
CASE STUDY - 1 :: Healthcare Provider Fraudulent Detection

This notebook contains the feature engineering :-

- **Final Model Pipelining**

Above steps are performed on the publicly available dataset at Kaggle.

Kindly checkout below link for gaining BUSINESS related insights related to this problem ::

- Deck : Detailed Explanation

Kindly checkout below link for TECHNICAL design document ::

- Technical Document

Kindly checkout below link for In-depth Description and Reasoning of all the Features ::

- Features Description

Notebook Contents

Importing_Libraries

In [ ]:
```
import os
import sys
import numpy as np
import pandas as pd
import joblib

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
from sklearn.calibration import CalibratedClassifierCV

%matplotlib inline
```

In [ ]:
```
pd.set_option('display.max_columns',80)
label_font_dict = {'family':'sans-serif','size':13.5,'color':'brown','style':'italic
title_font_dict = {'family':'sans-serif','size':16.5,'color':'Blue','style':'italic'
```

Unseen_Data_Files

In [ ]:
```
# Unseen dataset files
bene_file = "Test_Beneficiarydata-1542969243754.csv"
ip_file = "Test_Inpatientdata-1542969243754.csv"
op_file = "Test_Outpatientdata-1542969243754.csv"
tgt_lbl_file = "Test-1542969243754.csv"

# Folder hierarchies
folder_1 = "Dataset"
folder_2 = "TEST"
data_files_loc = os.path.join(os.path.join(os.getcwd(), folder_1), folder_2)
print("### Location where we have all the unseen data files --> {}".format(data_file
```

```
### Location where we have all the unseen data files --> E:\STUDY\PROJECTS\AAIC_Case
_Study_1\Dataset\TEST
```

Final_Pipeline

In [ ]:
```
def unseen_data_files_names(files_loc, bene_data, ip_data, op_data, tgt_lbl_data):
    """
    Description : This function is created for reading the files names of BENEFICIAR
    """
    # Generating the unseen data files names
    bene_file_name = os.path.join(files_loc, bene_data)
    ip_file_name = os.path.join(files_loc, ip_data)
    op_file_name = os.path.join(files_loc, op_data)
```

```
    tgt_label_name = os.path.join(files_loc, tgt_lbl_data)
    return bene_file_name, ip_file_name, op_file_name, tgt_label_name
```

In [ ]:
```
def read_unseen_data(unseen_data_loc, b_file, i_file, o_file, tgt_file):
    """
    Description : This function is created for reading the data based on the files n
    and TGT_LABELS.
    """
    # Reading the data files names
    bene_unseen_data, ip_unseen_data, op_unseen_data, tgt_lbl_unseen_data = unseen_d




    # Reading the data files
    unseen_bene_df = pd.read_csv(bene_unseen_data)
    unseen_ip_df = pd.read_csv(ip_unseen_data)
    unseen_op_df = pd.read_csv(op_unseen_data)
    unseen_tgt_lbls_df = pd.read_csv(tgt_lbl_unseen_data)

    # Creating a dictionary which will contain all the unseen data files
    entire_unseen_data = {"Unseen_Provider":unseen_tgt_lbls_df,
                          "Unseen_Beneficiary":unseen_bene_df,
                          "Unseen_Inpatient":unseen_ip_df,
                          "Unseen_Outpatient":unseen_op_df}
    return entire_unseen_data
```

In [ ]:
```
# Generating the unseen data dictionary
unseen_data_dict = read_unseen_data(unseen_data_loc=data_files_loc,
                                    b_file=bene_file,
                                    i_file=ip_file,
                                    o_file=op_file,
                                    tgt_file=tgt_lbl_file)
```

In [ ]:
```
def pre_process(X_input):
    """
    Description : This function is created for pre-processing the data files based o
    the best model.
    """
    # Load the unseen data files
    test_tgt_lbls_df = X_input['Unseen_Provider']
    test_bene_df = X_input['Unseen_Beneficiary']
    test_ip_df = X_input['Unseen_Inpatient']
    test_op_df = X_input['Unseen_Outpatient']

    # Adding Feature - 1
    test_ip_df["Admitted?"] = 1
    test_op_df["Admitted?"] = 0

    # Commom columns must be 28
    common_cols = [col for col in test_ip_df.columns if col in test_op_df.columns]

    # Merging the IP and OP dataset on the basis of common columns
    test_ip_op_df = pd.merge(left=test_ip_df, right=test_op_df, left_on=common_cols,

    # Joining the IP_OP dataset with the BENE data
    test_ip_op_bene_df = pd.merge(left=test_ip_op_df, right=test_bene_df, left_on='B

    # Joining the IP_OP_BENE dataset with the Tgt Label Provider Data
    test_iobp_df = pd.merge(left=test_ip_op_bene_df, right=test_tgt_lbls_df, left_on

    # Joining with the PRV Tgt Labels
    prvs_claims_df = pd.DataFrame(test_iobp_df.groupby(['Provider'])['ClaimID'].coun
```

```
prvs_claims_tgt_lbls_df = pd.merge(left=prvs_claims_df, right=test_tgt_lbls_df,

# Adding Feature - 2
test_iobp_df['DOB'] = pd.to_datetime(test_iobp_df['DOB'], format="%Y-%m-%d")
test_iobp_df['DOD'] = pd.to_datetime(test_iobp_df['DOD'], format="%Y-%m-%d")
test_iobp_df['Is_Alive?'] = test_iobp_df['DOD'].apply(lambda val: 'No' if val !=

# Adding Feature - 3
test_iobp_df['ClaimStartDt'] = pd.to_datetime(test_iobp_df['ClaimStartDt'], form
test_iobp_df['ClaimEndDt'] = pd.to_datetime(test_iobp_df['ClaimEndDt'], format="
test_iobp_df['Claim_Duration'] = (test_iobp_df['ClaimEndDt'] - test_iobp_df['Cla

# Adding Feature - 4
test_iobp_df['AdmissionDt'] = pd.to_datetime(test_iobp_df['AdmissionDt'], format
test_iobp_df['DischargeDt'] = pd.to_datetime(test_iobp_df['DischargeDt'], format
test_iobp_df['Admitted_Duration'] = (test_iobp_df['DischargeDt'] - test_iobp_df[

# Adding Feature - 5
# Filling the Null values as MAX Date of Death in the Dataset
test_iobp_df['DOD'].fillna(value=pd.to_datetime('2009-12-01',format ='%Y-%m-%d')
test_iobp_df['Bene_Age'] = round(((test_iobp_df['DOD'] - test_iobp_df['DOB']).dt

# Adding Feature - 6
test_iobp_df['Att_Phy_tot_claims'] = test_iobp_df.groupby(['AttendingPhysician']
test_iobp_df['Opr_Phy_tot_claims'] = test_iobp_df.groupby(['OperatingPhysician']
test_iobp_df['Oth_Phy_tot_claims'] = test_iobp_df.groupby(['OtherPhysician'])['C
test_iobp_df['Att_Phy_tot_claims'].fillna(value=0, inplace=True)
test_iobp_df['Opr_Phy_tot_claims'].fillna(value=0, inplace=True)
test_iobp_df['Oth_Phy_tot_claims'].fillna(value=0, inplace=True)
test_iobp_df['Att_Opr_Oth_Phy_Tot_Claims'] = test_iobp_df['Att_Phy_tot_claims']
test_iobp_df.drop(['Att_Phy_tot_claims', 'Opr_Phy_tot_claims', 'Oth_Phy_tot_clai

# Adding Feature - 7
test_iobp_df["Prv_Tot_Att_Phy"] = test_iobp_df.groupby(['Provider'])['AttendingP
test_iobp_df["Prv_Tot_Opr_Phy"] = test_iobp_df.groupby(['Provider'])['OperatingP
test_iobp_df["Prv_Tot_Oth_Phy"] = test_iobp_df.groupby(['Provider'])['OtherPhysi
test_iobp_df['Prv_Tot_Att_Phy'].fillna(value=0, inplace=True)
test_iobp_df['Prv_Tot_Opr_Phy'].fillna(value=0, inplace=True)
test_iobp_df['Prv_Tot_Oth_Phy'].fillna(value=0, inplace=True)
test_iobp_df['Prv_Tot_Att_Opr_Oth_Phys'] = test_iobp_df['Prv_Tot_Att_Phy'] + tes
test_iobp_df.drop(['Prv_Tot_Att_Phy', 'Prv_Tot_Opr_Phy', 'Prv_Tot_Oth_Phy'], axi

# Adding Feature - 8
test_iobp_df['PRV_Tot_Admit_DCodes'] = test_iobp_df.groupby(['Provider'])['ClmAd

# Adding Feature - 9
test_iobp_df['PRV_Tot_DGrpCodes'] = test_iobp_df.groupby(['Provider'])['Diagnosi

# Adding Feature - 10
test_iobp_df['DOB_Year'] = test_iobp_df['DOB'].dt.year
test_iobp_df['PRV_Tot_Unq_DOB_Years'] = test_iobp_df.groupby(['Provider'])['DOB_
test_iobp_df.drop(['DOB_Year'], axis=1, inplace=True)

# Adding Feature - 11
test_iobp_df['PRV_Bene_Age_Sum'] = test_iobp_df.groupby(['Provider'])['Bene_Age'

# Adding Feature - 12
test_iobp_df['PRV_Insc_Clm_ReImb_Amt'] = test_iobp_df.groupby(['Provider'])['Ins

# Adding Feature - 13
test_iobp_df['RenalDiseaseIndicator'] = test_iobp_df['RenalDiseaseIndicator'].ap
test_iobp_df['PRV_Tot_RKD_Patients'] = test_iobp_df.groupby(['Provider'])['Renal
test_iobp_df.drop(['NoOfMonths_PartACov', 'NoOfMonths_PartBCov'], axis=1, inplac
test_iobp_df['Admitted_Duration'].fillna(value=0,inplace=True)
```

```
# PRV Aggregate features
test_iobp_df["PRV_CoPayment"] = test_iobp_df.groupby('Provider')['DeductibleAmtP
test_iobp_df["PRV_IP_Annual_ReImb_Amt"] = test_iobp_df.groupby('Provider')['IPAn
test_iobp_df["PRV_IP_Annual_Ded_Amt"] = test_iobp_df.groupby('Provider')['IPAnnu
test_iobp_df["PRV_OP_Annual_ReImb_Amt"] = test_iobp_df.groupby('Provider')['OPAn
test_iobp_df["PRV_OP_Annual_Ded_Amt"] = test_iobp_df.groupby('Provider')['OPAnnu
test_iobp_df["PRV_Admit_Duration"] = test_iobp_df.groupby('Provider')['Admitted_
test_iobp_df["PRV_Claim_Duration"] = test_iobp_df.groupby('Provider')['Claim_Dur

def create_agg_feats(grp_col, feat_name, operation='sum'):
    """
    Description :: This function is created for adding the aggregated features i
        - Beneficiary
        - Attending Physician
        - Operating Physician
        - Other Physician and etc..

    Input Parameters :: It accepts below inputs:
        - grp_col : `str`
            - It represents the feature or level at which you want to perform th

        - feat_name : `str`
            - It represents the feature whose aggregated aspect you want to capt

        - operation : `str`
            - It represents the aggregation operation you want to perform.(By de
    """
    feat_1 = feat_name + "_Insc_ReImb_Amt"
    test_iobp_df[feat_1] = test_iobp_df.groupby(grp_col)['InscClaimAmtReimbursed

    feat_2 = feat_name + "_CoPayment"
    test_iobp_df[feat_2] = test_iobp_df.groupby(grp_col)['DeductibleAmtPaid'].tr

    feat_3 = feat_name + "_IP_Annual_ReImb_Amt"
    test_iobp_df[feat_3] = test_iobp_df.groupby(grp_col)['IPAnnualReimbursementA

    feat_4 = feat_name + "_IP_Annual_Ded_Amt"
    test_iobp_df[feat_4] = test_iobp_df.groupby(grp_col)['IPAnnualDeductibleAmt'

    feat_5 = feat_name + "_OP_Annual_ReImb_Amt"
    test_iobp_df[feat_5] = test_iobp_df.groupby(grp_col)['OPAnnualReimbursementA

    feat_6 = feat_name + "_OP_Annual_Ded_Amt"
    test_iobp_df[feat_6] = test_iobp_df.groupby(grp_col)['OPAnnualDeductibleAmt'

    feat_7 = feat_name + "_Admit_Duration"
    test_iobp_df[feat_7] = test_iobp_df.groupby(grp_col)['Admitted_Duration'].tr

    feat_8 = feat_name + "_Claim_Duration"
    test_iobp_df[feat_8] = test_iobp_df.groupby(grp_col)['Claim_Duration'].trans

# BENE, PHYs, Diagnosis Admit and Group Codes columns
create_agg_feats(grp_col='BeneID', feat_name="BENE")
create_agg_feats(grp_col='AttendingPhysician', feat_name="ATT_PHY")
create_agg_feats(grp_col='OperatingPhysician', feat_name="OPT_PHY")
create_agg_feats(grp_col='OtherPhysician', feat_name="OTH_PHY")
create_agg_feats(grp_col='ClmAdmitDiagnosisCode', feat_name="Claim_Admit_Diag_Co
create_agg_feats(grp_col='DiagnosisGroupCode', feat_name="Diag_GCode")

# Dropping these 3 columns as there 99% of values are same
test_iobp_df.drop(['ClmProcedureCode_4', 'ClmProcedureCode_5', 'ClmProcedureCode

# Diagnosis Codes columns
```

```
create_agg_feats(grp_col='ClmDiagnosisCode_1', feat_name="Claim_DiagCode1")
create_agg_feats(grp_col='ClmDiagnosisCode_2', feat_name="Claim_DiagCode2")
create_agg_feats(grp_col='ClmDiagnosisCode_3', feat_name="Claim_DiagCode3")
create_agg_feats(grp_col='ClmDiagnosisCode_4', feat_name="Claim_DiagCode4")
create_agg_feats(grp_col='ClmDiagnosisCode_5', feat_name="Claim_DiagCode5")
create_agg_feats(grp_col='ClmDiagnosisCode_6', feat_name="Claim_DiagCode6")
create_agg_feats(grp_col='ClmDiagnosisCode_7', feat_name="Claim_DiagCode7")
create_agg_feats(grp_col='ClmDiagnosisCode_8', feat_name="Claim_DiagCode8")
create_agg_feats(grp_col='ClmDiagnosisCode_9', feat_name="Claim_DiagCode9")
create_agg_feats(grp_col='ClmDiagnosisCode_10', feat_name="Claim_DiagCode10")

# Medical Procedure Codes columns
create_agg_feats(grp_col='ClmProcedureCode_1', feat_name="Claim_ProcCode1")
create_agg_feats(grp_col='ClmProcedureCode_2', feat_name="Claim_ProcCode2")
create_agg_feats(grp_col='ClmProcedureCode_3', feat_name="Claim_ProcCode3")

# PROVIDER <--> other features :: To get claim counts
test_iobp_df["ClmCount_Provider"]=test_iobp_df.groupby(['Provider'])['ClaimID'].
test_iobp_df["ClmCount_Provider_BeneID"]=test_iobp_df.groupby(['Provider','BeneI
test_iobp_df["ClmCount_Provider_AttendingPhysician"]=test_iobp_df.groupby(['Prov
test_iobp_df["ClmCount_Provider_OtherPhysician"]=test_iobp_df.groupby(['Provider
test_iobp_df["ClmCount_Provider_OperatingPhysician"]=test_iobp_df.groupby(['Prov
test_iobp_df["ClmCount_Provider_ClmAdmitDiagnosisCode"]=test_iobp_df.groupby(['P
test_iobp_df["ClmCount_Provider_ClmProcedureCode_1"]=test_iobp_df.groupby(['Prov
test_iobp_df["ClmCount_Provider_ClmProcedureCode_2"]=test_iobp_df.groupby(['Prov
test_iobp_df["ClmCount_Provider_ClmProcedureCode_3"]=test_iobp_df.groupby(['Prov
test_iobp_df["ClmCount_Provider_ClmDiagnosisCode_1"]=test_iobp_df.groupby(['Prov
test_iobp_df["ClmCount_Provider_ClmDiagnosisCode_2"]=test_iobp_df.groupby(['Prov
test_iobp_df["ClmCount_Provider_ClmDiagnosisCode_3"]=test_iobp_df.groupby(['Prov
test_iobp_df["ClmCount_Provider_ClmDiagnosisCode_4"]=test_iobp_df.groupby(['Prov
test_iobp_df["ClmCount_Provider_ClmDiagnosisCode_5"]=test_iobp_df.groupby(['Prov
test_iobp_df["ClmCount_Provider_ClmDiagnosisCode_6"]=test_iobp_df.groupby(['Prov
test_iobp_df["ClmCount_Provider_ClmDiagnosisCode_7"]=test_iobp_df.groupby(['Prov
test_iobp_df["ClmCount_Provider_ClmDiagnosisCode_8"]=test_iobp_df.groupby(['Prov
test_iobp_df["ClmCount_Provider_ClmDiagnosisCode_9"]=test_iobp_df.groupby(['Prov
test_iobp_df["ClmCount_Provider_ClmDiagnosisCode_10"]=test_iobp_df.groupby(['Pro
test_iobp_df["ClmCount_Provider_DiagnosisGroupCode"]=test_iobp_df.groupby(['Prov

# PROVIDER <--> BENE <--> PHYSICIANS :: To get claim counts
test_iobp_df["ClmCount_Provider_BeneID_AttendingPhysician"]=test_iobp_df.groupby
test_iobp_df["ClmCount_Provider_BeneID_OtherPhysician"]=test_iobp_df.groupby(['P
test_iobp_df["ClmCount_Provider_BeneID_OperatingPhysician"]=test_iobp_df.groupby

# PROVIDER <--> BENE <--> ATTENDING PHYSICIAN <--> PROCEDURE CODES :: To get cla
test_iobp_df["ClmCount_Provider_BeneID_AttendingPhysician_ClmProcedureCode_1"]=t
test_iobp_df["ClmCount_Provider_BeneID_AttendingPhysician_ClmProcedureCode_2"]=t
test_iobp_df["ClmCount_Provider_BeneID_AttendingPhysician_ClmProcedureCode_3"]=t

# PROVIDER <--> BENE <--> OPERATING PHYSICIAN <--> PROCEDURE CODES :: To get cla
test_iobp_df["ClmCount_Provider_BeneID_OperatingPhysician_ClmProcedureCode_1"]=t
test_iobp_df["ClmCount_Provider_BeneID_OperatingPhysician_ClmProcedureCode_2"]=t
test_iobp_df["ClmCount_Provider_BeneID_OperatingPhysician_ClmProcedureCode_3"]=t

# PROVIDER <--> BENE <--> OTHER PHYSICIAN <--> PROCEDURE CODES :: To get claim c
test_iobp_df["ClmCount_Provider_BeneID_OtherPhysician_ClmProcedureCode_1"]=test_
test_iobp_df["ClmCount_Provider_BeneID_OtherPhysician_ClmProcedureCode_2"]=test_
test_iobp_df["ClmCount_Provider_BeneID_OtherPhysician_ClmProcedureCode_3"]=test_

# PROVIDER <--> BENE <--> ATTENDING PHYSICIAN <--> DIAGNOSIS CODES :: To get cla
test_iobp_df["ClmCount_Provider_BeneID_AttendingPhysician_ClmDiagnosisCode_1"]=t
test_iobp_df["ClmCount_Provider_BeneID_AttendingPhysician_ClmDiagnosisCode_2"]=t
test_iobp_df["ClmCount_Provider_BeneID_AttendingPhysician_ClmDiagnosisCode_3"]=t
test_iobp_df["ClmCount_Provider_BeneID_AttendingPhysician_ClmDiagnosisCode_4"]=t
test_iobp_df["ClmCount_Provider_BeneID_AttendingPhysician_ClmDiagnosisCode_5"]=t
```

```
test_iobp_df["ClmCount_Provider_BeneID_AttendingPhysician_ClmDiagnosisCode_6"]=t
test_iobp_df["ClmCount_Provider_BeneID_AttendingPhysician_ClmDiagnosisCode_7"]=t
test_iobp_df["ClmCount_Provider_BeneID_AttendingPhysician_ClmDiagnosisCode_8"]=t
test_iobp_df["ClmCount_Provider_BeneID_AttendingPhysician_ClmDiagnosisCode_9"]=t
test_iobp_df["ClmCount_Provider_BeneID_AttendingPhysician_ClmDiagnosisCode_10"]=

# PROVIDER <--> BENE <--> OPERATING PHYSICIAN <--> DIAGNOSIS CODES :: To get cla
test_iobp_df["ClmCount_Provider_BeneID_OperatingPhysician_ClmDiagnosisCode_1"]=t
test_iobp_df["ClmCount_Provider_BeneID_OperatingPhysician_ClmDiagnosisCode_2"]=t
test_iobp_df["ClmCount_Provider_BeneID_OperatingPhysician_ClmDiagnosisCode_3"]=t
test_iobp_df["ClmCount_Provider_BeneID_OperatingPhysician_ClmDiagnosisCode_4"]=t
test_iobp_df["ClmCount_Provider_BeneID_OperatingPhysician_ClmDiagnosisCode_5"]=t
test_iobp_df["ClmCount_Provider_BeneID_OperatingPhysician_ClmDiagnosisCode_6"]=t
test_iobp_df["ClmCount_Provider_BeneID_OperatingPhysician_ClmDiagnosisCode_7"]=t
test_iobp_df["ClmCount_Provider_BeneID_OperatingPhysician_ClmDiagnosisCode_8"]=t
test_iobp_df["ClmCount_Provider_BeneID_OperatingPhysician_ClmDiagnosisCode_9"]=t
test_iobp_df["ClmCount_Provider_BeneID_OperatingPhysician_ClmDiagnosisCode_10"]=

# PROVIDER <--> BENE <--> OTHER PHYSICIAN <--> DIAGNOSIS CODES :: To get claim c
test_iobp_df["ClmCount_Provider_BeneID_OtherPhysician_ClmDiagnosisCode_1"]=test_
test_iobp_df["ClmCount_Provider_BeneID_OtherPhysician_ClmDiagnosisCode_2"]=test_
test_iobp_df["ClmCount_Provider_BeneID_OtherPhysician_ClmDiagnosisCode_3"]=test_
test_iobp_df["ClmCount_Provider_BeneID_OtherPhysician_ClmDiagnosisCode_4"]=test_
test_iobp_df["ClmCount_Provider_BeneID_OtherPhysician_ClmDiagnosisCode_5"]=test_
test_iobp_df["ClmCount_Provider_BeneID_OtherPhysician_ClmDiagnosisCode_6"]=test_
test_iobp_df["ClmCount_Provider_BeneID_OtherPhysician_ClmDiagnosisCode_7"]=test_
test_iobp_df["ClmCount_Provider_BeneID_OtherPhysician_ClmDiagnosisCode_8"]=test_
test_iobp_df["ClmCount_Provider_BeneID_OtherPhysician_ClmDiagnosisCode_9"]=test_
test_iobp_df["ClmCount_Provider_BeneID_OtherPhysician_ClmDiagnosisCode_10"]=test

# PROVIDER <--> BENE <--> PROCEDURE CODES :: To get claim counts
test_iobp_df["ClmCount_Provider_BeneID_ClmProcedureCode_1"]=test_iobp_df.groupby
test_iobp_df["ClmCount_Provider_BeneID_ClmProcedureCode_2"]=test_iobp_df.groupby
test_iobp_df["ClmCount_Provider_BeneID_ClmProcedureCode_3"]=test_iobp_df.groupby

# PROVIDER <--> BENE <--> DIAGNOSIS CODES :: To get claim counts
test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_1"]=test_iobp_df.groupby
test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_2"]=test_iobp_df.groupby
test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_3"]=test_iobp_df.groupby
test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_4"]=test_iobp_df.groupby
test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_5"]=test_iobp_df.groupby
test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_6"]=test_iobp_df.groupby
test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_7"]=test_iobp_df.groupby
test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_8"]=test_iobp_df.groupby
test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_9"]=test_iobp_df.groupby
test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_10"]=test_iobp_df.groupb

# PROVIDER <--> BENE <--> DIAGNOSIS CODES <--> PROCEDURE CODES :: To get claim c
test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_1_ClmProcedureCode_1"]=t
test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_1_ClmProcedureCode_2"]=t
test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_1_ClmProcedureCode_3"]=t
test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_2_ClmProcedureCode_1"]=t
test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_2_ClmProcedureCode_2"]=t
test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_2_ClmProcedureCode_3"]=t
test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_3_ClmProcedureCode_1"]=t
test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_3_ClmProcedureCode_2"]=t
test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_3_ClmProcedureCode_3"]=t
test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_4_ClmProcedureCode_1"]=t
test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_4_ClmProcedureCode_2"]=t
test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_4_ClmProcedureCode_3"]=t
test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_5_ClmProcedureCode_1"]=t
test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_5_ClmProcedureCode_2"]=t
test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_5_ClmProcedureCode_3"]=t
test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_6_ClmProcedureCode_1"]=t
```

```
    test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_6_ClmProcedureCode_2"]=t
    test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_6_ClmProcedureCode_3"]=t
    test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_7_ClmProcedureCode_1"]=t
    test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_7_ClmProcedureCode_2"]=t
    test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_7_ClmProcedureCode_3"]=t
    test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_8_ClmProcedureCode_1"]=t
    test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_8_ClmProcedureCode_2"]=t
    test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_8_ClmProcedureCode_3"]=t
    test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_9_ClmProcedureCode_1"]=t
    test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_9_ClmProcedureCode_2"]=t
    test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_9_ClmProcedureCode_3"]=t
    test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_10_ClmProcedureCode_1"]=
    test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_10_ClmProcedureCode_2"]=
    test_iobp_df["ClmCount_Provider_BeneID_ClmDiagnosisCode_10_ClmProcedureCode_3"]=

    # Removing unwanted columns
    remove_unwanted_columns=['BeneID', 'ClaimID', 'ClaimStartDt','ClaimEndDt','Atten
                             'AdmissionDt', 'ClmAdmitDiagnosisCode', 'DischargeDt', 'Di
                             'ClmDiagnosisCode_1', 'ClmDiagnosisCode_2', 'ClmDiagnosisC
                             'ClmDiagnosisCode_6', 'ClmDiagnosisCode_7', 'ClmDiagnosisC
                             'ClmProcedureCode_1', 'ClmProcedureCode_2', 'ClmProcedureC

    test_iobp_df.drop(columns=remove_unwanted_columns, axis=1, inplace=True)

    # Filling Nulls in Deductible Amt Paid by Patient
    test_iobp_df['DeductibleAmtPaid'].fillna(value=0, inplace=True)

    # Binary encoding the categorical features --> 0 means No and 1 means Yes
    test_iobp_df['Gender'] = test_iobp_df['Gender'].apply(lambda val: 0 if val == 2
    test_iobp_df['Is_Alive?'] = test_iobp_df['Is_Alive?'].apply(lambda val: 0 if val

    test_iobp_df['ChronicCond_Alzheimer'] = test_iobp_df['ChronicCond_Alzheimer'].ap
    test_iobp_df['ChronicCond_Heartfailure'] = test_iobp_df['ChronicCond_Heartfailur
    test_iobp_df['ChronicCond_KidneyDisease'] = test_iobp_df['ChronicCond_KidneyDise
    test_iobp_df['ChronicCond_Cancer'] = test_iobp_df['ChronicCond_Cancer'].apply(la
    test_iobp_df['ChronicCond_ObstrPulmonary'] = test_iobp_df['ChronicCond_ObstrPulm
    test_iobp_df['ChronicCond_Depression'] = test_iobp_df['ChronicCond_Depression'].
    test_iobp_df['ChronicCond_Diabetes'] = test_iobp_df['ChronicCond_Diabetes'].appl
    test_iobp_df['ChronicCond_IschemicHeart'] = test_iobp_df['ChronicCond_IschemicHe
    test_iobp_df['ChronicCond_Osteoporasis'] = test_iobp_df['ChronicCond_Osteoporasi
    test_iobp_df['ChronicCond_rheumatoidarthritis'] = test_iobp_df['ChronicCond_rheu
    test_iobp_df['ChronicCond_stroke'] = test_iobp_df['ChronicCond_stroke'].apply(la

    # Encoding the Categorical features
    test_iobp_df = pd.get_dummies(test_iobp_df,columns=['Gender', 'Race', 'Admitted?

    # Filling Nulls in the aggregated features
    test_iobp_df.fillna(value=0, inplace=True)

    # Grouping the records
    test_iobp_df = test_iobp_df.groupby(['Provider'],as_index=False).agg('sum')

    # Segregating the sets
    X_unseen = test_iobp_df.drop(axis=1, columns=['Provider'])
    y_unseen_prvs = test_iobp_df['Provider']

    return X_unseen, y_unseen_prvs
```

In [ ]:
```
X_unseen_pp, y_unseen_pp = pre_process(unseen_data_dict)
```

In [ ]:
```
# Checking shape of unseen data after pre-processing it should have 299 features
X_unseen_pp.shape, y_unseen_pp.shape
```

Out[ ]: ((1353, 299), (1353,))

In [ ]:
```
# Loading the best model
rfc_3 = joblib.load('best_model.pkl')
```

In [ ]:
```
# Making the predictions on the unseen data
unseen_preds = rfc_3.predict(X_unseen_pp)
```

In [ ]:
```
unseen_results = pd.concat([pd.DataFrame(y_unseen_pp), pd.DataFrame(unseen_preds)],a
unseen_results.reset_index(drop=True, inplace=True)
unseen_results.columns = ['Providers','Potential_Fraud?']
unseen_results
```

Out[ ]:

	Providers	Potential_Fraud?
0	PRV51002	1
1	PRV51006	1
2	PRV51009	1
3	PRV51010	1
4	PRV51018	1
...	...	...
1348	PRV57713	0
1349	PRV57726	0
1350	PRV57745	0
1351	PRV57749	1
1352	PRV57750	1

1353 rows × 2 columns

SUMMARY

- Here, we have generated the predictions from the best-trained model on the unseen dataset.

Performance_on_TRAIN_and_VALIDATION_Sets

In [ ]:
```
def pred_prob(clf, data):
    """
    Description :: This function is created for storing the predicted probabability

    Input :: It accepts below input parameters :
      - clf : Trained model classifier
      - data : Dataset for which we want to generate the predictions
    """
    y_pred = clf.predict_proba(data)[:,1]
    return y_pred

def draw_roc(train_fpr, train_tpr, test_fpr, test_tpr):
    """
    Description :: This function is created for calculating the AUC score on train a

    Input :: It accepts below input parameters :
```

```
      - train_fpr : Train False +ve rate
      - train_tpr : Train True +ve rate
      - test_fpr : Test False +ve rate
      - test_tpr : Test True +ve rate
    """
    # calculate auc for train and test
    train_auc = auc(train_fpr, train_tpr)
    test_auc = auc(test_fpr, test_tpr)
    with plt.style.context('seaborn-poster'):
      plt.plot(train_fpr, train_tpr, label="Train AUC ="+"{:.4f}".format(train_auc),
      plt.plot(test_fpr, test_tpr, label="Test AUC ="+"{:.4f}".format(test_auc), col
      plt.legend()
      plt.xlabel("False Positive Rate(FPR)", fontdict=label_font_dict)
      plt.ylabel("True Positive Rate(TPR)", fontdict=label_font_dict)
      plt.title("Area Under Curve", fontdict=title_font_dict)
      plt.grid(b=True, which='major', color='lightgrey', linestyle='--')
      plt.minorticks_on()
      plt.show()

def find_best_threshold(threshold, fpr, tpr):
    """
    Description :: This function is created for finding the best threshold value.
    """
    t = threshold[np.argmax(tpr * (1-fpr))]
    return t

def predict_with_best_t(proba, threshold):
    """
    Description :: This function is created for generating the predictions based on
    """
    predictions = []
    for i in proba:
        if i>=threshold:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions

def draw_confusion_matrix(best_t, x_train, x_test, y_train, y_test, y_train_pred, y_
    """
    Description :: This function is created for plotting the confusion matrix of TRA
    """
    fig, ax = plt.subplots(1,2, figsize=(20,6))

    train_prediction = predict_with_best_t(y_train_pred, best_t)
    cm = confusion_matrix(y_train, train_prediction)
    with plt.style.context('seaborn'):
        sns.heatmap(cm, annot=True, fmt='d', ax=ax[0], cmap='viridis')
        ax[0].set_title('Train Dataset Confusion Matrix', fontdict=title_font_dict)
        ax[0].set_xlabel("Predicted Label", fontdict=label_font_dict)
        ax[0].set_ylabel("Actual Label", fontdict=label_font_dict)

    test_prediction = predict_with_best_t(y_test_pred, best_t)
    cm = confusion_matrix(y_test, test_prediction)
    with plt.style.context('seaborn'):
        sns.heatmap(cm, annot=True, fmt='d', ax=ax[1], cmap='summer')
        ax[1].set_title('Test Dataset Confusion Matrix', fontdict=title_font_dict)
        ax[1].set_xlabel("Predicted Label", fontdict=label_font_dict)
        ax[1].set_ylabel("Actual Label", fontdict=label_font_dict)

    plt.show()

    return train_prediction, test_prediction
```

In [ ]:
```
def validate_model(clf, x_train, x_test, y_train, y_test):
    """
    Description :: This function is created for performing the evaluation of the tra
    """
    # predict the probability of train data
    y_train_pred = pred_prob(clf, x_train)

    # predict the probability of test data
    y_test_pred = pred_prob(clf, x_test)

    # calculate tpr, fpr using roc_curve
    train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
    test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

    # calculate auc for train and test
    train_auc = auc(train_fpr, train_tpr)
    print("### Train AUC = {}".format(train_auc))
    test_auc = auc(test_fpr, test_tpr)
    print("### Test AUC = {}".format(test_auc))

    # plotting the ROC curve
    draw_roc(train_fpr, train_tpr, test_fpr, test_tpr)

    # Best threshold value
    best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)

    # Plotting the confusion matrices
    train_prediction, test_prediction = draw_confusion_matrix(best_t, x_train, x_tes

    # Generating the F1-scores
    train_f1_score = f1_score(y_train, train_prediction)
    test_f1_score = f1_score(y_test, test_prediction)

    return test_auc, train_f1_score, test_f1_score, best_t
```

In [ ]:
```
# Reading the TRAIN and VALIDATION sets for checking the model performance
X_train_std = pd.read_csv("X_train_std.csv", index_col=0)
X_test_std = pd.read_csv("X_test_std.csv", index_col=0)

y_train = pd.read_csv("y_train.csv", index_col=0)
y_test = pd.read_csv("y_test.csv", index_col=0)
```

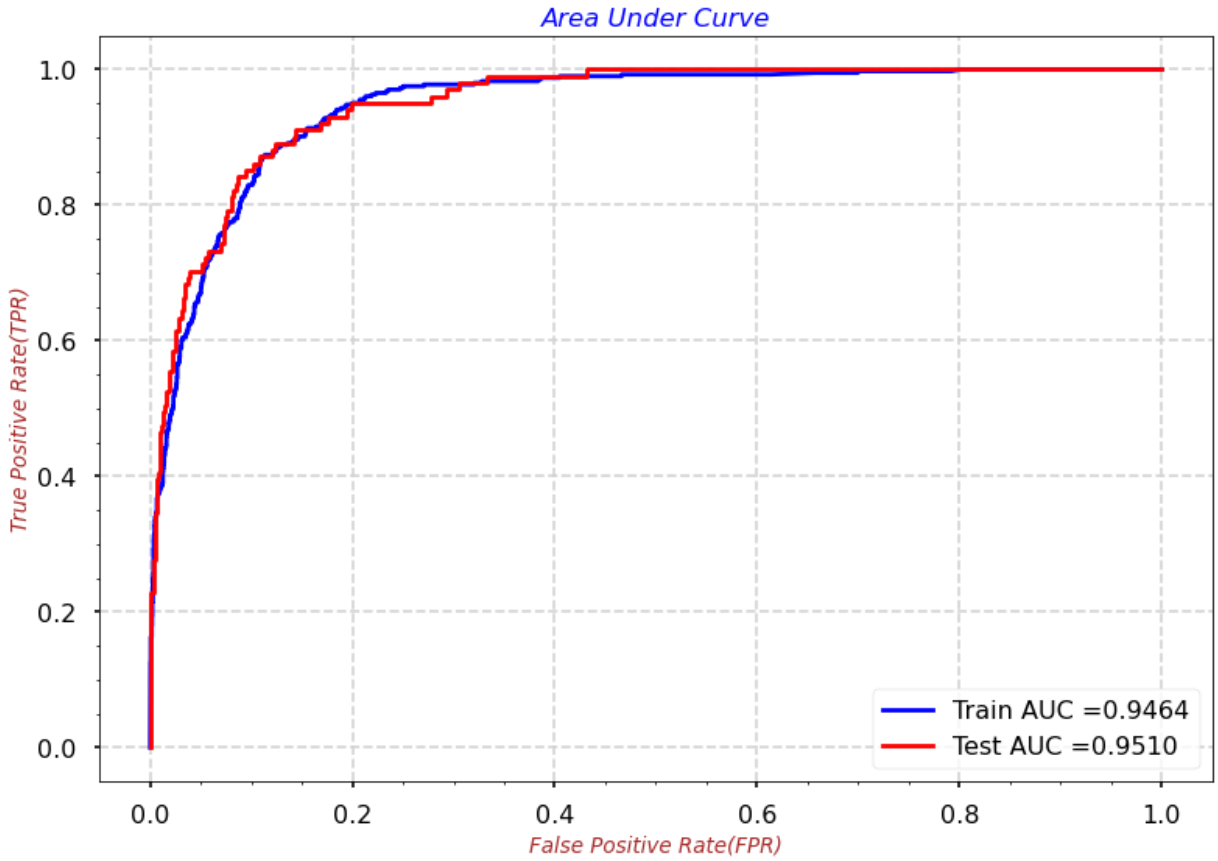
In [ ]:
```
np.ravel(y_test)
```

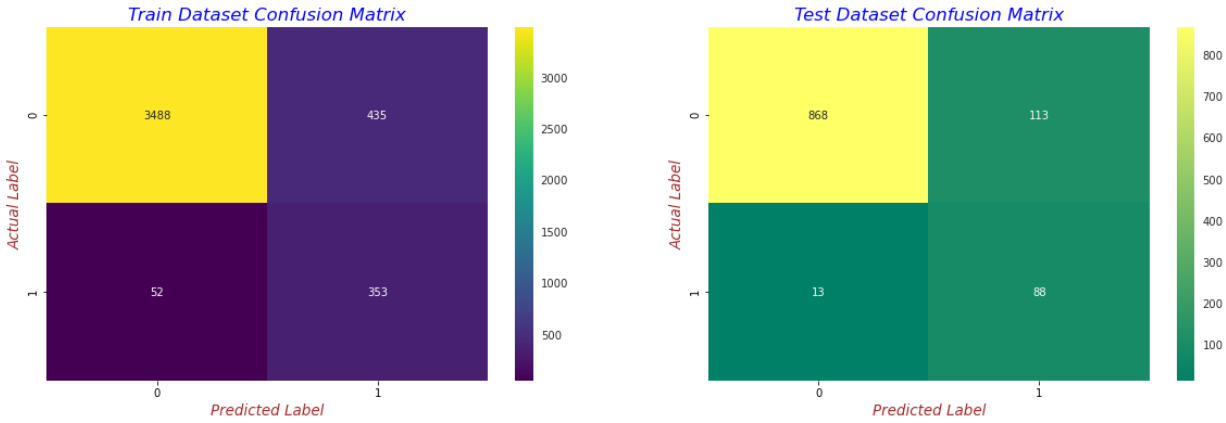
Out[ ]:
```
array([0, 0, 0, ..., 0, 0, 1], dtype=int64)
```

In [ ]:
```
# Validate model
test_auc, train_f1_score, test_f1_score, best_t = validate_model(rfc_3, X_train_std,

print("\n")
print("### Best Threshold = {:.4f}".format(best_t))
print("### Model AUC is : {:.4f}".format(test_auc))
print("### Model Train F1 Score is : {:.4f}".format(train_f1_score))
print("### Model Test F1 Score is : {:.4f}".format(test_f1_score))
```

```
### Train AUC = 0.9464100603279804
### Test AUC = 0.9509593161150978
```






```
### Best Threshold = 0.5567
### Model AUC is : 0.9510
### Model Train F1 Score is : 0.5918
### Model Test F1 Score is : 0.5828
```

In [ ]:
```
feats_imps_4 = pd.DataFrame({'Features': X_train_std.columns, 'Importance_Model_1':
feats_imps_4 = feats_imps_4[feats_imps_4['Importance_Model_1'] != 0]
feats_imps_4.reset_index(drop=True, inplace=True)
feats_imps_4.head()
```

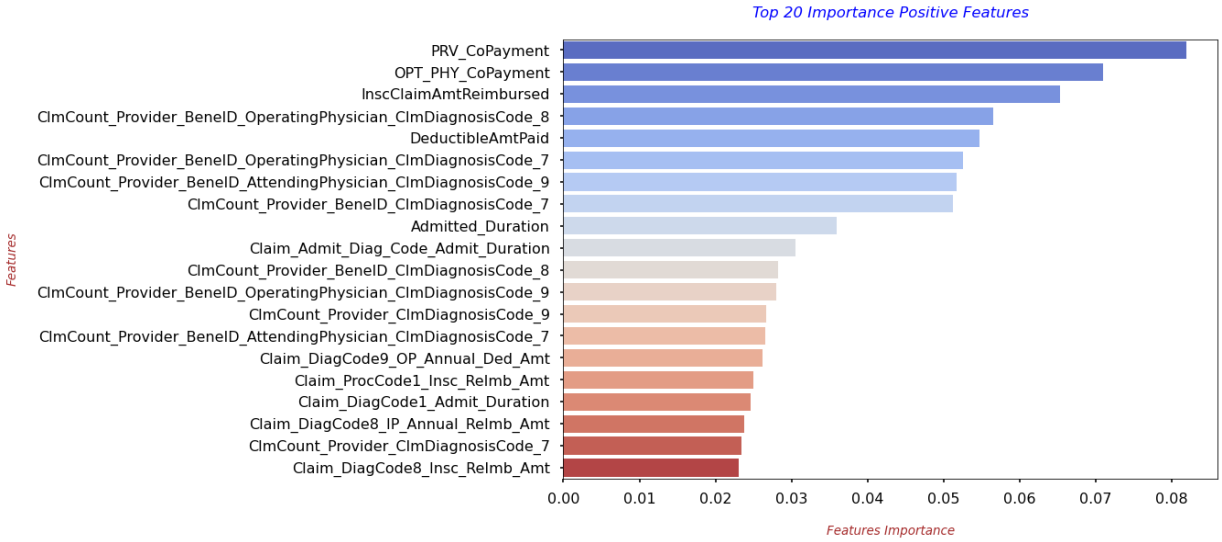
Out[ ]:

	Features	Importance_Model_1
0	InscClaimAmtReimbursed	0.065302
1	DeductibleAmtPaid	0.054782
2	RenalDiseaseIndicator	0.000297
3	ChronicCond_ObstrPulmonary	0.000880
4	ChronicCond_Osteoporasis	0.000377

In [ ]:
```
top_20_pos_feats_4 = feats_imps_4.sort_values(by='Importance_Model_1',axis=0,ascendi
```

```
top_20_pos_feats_scores_4 = feats_imps_4.sort_values(by='Importance_Model_1',axis=0,
```

In [ ]:
```
with plt.style.context('seaborn-poster'):
    sns.barplot(y=top_20_pos_feats_4, x=top_20_pos_feats_scores_4, orient='h', palet
    plt.xlabel("\nFeatures Importance", fontdict=label_font_dict)
    plt.ylabel("Features\n", fontdict=label_font_dict)
    plt.title("Top 20 Importance Positive Features\n", fontdict=title_font_dict)
```



In [ ]:
```
result_feats_scrs_4 = pd.DataFrame({'Feature': top_20_pos_feats_4, 'Imp_Score': top_
result_feats_scrs_4
```

Out[ ]:

	Feature	Imp_Score
17	PRV_CoPayment	0.081920
35	OPT_PHY_CoPayment	0.070996
0	InscClaimAmtReimbursed	0.065302
145	ClmCount_Provider_BeneID_OperatingPhysician_Cl...	0.056550
1	DeductibleAmtPaid	0.054782
144	ClmCount_Provider_BeneID_OperatingPhysician_Cl...	0.052589
142	ClmCount_Provider_BeneID_AttendingPhysician_Cl...	0.051732
153	ClmCount_Provider_BeneID_ClmDiagnosisCode_7	0.051206
10	Admitted_Duration	0.035899
50	Claim_Admit_Diag_Code_Admit_Duration	0.030555
154	ClmCount_Provider_BeneID_ClmDiagnosisCode_8	0.028236
146	ClmCount_Provider_BeneID_OperatingPhysician_Cl...	0.028039
131	ClmCount_Provider_ClmDiagnosisCode_9	0.026697
140	ClmCount_Provider_BeneID_AttendingPhysician_Cl...	0.026592
105	Claim_DiagCode9_OP_Annual_Ded_Amt	0.026252
113	Claim_ProcCode1_Insc_Relmb_Amt	0.025006
63	Claim_DiagCode1_Admit_Duration	0.024637
94	Claim_DiagCode8_IP_Annual_Relmb_Amt	0.023743
129	ClmCount_Provider_ClmDiagnosisCode_7	0.023474

	Feature	Imp_Score
92	Claim_DiagCode8_Insc_ReImb_Amt	0.023008

SUMMARY

- Here, we have the most important features on the basis of which model is giving the predictions.