"""

Created on Sun Jun 17 08:14:33 2018

@author: sumedh

"""

import os

import pandas as pd

import numpy as np

## Split data in Training (Development Sample) 70% and Hold-out Sample 30%

# Set working directory

os.chdir("D:/Great Lakes PGPDSE/Great Lakes/10 Supervised Learning - Classification/Supervised Learning Classification/Mini\_Project")

#Loading Dataset

pl = pd.read\_csv("PL\_XSELL.csv")

pl['split'] = np.random.randn(pl.shape[0], 1)

br = np.random.rand(len(pl)) <= 0.7

train = pl[br]

test = pl[~br]

# Writing Development\_Sample and Hold\_Out\_Sample into two different csv files

train.to\_csv("Development\_Sample.csv", encoding='utf-8', index=False)

test.to\_csv("Hold\_out\_Sample .csv", encoding='utf-8', index=False)

## Classification Tree on Unbalanced Dataset

import os

import pandas as pd

#Set the working directory

os.chdir("D:/Great Lakes PGPDSE/Great Lakes/10 Supervised Learning - Classification/Supervised Learning Classification/Mini\_Project")

#Load the Dataset

CTDF\_dev = pd.read\_csv("Development\_Sample.csv")

CTDF\_holdout = pd.read\_csv("Hold\_out\_Sample .csv")

del CTDF\_dev['split']

del CTDF\_holdout['split']

print( len(CTDF\_dev), len(CTDF\_holdout)) ###13993 6007

CTDF\_dev.head()

import numpy as np

import matplotlib.pyplot as plt

#Data Preprocessing

#Splitting into features and response variables

CTDF\_dev.columns

X = CTDF\_dev[['GENDER', 'BALANCE', 'OCCUPATION',

'AGE\_BKT', 'SCR', 'HOLDING\_PERIOD', 'ACC\_TYPE',

'LEN\_OF\_RLTN\_IN\_MNTH', 'NO\_OF\_L\_CR\_TXNS', 'NO\_OF\_L\_DR\_TXNS',

'TOT\_NO\_OF\_L\_TXNS','FLG\_HAS\_CC','AMT\_L\_DR', 'FLG\_HAS\_ANY\_CHGS',

'AMT\_MIN\_BAL\_NMC\_CHGS','FLG\_HAS\_OLD\_LOAN']]

#Categorical Variable to Numerical Variables

X\_train = pd.get\_dummies(X)

X\_train.columns

y\_train = CTDF\_dev["TARGET"]

print (type(X\_train) , type(y\_train))

#Decision Tree

#Loading the library

from sklearn.tree import DecisionTreeClassifier

#Setting the parameter

clf = DecisionTreeClassifier(criterion = "gini" ,

min\_samples\_split = 100,

min\_samples\_leaf = 10,

max\_depth = 100)

#Calling the fit function to built the tree

clf.fit(X\_train,y\_train)

import pydot

from sklearn.tree import tree

from sklearn.tree import export\_graphviz

from sklearn.externals.six import StringIO

dot\_data = StringIO()

feature\_list = list(X\_train.columns.values)

export\_graphviz(clf,

out\_file = dot\_data,

feature\_names = feature\_list)

graph=pydot.graph\_from\_dot\_data(dot\_data.getvalue())

graph[0].write\_pdf("D:/Great Lakes PGPDSE/Great Lakes/10 Supervised Learning - Classification/Supervised Learning Classification/Mini\_Project.classification\_tree\_output.pdf")

Nodes = pd.DataFrame(clf.tree\_.\_\_getstate\_\_()["nodes"])

Nodes

feature\_importance = pd.DataFrame([X\_train.columns,

clf.tree\_.compute\_feature\_importances()])

feature\_importance.T

## Let us see how good is the model

pred\_y\_train = clf.predict(X\_train )

pred\_y\_train

## Let us see the classification accuracy of our model

from sklearn.metrics import accuracy\_score

from sklearn.metrics import roc\_curve

from sklearn.metrics import auc

score = accuracy\_score(y\_train, pred\_y\_train)

score ##0.8891588651468592

y\_train\_prob = clf.predict\_proba(X\_train)

## AUC for training

fpr, tpr, thresholds = roc\_curve(y\_train, y\_train\_prob[:,1])

auc(fpr, tpr) ## 0.8909147836850357

## Let us see how good is the model

X\_holdout = CTDF\_holdout[['GENDER', 'BALANCE', 'OCCUPATION',

'AGE\_BKT', 'SCR', 'HOLDING\_PERIOD', 'ACC\_TYPE',

'LEN\_OF\_RLTN\_IN\_MNTH', 'NO\_OF\_L\_CR\_TXNS', 'NO\_OF\_L\_DR\_TXNS',

'TOT\_NO\_OF\_L\_TXNS','FLG\_HAS\_CC','AMT\_L\_DR', 'FLG\_HAS\_ANY\_CHGS',

'AMT\_MIN\_BAL\_NMC\_CHGS','FLG\_HAS\_OLD\_LOAN']]

X\_test = pd.get\_dummies(X\_holdout)

y\_test = CTDF\_holdout["TARGET"]

pred\_y\_test = clf.predict(X\_test)

score\_h = accuracy\_score(y\_test, pred\_y\_test)

score\_h ## 0.874812718495089

y\_test\_prob = clf.predict\_proba(X\_test)

fpr, tpr, thresholds = roc\_curve(y\_test, y\_test\_prob[:,1])

auc(fpr, tpr) ## 0.7990326076444083

# AUC\_diff = (0.8909147836850357-0.7990326076444083)/0.8909147836850357

# AUC\_diff = 0.1031323957388841

y\_freq = np.bincount(y\_train)

y\_val = np.nonzero(y\_freq)[0]

np.vstack((y\_val,y\_freq[y\_val])).T

#array([[ 0, 12251],

# [ 1, 1742]], dtype=int64) 1742/(12251+1742)=12.44

#Cross validation function

from sklearn.cross\_validation import cross\_val\_score

scores = cross\_val\_score(clf, X\_train , y\_train, cv = 10, scoring='roc\_auc')

scores.mean() # 0.7879884317540934

scores.std() # 0.0147261362940826

y\_train\_prob = clf.predict\_proba(X\_train)

fpr, tpr, thresholds = roc\_curve(y\_train, y\_train\_prob[:,1])

auc(fpr, tpr) # 0.8909147836850357

y\_test\_prob = clf.predict\_proba(X\_test)

fpr, tpr, thresholds = roc\_curve(y\_test, y\_test\_prob[:,1])

auc(fpr, tpr) # 0.7990326076444083

## Tuning the Classifier using GridSearchCV

from sklearn.grid\_search import GridSearchCV

#help(GridSearchCV)

param\_dist = {"criterion": ["gini","entropy"],

"max\_depth": np.arange(3,10),

}

tree = DecisionTreeClassifier(min\_samples\_split = 100,

min\_samples\_leaf = 10)

tree\_cv = GridSearchCV(tree, param\_dist, cv = 10,

scoring = 'roc\_auc', verbose = 100)

tree\_cv.fit(X\_train,y\_train)

## Building the model using best combination of parameters

print("Tuned Decision Tree parameter : {}".format(tree\_cv.best\_params\_))

# Tuned Decision Tree parameter : {'criterion': 'entropy', 'max\_depth': 9}

classifier = tree\_cv.best\_estimator\_

classifier.fit(X\_train,y\_train)

#predicting probabilities

y\_train\_prob = classifier.predict\_proba(X\_train)

fpr, tpr, thresholds = roc\_curve(y\_train, y\_train\_prob[:,1])

auc\_prob\_d = auc(fpr, tpr)

auc\_prob\_d # 0.8208470013132318

y\_test\_prob = classifier.predict\_proba(X\_test)

fpr, tpr, thresholds = roc\_curve(y\_test, y\_test\_prob[:,1])

auc\_prob\_h = auc(fpr, tpr)

auc\_prob\_h # 0.7508230646573208

#(auc\_prob\_d-auc\_prob\_h)/auc\_prob\_d # 0.0853069287502826

## Rank Ordering

Prediction = classifier.predict\_proba(X\_train)

CTDF\_dev["prob\_score"] = Prediction[:,1]

#scoring step

#decile code

def deciles(x):

decile = pd.Series(index=[0,1,2,3,4,5,6,7,8,9])

for i in np.arange(0.1,1.1,0.1):

decile[int(i\*10)]=x.quantile(i)

def z(x):

if x<decile[1]: return(1)

elif x<decile[2]: return(2)

elif x<decile[3]: return(3)

elif x<decile[4]: return(4)

elif x<decile[5]: return(5)

elif x<decile[6]: return(6)

elif x<decile[7]: return(7)

elif x<decile[8]: return(8)

elif x<decile[9]: return(9)

elif x<=decile[10]: return(10)

else:return(np.NaN)

s=x.map(z)

return(s)

def Rank\_Ordering(X,y,Target):

X['decile']=deciles(X[y])

Rank=X.groupby('decile').apply(lambda x: pd.Series([

np.min(x[y]),

np.max(x[y]),

np.mean(x[y]),

np.size(x[y]),

np.sum(x[Target]),

np.size(x[Target][x[Target]==0]),

],

index=(["min\_resp","max\_resp","avg\_resp",

"cnt","cnt\_resp","cnt\_non\_resp"])

)).reset\_index()

Rank = Rank.sort\_values(by='decile',ascending=False)

Rank["rrate"] = Rank["cnt\_resp"]\*100/Rank["cnt"]

Rank["cum\_resp"] = np.cumsum(Rank["cnt\_resp"])

Rank["cum\_non\_resp"] = np.cumsum(Rank["cnt\_non\_resp"])

Rank["cum\_resp\_pct"] = Rank["cum\_resp"]/np.sum(Rank["cnt\_resp"])

Rank["cum\_non\_resp\_pct"]=Rank["cum\_non\_resp"]/np.sum(Rank["cnt\_non\_resp"])

Rank["KS"] = Rank["cum\_resp\_pct"] - Rank["cum\_non\_resp\_pct"]

Rank

return(Rank)

Rank = Rank\_Ordering(CTDF\_dev,"prob\_score","TARGET")

Rank # Considere the highest KS value among all available values 0.461771

## Let us see the Rank Ordering on Hold-Out Dataset

Prediction\_h = classifier.predict\_proba(X\_test)

CTDF\_holdout["prob\_score"] = Prediction\_h[:,1]

Rank\_h = Rank\_Ordering(CTDF\_holdout,"prob\_score","TARGET")

Rank\_h # On Holding dataset highest KS value to be considered i.e.0.362128

#Rank = (Rank - Rank\_h) / Rank

# (0.461771-0.362128)/0.461771 = 0.21578444726931745

y\_freq = np.bincount(y\_test)

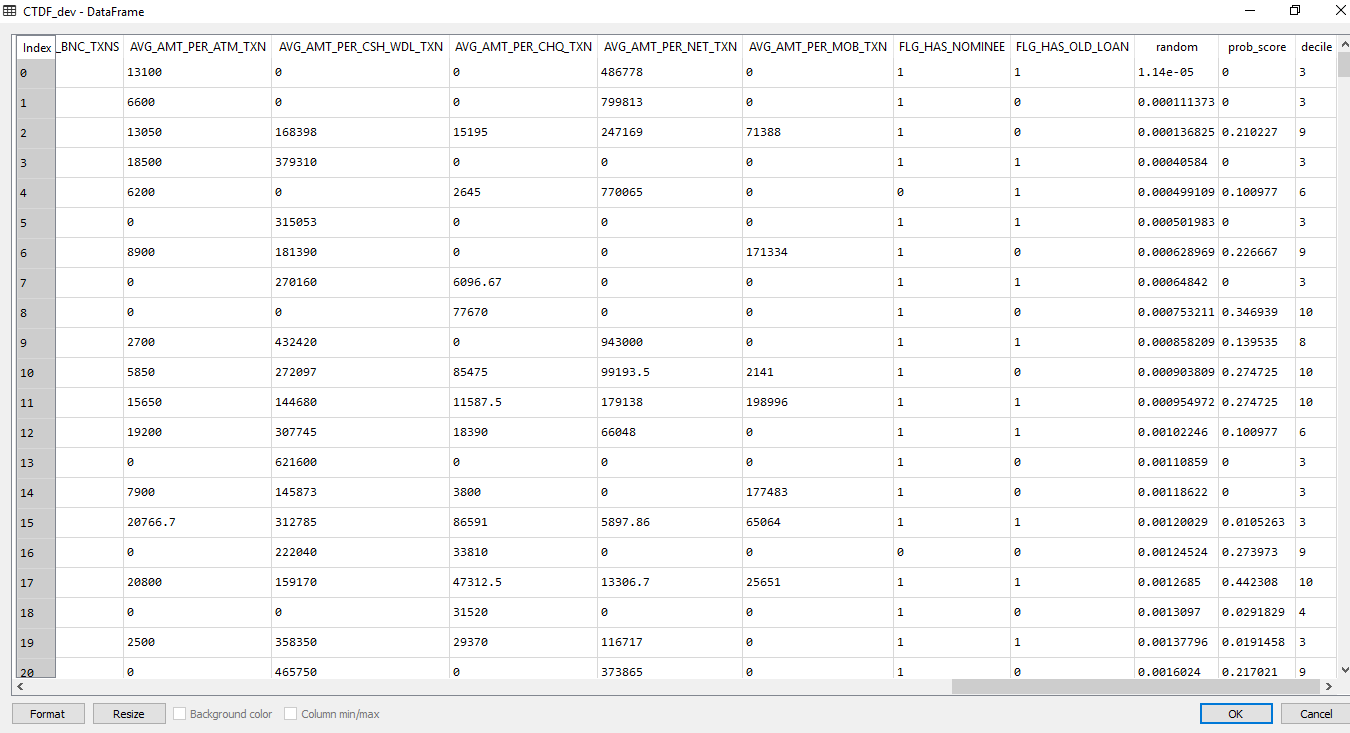
y\_val = np.nonzero(y\_freq)[0]

np.vstack((y\_val,y\_freq[y\_val])).T

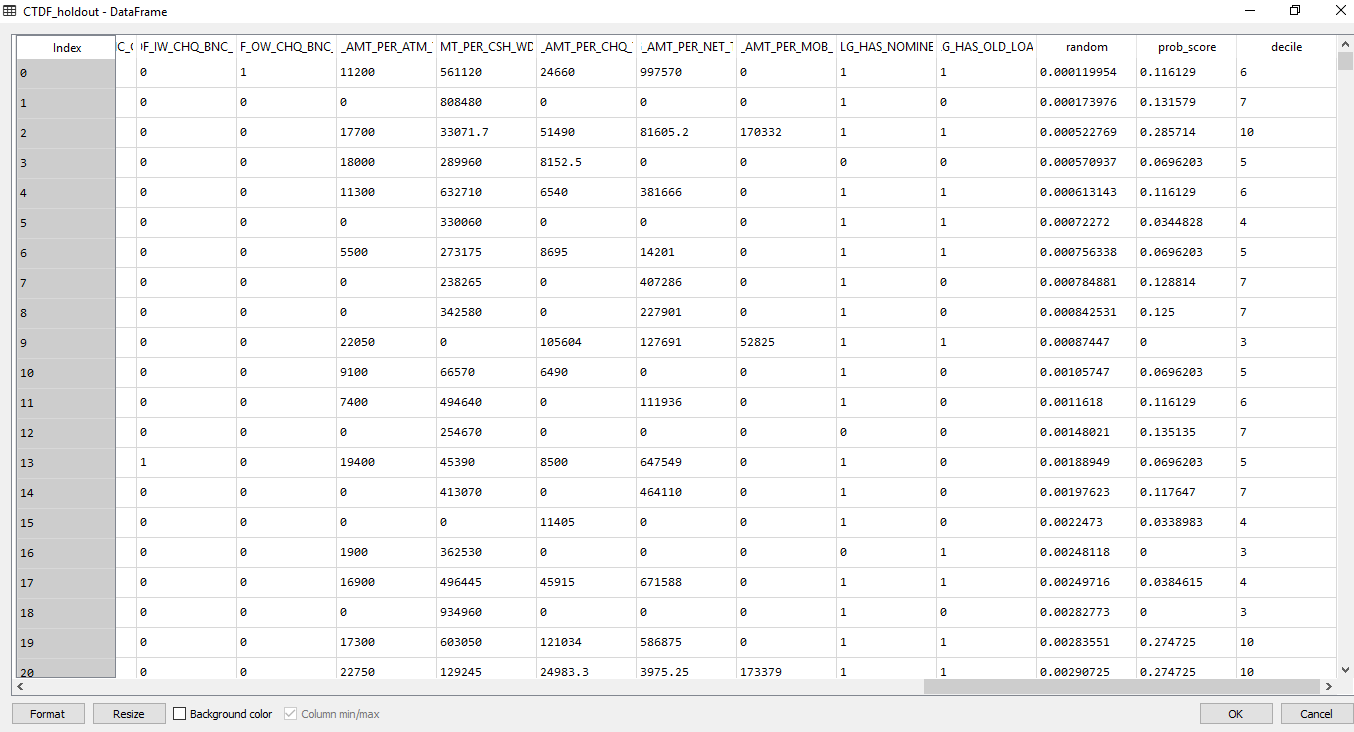
# array([[ 0, 5237],

# [ 1, 770]], dtype=int64) 770/(770+5237) = 12.8183

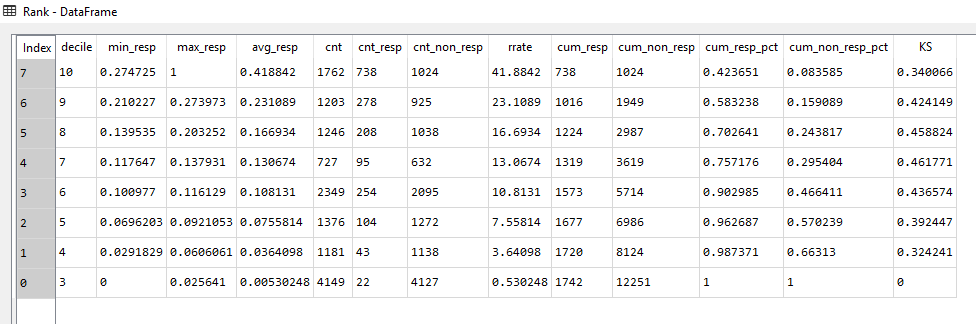
CTDF\_dev ( training dataset ) with probability score and decile



CTDF\_holdout ( testing dataset ) with probability score and deciles in it.



Rank ordering of the dataset ( train dataset ) – Development

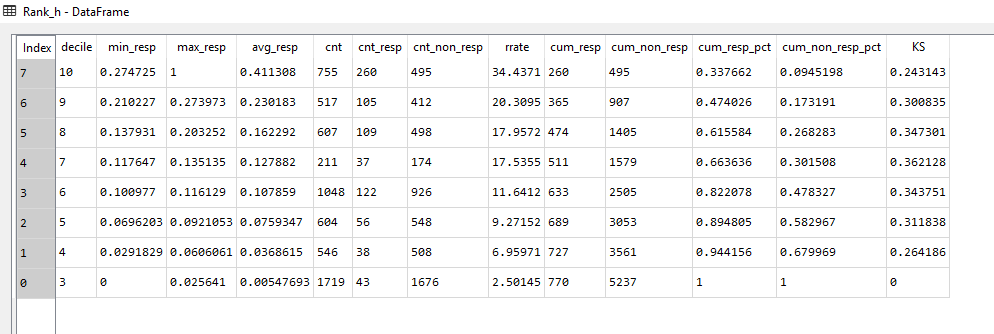


For Training dataset maximum ks statistics value is coming at 7th decile and its value is 0.461771.

=((0.34\*738)+(0.42\*278)+(0.4588\*208)+(0.4617\*95))/(738+278+208+95) =0.3828.

If we target top four decile i.e. 10,9,8 and 7 having the maximum KS Statistics is 0.461771. Then there is a chance of 0.3828 times We can can taget the customer for Personal Loan having the higher probability of taking the Personal loan by them.

Rank ordering of the dataset ( test dataset ) – Hold out



For Testing dataset maximum ks statistics value is coming at 7th decile and its value is 0.362128.

=((0.2431\*260)+(0.30\*105)+(0.34\*109)+(0.36\*37)/(260+105+109+37)=0.2839

If we target top four decile i.e. 10,9,8 and 7 having the maximum KS Statistics is 0.62128. Then there is a chance of 0.2839 times We can can taget the customer for Personal Loan having the higher probability of taking the Personal loan by them.

Accuracy for the X\_train model is 88.91%

AUC curve for the X\_train model is 89.09%.

Accuracy for the y\_test data is 87.48%

AUC curve for the y\_test data is 79.09%.

This Model is showing better results in Python in compare to R.

AUC and Accuracy of model is coming less for same dataset in R but for same dataset in Python showing high results for same parameters.