886 0

88 293 0.409096 0.724379 0.7 0.611158 0

99 341 0.409096 0.278068 0.4 0.362388 0

120 413 0.409096 0.278068 0.4 0.362388 0

122 424 0.409096 0.312302 0.3 0.340466 0

123 425 0.409096 0.580902 0.7 0.563333 0

124 429 0.409096 0.536108 0.7 0.548401 0

126 431 0.409096 0.536108 0.6 0.515068 0