**Experiential Learning Approach for Deloitte**

# **Case Study 1 Report**

**Big Data Analytics and Visualization for the client ABC Bank**

# **Credit Fraud Analysis: Functional Design**

**By Raktim Ray**

## **Introduction**

ABC bank is leading Banking firm based in the UK. Since past few years, they have seen a huge revenue growth year over the year. In last couple of years, they have expanded to 12 countries in Europe, 7 in Asia, 1 in Latin America and 1 in Africa.

## **Problem statement**

Along with geographical expansion, ABC Bank have observed an increase in the number of cases related to loan fraud.

Therefore, they want to employ modern business analytics-based techniques to accurately predict the probability of fraud when we receive a loan application. They want to detect a minimum of 90% of the fraud cases by way of this analysis and reject such applications. In this process, a few genuine customers may be denied loan. However, they are more concerned about avoiding maximum number of potential fraud cases.

## **Description of data provided**

ABC Bank provided 3 data sets for analysis. The 3 sets were csv files containing:

1. Previous application data
2. Current application data
3. Description of the columns of the datasets

## **Proposed algorithm and Solution Steps**

The proposed code is given below, divided into several parts, each having their own functionalities

**imports**

import seaborn as sns

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn import preprocessing

import warnings

%matplotlib inline

from sklearn import tree

from sklearn.model\_selection import train\_test\_split # Import train\_test\_split function

from sklearn import metrics

from sklearn.tree import DecisionTreeClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn import svm

from sklearn.ensemble import RandomForestClassifier

from sklearn.neural\_network import MLPClassifier

from sklearn.metrics import classification\_report, plot\_confusion\_matrix#for visualizing tree

from sklearn import svm

import datetime

warnings.filterwarnings("ignore")

**Loading the data**

data = pd.read\_csv("C:/Users/rakray/Documents/Deloitte\_Training/Code/deloitte-training-project-1/datasets/application\_data.csv")

df\_application\_data = pd.DataFrame(data)

data = pd.read\_csv("C:/Users/rakray/Documents/Deloitte\_Training/Code/deloitte-training-project-1/datasets/previous\_application.csv")

df\_previous\_application = pd.DataFrame(data)

**Missing value removal**

acceptable\_non\_NAN\_values\_fraction = 0.5

df\_application\_data\_2 = df\_application\_data.dropna(axis='columns', how="any", thresh=(1-acceptable\_non\_NAN\_values\_fraction)\*len(df\_application\_data.index))

df\_previous\_application\_2 = df\_previous\_application.dropna(axis='columns', how="any", thresh=(1-acceptable\_non\_NAN\_values\_fraction)\*len(df\_previous\_application.index))

**NAN value replacement**

def NAN\_value\_replacement(dataframe):

for col in dataframe:

if (dataframe[col].dtype == "int64" or dataframe[col].dtype == "float64"):

dataframe[col] = dataframe[col].fillna(dataframe[col].median())

elif (dataframe[col].dtype == "object"):

dataframe[col] = dataframe[col].fillna(dataframe[col].mode().iloc[0])

return dataframe

df\_application\_data\_2 = NAN\_value\_replacement(df\_application\_data\_2)

df\_previous\_application\_2 = NAN\_value\_replacement(df\_previous\_application\_2)

**Boxplot**

def Boxplot(dataframe, column):

plt.figure(figsize=(10,7))

plt.title(column)

plt.boxplot(dataframe[column])

plt.show()

**Numerical Dataframe**

def numerical\_df(df):

numerical = df.select\_dtypes(exclude='object')

return numerical

***# gettting numerical dfs***

numerical\_df\_previous\_application = numerical\_df(df\_previous\_application\_2)

numerical\_df\_application\_data = numerical\_df(df\_application\_data\_2)

**Removing outliers**

***# Outlier removing function***

def new\_IQR(res\_df, numerical\_df):

for i in list(numerical\_df):

sorted(i)

iqr1 = res\_df[i].quantile(0.25)

iqr3 = res\_df[i].quantile(0.75)

iqr = iqr3-iqr1

lower\_limit = iqr1 - 1.5\*iqr

upper\_limit = iqr3 + 1.5\*iqr

temp\_data = res\_df[(res\_df[i] > lower\_limit) & (res\_df[i] <upper\_limit)]

if temp\_data[i].value\_counts().shape[0] > 10:

res\_df = res\_df[(res\_df[i] > lower\_limit) & (res\_df[i] <upper\_limit)] return res\_df

***# Removing outliers***

df\_previous\_application\_2 = new\_IQR(df\_previous\_application\_2, numerical\_df\_previous\_application)

df\_application\_data\_2 = new\_IQR(df\_application\_data\_2, numerical\_df\_application\_data)

**Categorical data**

df\_previous\_application\_2["ad\_MONTHS\_decision\_ct"] = abs(df\_previous\_application\_2["ad\_days\_decision\_ct"])/30

bins = [0,1,2,3,4,5,6,7,8,9,np.inf]

slots = ["0-1","1-2","2-3","3-4","4-5","5-6","6-7","7-8","8-9","Above 9"]

df\_previous\_application\_2["ad\_MONTHS\_decision\_ct"] = pd.cut(df\_previous\_application\_2["ad\_MONTHS\_decision\_ct"], bins=bins, labels=slots)

df\_application\_data\_2["ad\_YEARS\_birth\_ct"] = abs(df\_application\_data\_2["ad\_days\_birth\_ct"])/365

bins = [0,10,20,30,40,50,60,70,80,90,np.inf]

slots = ["0-10","11-20","21-30","31-40","41-50","51-60","61-70","71-80","81-90","Above 90"]

df\_application\_data\_2["ad\_YEARS\_birth\_ct"] = pd.cut(df\_application\_data\_2["ad\_YEARS\_birth\_ct"], bins=bins, labels=slots)

**Encoding**

def encoder(dataframe):

label\_encoder = preprocessing.LabelEncoder()

for (columnName, columnData) in dataframe.iteritems():

if columnData.dtypes == "object" or columnData.dtype.name == "category":

dataframe[columnName] = label\_encoder.fit\_transform(dataframe[columnName])

return dataframe

**Feature Selection**

***# Feature Selection - Correlation Coefficient***

def corr\_co(name, dataframe, thresh, plot\_visibility):

corr\_matrix = dataframe.corr()

if (plot\_visibility):

sns.heatmap(corr\_matrix,annot=True,cmap=plt.cm.CMRmap\_r)

plt.show()

coll\_corr = []

threshold = thresh

flag = 0

for i in range(len(corr\_matrix.columns)):

for j in range(i):

if abs(corr\_matrix.iloc[i,j]) > threshold:

colname = corr\_matrix.columns[i]

if colname not in coll\_corr and flag > 0:

coll\_corr.append(colname)

# print(colname)

flag+=1

return coll\_corr

***# calling correlation coefficient function***

corr\_matrix\_previous\_application = corr\_co("previous\_application.csv", df\_previous\_application\_2, 0.85, False)

corr\_matrix\_application\_data = corr\_co("current\_application", df\_application\_data\_2, 0.85, False)

***# drop the columns obtained from coefficient function***

df\_previous\_application\_2\_dropped = df\_previous\_application\_2.drop(labels=corr\_matrix\_previous\_application, inplace=False, axis=1)

df\_application\_data\_2\_dropped = df\_application\_data\_2.drop(labels=corr\_matrix\_application\_data, inplace=False, axis=1)

**Merge**

merged\_dataframe = pd.merge(df\_application\_data\_2, df\_previous\_application\_2, how='inner', on='ad\_sk\_id\_curr\_ct')

today = datetime.datetime.today().strftime('%d-%m-%Y')

merged\_dataframe.to\_csv(f'raktim-cleaned-data-{today}.csv', index=False)

**Creating table schema for redshift from dataframe**

print("create table sample (")

output = ""

for i in merged\_dataframe.columns:

datatype = merged\_dataframe[i].dtypes

if datatype == 'int64':

output = 'int'

elif datatype == 'float64':

output = 'float'

else:

output = 'varchar(100)'

print(i+" "+output+",")

print(");")

**Univariate**

***# sample***

def data\_type(dataset,col):

if dataset[col].dtype == np.int64 or dataset[col].dtype == np.float64 or dataset[col].dtype == np.int32:

return "numerical"

if dataset[col].dtype == "object":

return "categorical"

def univariate(dataset,col,target\_col,ylog=False,x\_label\_angle=False,h\_layout=True):

if data\_type(dataset,col) == "numerical":

sns.distplot(dataset[col],hist=False)

elif data\_type(dataset,col) == "categorical":

val\_count = dataset[col].value\_counts()

df1 = pd.DataFrame({col: val\_count.index,'count': val\_count.values})

target\_1\_percentage = dataset[[col, target\_col]].groupby([col],as\_index=False).mean()

target\_1\_percentage[target\_col] = target\_1\_percentage[target\_col]\*100

target\_1\_percentage.sort\_values(by=target\_col,inplace = True)

***# If the plot is not readable, use the log scale***

if(h\_layout):

fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(15,7))

else:

fig, (ax1, ax2) = plt.subplots(nrows=2, figsize=(25,35))

***# 1. Subplot 1: Count plot of the column***

s = sns.countplot(ax=ax1, x=col, data=dataset, hue=target\_col)

ax1.set\_title(col, fontsize = 20)

ax1.legend(['Repayer','Defaulter'])

ax1.set\_xlabel(col,fontdict={'fontsize' : 15, 'fontweight' : 3})

if(x\_label\_angle):

s.set\_xticklabels(s.get\_xticklabels(),rotation=75)

***# 2. Subplot 2: Percentage of defaulters within the column***

s = sns.barplot(ax=ax2, x = col, y=target\_col, data=target\_1\_percentage)

ax2.set\_title("Defaulters % in "+col, fontsize = 20)

ax2.set\_xlabel(col,fontdict={'fontsize' : 15, 'fontweight' : 3})

ax2.set\_ylabel(target\_col,fontdict={'fontsize' : 15, 'fontweight' : 3})

if(x\_label\_angle):

s.set\_xticklabels(s.get\_xticklabels(),rotation=75)

***# If the plot is not readable, use the log scale***

if ylog:

ax1.set\_yscale('log')

ax1.set\_ylabel("Count (log)",fontdict={'fontsize' : 15, 'fontweight' : 3})

else:

ax1.set\_ylabel("Count",fontdict={'fontsize' : 15, 'fontweight' : 3})

plt.show()

***# Univariate call***

univariate(df\_application\_data\_2\_dropped, "ad\_code\_gender\_ct", "ad\_target\_ct")

Chart, bar chart

Description automatically generated

univariate(df\_application\_data\_2\_dropped, "ad\_name\_education\_type\_ct", "ad\_target\_ct")

Chart, waterfall chart

Description automatically generated

**Insights**

***# Insight dataframe 1***

insight\_df\_1 = df\_application\_data\_2[["ad\_sk\_id\_curr\_ct", "ad\_target\_ct", "ad\_name\_education\_type\_ct"]]

print("\nInsight dataframe - 1: Education Type\n")

print(insight\_df\_1.head())

col = 'ad\_name\_education\_type\_ct'

target\_col = 'ad\_target\_ct'

sns.set()

s = sns.countplot(insight\_df\_1[col],hue=insight\_df\_1[target\_col])

s.set\_xticklabels(s.get\_xticklabels(), rotation=45)

Chart, waterfall chart

Description automatically generated

***# Insight dataframe 2***

insight\_df\_2 = df\_application\_data\_2[["ad\_sk\_id\_curr\_ct", "ad\_target\_ct", "ad\_name\_housing\_type\_ct"]]

print("\nInsight dataframe - 2: Housing Type\n")

print(insight\_df\_2.head())

col = 'ad\_name\_housing\_type\_ct'

target\_col = 'ad\_target\_ct'

sns.set()

s = sns.countplot(insight\_df\_2[col],hue=insight\_df\_2[target\_col])

s.set\_xticklabels(s.get\_xticklabels(), rotation=45)

Chart

Description automatically generated

***# Insight dataframe 3***

insight\_df\_3 = df\_application\_data\_2[["ad\_sk\_id\_curr\_ct", "ad\_target\_ct", "ad\_occupation\_type\_ct"]]

print("\nInsight dataframe - 3: Occupation Type\n")

print(insight\_df\_3.head())

col = 'ad\_occupation\_type\_ct'

target\_col = 'ad\_target\_ct'

sns.set()

s = sns.countplot(insight\_df\_3[col],hue=insight\_df\_3[target\_col])

s.set\_xticklabels(s.get\_xticklabels(), rotation=45)

Graphical user interface

Description automatically generated

**Encoding before machine learning**

merged\_dataframe = encoder(merged\_dataframe)

target\_column = "ad\_target\_ct"

df1 = merged\_dataframe.drop(target\_column, 1)

df1 = df1.loc[:, ~df1.columns.str.contains('^Unnamed')]

feature\_columns = list(df1)

X = merged\_dataframe[feature\_columns]

Y = merged\_dataframe[target\_column]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=1) # 80% training and 20% test

**Binary classification**

***# Decision Tree classifer***

clf = DecisionTreeClassifier()

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

y\_pred = clf.predict(X\_test)

print("Classification report - \n", classification\_report(y\_test,y\_pred))

Report -

precision recall f1-score support

0 0.96 0.95 0.95 20302

1 0.62 0.66 0.64 2561

accuracy 0.92 22863

macro avg 0.79 0.80 0.80 22863

weighted avg 0.92 0.92 0.92 22863

***# Logisitic Regression***

clf = LogisticRegression()

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

y\_pred = clf.predict(X\_test)

print("Classification report - \n", classification\_report(y\_test,y\_pred))

Report -

precision recall f1-score support

0 0.89 1.00 0.94 20302

1 0.00 0.00 0.00 2561

accuracy 0.89 22863

macro avg 0.44 0.50 0.47 22863

weighted avg 0.79 0.89 0.84 22863

***# Support Vector Machines***

clf = svm.LinearSVC()

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

y\_pred = clf.predict(X\_test)

print("Classification report - \n", classification\_report(y\_test,y\_pred))

Report -

precision recall f1-score support

0 0.96 0.00 0.00 20302

1 0.11 1.00 0.20 2561

accuracy 0.11 22863

macro avg 0.54 0.50 0.10 22863

weighted avg 0.87 0.11 0.03 22863

***# Random Forest Classifier***

clf = RandomForestClassifier()

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

y\_pred = clf.predict(X\_test)

print("Classification report - \n", classification\_report(y\_test,y\_pred))

Report -

precision recall f1-score support

0 0.93 1.00 0.96 20302

1 1.00 0.41 0.58 2561

accuracy 0.93 22863

macro avg 0.97 0.71 0.77 22863

weighted avg 0.94 0.93 0.92 22863

***# Neural Networks***

clf = MLPClassifier()

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

y\_pred = clf.predict(X\_test)

print("Classification report - \n", classification\_report(y\_test,y\_pred))

Report -

precision recall f1-score support

0 0.89 1.00 0.94 20302

1 0.24 0.01 0.02 2561

accuracy 0.89 22863

macro avg 0.56 0.50 0.48 22863

weighted avg 0.82 0.89 0.84 22863

**Quicksight**

**Visualization 1**

Chart

Description automatically generated

*More below…*

**Visualization 2**

Chart, bar chart

Description automatically generated

**Visualization 3**

Chart, pie chart

Description automatically generated

## **Closure**

Data was cleaned with Jupyter notebook and pandas, then uploaded to Redshift, and finally visualized in Quicksight.