Assignment - 3

Credit Risk Modelling – Loan Classification

Submitted By Mitesh Khadgi

Problem:

The Loan Classification project is designed to help you understand and apply machine learning technique to classify loan applicants based on their likelihood of loan repayment. This project will you hands-on experience in data pre-processing, feature engineering, model training, evaluation, and interpretation. CSV file is shared on whatsapp group.

Solution:

OUTPUT is mentioned in a separate "output.txt" file, and plots/figures are also uploaded:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import sklearn

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

import xgboost

from xgboost import XGBClassifier

from sklearn.preprocessing import StandardScaler, MinMaxScaler

from sklearn.model_selection import GridSearchCV, RandomizedSearchCV

from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

import warnings

warnings.filterwarnings('ignore')

df = pd.read_csv('loan_detection.csv')

df.head()

df.shape

df.columns
```

```
df.info()

#Imbalance Data

df['Loan_Status_label'].value_counts()

len(df)

print(len(df[df['Loan_Status_label'] == 0])/len(df)*100)

print(len(df[df['Loan_Status_label'] == 1])/len(df)*100)

plt.pie(df['Loan_Status_label'].value_counts(), autopct = "%1.2f%%", labels = ['Loan Not Applicable',
'Loan Applicable'])

sns.countplot(x = 'Loan_Status_label', data = df)

#Missing Data

df.isnull().sum()

df.isnull().mean()*100

sns.displot(df['previous'], label = df.previous.skew(), color = 'r')

plt.legend()

sns.displot(df['no_previous_contact'], label = df.no_previous_contact.skew(), color = 'c')

plt.legend()

sns.displot(df['not_working'], label = df.not_working.skew(), color = 'g')

plt.legend()

print(df['previous'].mean())

print(df['no_previous_contact'].mean())

print(df['not_working'].mean())

df.isna().sum()

for i in df.columns:
  if df[i].isna().sum() > 0:
    df[i].fillna(df[i].mean(), inplace = True)

df.isna().sum()

#Duplicate Data

df.duplicated().sum()

df[df.duplicated()]

#Outlier Treatment

df.describe()
```

```
Q1 = df.quantile(0.25)

Q3 = df.quantile(0.75)

IQR = Q3 - Q1

print(Q1 - 1.5*IQR)

print('\n')

print(Q3 + 1.5*IQR)

df[~((df < (Q1 - 1.5*IQR)) | (df > (Q3 + 1.5*IQR))).any(axis = 1)]

Q1 = df.quantile(0.25)

Q3 = df.quantile(0.75)

IQR = Q3 - Q1

data = df[~((df < (Q1 - 1.5*IQR)) | (df > (Q3 + 1.5*IQR))).any(axis = 1)]

#Feature Selection

data.corr()

#plt.figure(figsize = (12, 7))

plt.figure()

sns.heatmap(data.corr(), annot = True)

# data.corr['Loan_Status_label']

#Model Building

#Separate your Independent and Dependent data

data.head()

data.columns

# X = data[['previous', 'Hardness', 'Solids', Chloramines', 'Sulfate', 'Conductivity', 'Organic_carbon',
'not_working', 'Turbidity']]

X = df[['age', 'campaign', 'pdays', 'previous', 'no_previous_contact', 'not_working', 'job_admin.',
'job_blue-collar', 'job_entrepreneur', 'job_housemaid', 'job_management', 'job_retired', 'job_self-
employed', 'job_services', 'job_student', 'job_technician', 'job_unemployed', 'job_unknown',
'marital_divorced', 'marital_married', 'marital_single', 'marital_unknown', 'education_basic.4y',
'education_basic.6y', 'education_basic.9y', 'education_high.school', 'education_illiterate',
'education_professional.course', 'education_university.degree', 'education_unknown', 'default_no',
'default_unknown', 'default_yes', 'housing_no', 'housing_unknown', 'housing_yes', 'loan_no',
'loan_unknown', 'loan_yes', 'contact_cellular', 'contact_telephone', 'month_apr', 'month_aug',
'month_dec', 'month_jul', 'month_jun', 'month_mar', 'month_may', 'month_nov', 'month_oct',
'month_sep', 'day_of_week_fri', 'day_of_week_mon', 'day_of_week_thu', 'day_of_week_tue',
'day_of_week_wed', 'poutcome_failure', 'poutcome_nonexistent', 'poutcome_success']]
```

```
#X = data[['age', 'campaign', 'pdays', 'previous', 'no_previous_contact', 'not_working', 'job_admin.', 'job_blue-collar', 'job_entrepreneur', 'job_housemaid', 'job_management', 'job_retired', 'job_self-employed', 'job_services', 'job_student', 'job_technician', 'job_unemployed', 'job_unknown', 'marital_divorced', 'marital_married', 'marital_single', 'marital_unknown', 'education_basic.4y', 'education_basic.6y', 'education_basic.9y', 'education_high.school', 'education_illiterate', 'education_professional.course', 'education_university.degree', 'education_unknown', 'default_no', 'default_unknown', 'default_yes', 'housing_no', 'housing_unknown', 'housing_yes', 'loan_no', 'loan_unknown', 'loan_yes', 'contact_cellular', 'contact_telephone', 'month_apr', 'month_aug', 'month_dec', 'month_jul', 'month_jun', 'month_mar', 'month_may', 'month_nov', 'month_oct', 'month_sep', 'day_of_week_fri', 'day_of_week_mon', 'day_of_week_thu', 'day_of_week_tue', 'day_of_week_wed', 'poutcome_failure', 'poutcome_nonexistent', 'poutcome_success']]

print("X: ", X)

#y = data['Loan_Status_label']

y = df['Loan_Status_label']

print("y: ", y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_state = 42)

X_train

X_test

#Feature Scaling

from sklearn.preprocessing import StandardScaler, MinMaxScaler

sc = StandardScaler()

X_train_scale = sc.fit_transform(X_train)

X_test_scale = sc.transform(X_test)

print(X_train_scale.shape, X_train_scale)

print(X_test_scale.shape, X_test_scale)

#Using Logistic Regression

lr = LogisticRegression()

lr.fit(X_train, y_train)

print(f'Training Accuracy - {lr.score(X_train, y_train)}')

print(f'Test Accuracy     - {lr.score(X_test, y_test)}')

lr = LogisticRegression()

lr.fit(X_train_scale, y_train)

print(f'Training Accuracy - {lr.score(X_train_scale, y_train)}')

print(f'Test Accuracy     - {lr.score(X_test_scale, y_test)}')
```

```
#Using Decision Tree
dt = DecisionTreeClassifier(max_depth = 4)
dt.fit(X_train, y_train)
print(f'Training Accuracy - {dt.score(X_train, y_train)}')
print(f'Test Accuracy     - {dt.score(X_test, y_test)}')
dt = DecisionTreeClassifier(max_depth = 4)
dt.fit(X_train_scale, y_train)
y_pred_dtr = dt.predict(X_train_scale)
y_pred_dts = dt.predict(X_test_scale)
accuracy_score(y_train, y_pred_dtr)
accuracy_score(y_test, y_pred_dts)
print(f'Training Accuracy - {dt.score(X_train_scale, y_train)}')
print(f'Test Accuracy     - {dt.score(X_test_scale, y_test)}')
#Using Random Forest
rf = RandomForestClassifier(max_depth = 4, random_state = 42)
rf.fit(X_train, y_train)
print(f'Training Accuracy - {rf.score(X_train, y_train)}')
print(f'Test Accuracy     - {rf.score(X_test, y_test)}')
xgb = XGBClassifier(gamma = 0.5, reg_alpha = 0.6, reg_lambda = 0.3)
xgb.fit(X_train, y_train)
print(f'Training Accuracy - {xgb.score(X_train, y_train)}')
print(f'Test Accuracy     - {xgb.score(X_test, y_test)}')
#Using Xgboost
xgb = XGBClassifier(gamma = 0.5, reg_alpha = 0.6, reg_lambda = 0.3)
xgb.fit(X_train_scale, y_train)
y_pred_xgtr = xgb.predict(X_train_scale)
y_pred_xgts = xgb.predict(X_test_scale)
confusion_matrix(y_train, y_pred_xgtr)
sns.heatmap(confusion_matrix(y_train, y_pred_xgtr), annot = True, fmt = 'd')
accuracy_score(y_train, y_pred_xgtr)
confusion_matrix(y_test, y_pred_xgts)
```

```
sns.heatmap(confusion_matrix(y_test, y_pred_xgts), annot = True, fmt = 'd')

accuracy_score(y_test, y_pred_xgts)

plt.show()

#Hyperparameter Tuning with Xgboost

#Define the parameter grid

parameters = {

 'n_estimators': [100, 200],

 'learning_rate': [0.01, 0.05],

 'max_depth': [3, 4, 5],

 'gamma': [0.2, 0.3],

 'reg_lambda': [0.1, 1],

 'reg_alpha': [0.1, 1],

}

#Perform Grid Search

grid_search = GridSearchCV(estimator=xgb, param_grid=parameters, scoring='accuracy', cv=5,
verbose=3, n_jobs=-1)

grid_search.fit(X_train, y_train)

#Best parameters

print("Best hyperparameters: ", grid_search.best_params_)

#Best Estimator

print("Best Estimator: ", grid_search.best_estimator_)

xgb = XGBClassifier(gamma=0.2, learning_rate = 0.01, max_depth=44, n_estimators=200,
reg_Alpha=1, reg_lambda=0.1)

xgb.fit(X_train_scale, y_train)

y_pred_train = xgb.predict(X_train_scale)

y_pred_test = xgb.predict(X_test_scale)

confusion_matrix(y_train, y_pred_train)

accuracy_score(y_train, y_pred_train)

confusion_matrix(y_test,y_pred_test)

accuracy_score(y_test, y_pred_test)
```