**Credit Card Fraud Detection Predictive Models**

**Introduction**

This report aims to detect fraudulent credit card transactions using a dataset containing transactions made by European cardholders in September 2013. The dataset is highly imbalanced, with only 0.172% of the transactions being fraudulent.

**Data Description**

* **Time**: The seconds elapsed between each transaction and the first transaction in the dataset.
* **V1, V2, ..., V28**: Principal components obtained via PCA (Principal Component Analysis).
* **Amount**: The transaction amount.
* **Class**: The target variable (0 = non-fraudulent, 1 = fraudulent).

**Step 1: Load Packages**

First, we need to import the necessary Python libraries.

python

Copy code

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import plotly.graph\_objs as go

import plotly.figure\_factory as ff

from plotly.offline import init\_notebook\_mode, iplot

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import roc\_auc\_score

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier

from catboost import CatBoostClassifier

import lightgbm as lgb

from lightgbm import LGBMClassifier

import xgboost as xgb

# Initialize Plotly

init\_notebook\_mode(connected=True)

**Step 2: Load Data**

Load the dataset into a DataFrame and display the first few rows.

python

Copy code

data\_df = pd.read\_csv("../input/creditcard.csv")

print("Credit Card Fraud Detection data - rows:", data\_df.shape[0], "columns:", data\_df.shape[1])

data\_df.head()

**Step 3: Check for Missing Data**

Check if there are any missing values in the dataset.

python

Copy code

total = data\_df.isnull().sum().sort\_values(ascending=False)

percent = (data\_df.isnull().sum() / data\_df.isnull().count() \* 100).sort\_values(ascending=False)

missing\_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])

missing\_data

**Explanation:**

* data\_df.isnull().sum(): Counts the number of missing values in each column.
* sort\_values(ascending=False): Sorts the counts in descending order.
* pd.concat([total, percent], axis=1, keys=['Total', 'Percent']): Combines the total and percent missing data into a single DataFrame for easy viewing.

Since the dataset has no missing values, we can proceed to the next step.

**Step 4: Data Unbalance**

Visualize the distribution of the target variable (Class).

python

Copy code

temp = data\_df["Class"].value\_counts()

df = pd.DataFrame({'Class': temp.index, 'values': temp.values})

trace = go.Bar(

x = df['Class'], y = df['values'],

name="Credit Card Fraud Class - data unbalance (Not fraud = 0, Fraud = 1)",

marker=dict(color="Red"),

text=df['values']

)

data = [trace]

layout = dict(title='Credit Card Fraud Class - data unbalance (Not fraud = 0, Fraud = 1)',

xaxis=dict(title='Class', showticklabels=True),

yaxis=dict(title='Number of transactions'),

hovermode='closest', width=600

)

fig = dict(data=data, layout=layout)

iplot(fig, filename='class')

**Visualization:**

**Explanation:**

* data\_df["Class"].value\_counts(): Counts the occurrences of each class (0 or 1).
* go.Bar(): Creates a bar plot.
* iplot(fig, filename='class'): Displays the plot.

**Step 5: Exploratory Data Analysis (EDA)**

**Transactions Over Time**

Visualize the distribution of transactions over time for both classes.

python

Copy code

class\_0 = data\_df.loc[data\_df['Class'] == 0]["Time"]

class\_1 = data\_df.loc[data\_df['Class'] == 1]["Time"]

hist\_data = [class\_0, class\_1]

group\_labels = ['Not Fraud', 'Fraud']

fig = ff.create\_distplot(hist\_data, group\_labels, show\_hist=False, show\_rug=False)

fig['layout'].update(title='Credit Card Transactions Time Density Plot', xaxis=dict(title='Time [s]'))

iplot(fig, filename='dist\_only')

**Visualization:**

**Explanation:**

* data\_df.loc[data\_df['Class'] == 0]["Time"]: Selects 'Time' values where 'Class' is 0.
* data\_df.loc[data\_df['Class'] == 1]["Time"]: Selects 'Time' values where 'Class' is 1.
* ff.create\_distplot(): Creates a distribution plot (density plot).
* iplot(fig, filename='dist\_only'): Displays the plot.

**Transactions Count and Amount by Hour**

Group transactions by hour and calculate summary statistics.

python

Copy code

data\_df['Hour'] = data\_df['Time'].apply(lambda x: np.floor(x / 3600))

tmp = data\_df.groupby(['Hour', 'Class'])['Amount'].aggregate(['min', 'max', 'count', 'sum', 'mean', 'median', 'var']).reset\_index()

df = pd.DataFrame(tmp)

df.columns = ['Hour', 'Class', 'Min', 'Max', 'Transactions', 'Sum', 'Mean', 'Median', 'Var']

fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(18,6))

sns.lineplot(ax=ax1, x="Hour", y="Sum", data=df.loc[df.Class == 0])

sns.lineplot(ax=ax2, x="Hour", y="Sum", data=df.loc[df.Class == 1], color="red")

plt.suptitle("Total Amount")

plt.show()

fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(18,6))

sns.lineplot(ax=ax1, x="Hour", y="Transactions", data=df.loc[df.Class == 0])

sns.lineplot(ax=ax2, x="Hour", y="Transactions", data=df.loc[df.Class == 1], color="red")

plt.suptitle("Total Number of Transactions")

plt.show()

fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(18,6))

sns.lineplot(ax=ax1, x="Hour", y="Mean", data=df.loc[df.Class == 0])

sns.lineplot(ax=ax2, x="Hour", y="Mean", data=df.loc[df.Class == 1], color="red")

plt.suptitle("Average Amount of Transactions")

plt.show()

**Visualization:**

**Explanation:**

* data\_df['Time'].apply(lambda x: np.floor(x / 3600)): Converts the 'Time' column from seconds to hours.
* data\_df.groupby(['Hour', 'Class'])['Amount'].aggregate(): Groups data by 'Hour' and 'Class' and computes various statistics for the 'Amount' column.
* sns.lineplot(): Creates line plots to visualize the data.

**Box Plot of Transaction Amounts**

Create box plots to compare transaction amounts for fraudulent and non-fraudulent transactions.

python

Copy code

fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12,6))

sns.boxplot(ax=ax1, x="Class", y="Amount", hue="Class", data=data\_df, palette="PRGn", showfliers=True)

sns.boxplot(ax=ax2, x="Class", y="Amount", hue="Class", data=data\_df, palette="PRGn", showfliers=False)

plt.show()

**Visualization:**

**Explanation:**

* sns.boxplot(): Creates box plots to visualize the distribution of transaction amounts for different classes.

**Fraudulent Transactions Over Time**

Create a scatter plot for fraudulent transactions over time.

python

Copy code

fraud = data\_df.loc[data\_df['Class'] == 1]

trace = go.Scatter(

x = fraud['Time'], y = fraud['Amount'],

name="Amount",

marker=dict(color='rgb(238,23,11)', line=dict(color='red', width=1), opacity=0.5),

text=fraud['Amount'],

mode="markers"

)

data = [trace]

layout = dict(title='Amount of fraudulent transactions',

xaxis=dict(title='Time [s]', showticklabels=True),

yaxis=dict(title='Amount'),

hovermode='closest'

)

fig = dict(data=data, layout=layout)

iplot(fig, filename='fraud-amount')

**Visualization:**

**Explanation:**

* data\_df.loc[data\_df['Class'] == 1]: Selects rows where 'Class' is 1 (fraudulent transactions).
* go.Scatter(): Creates a scatter plot.
* iplot(fig, filename='fraud-amount'): Displays the plot.

**Correlation Plot**

Create a heatmap to visualize correlations between features.

python

Copy code

plt.figure(figsize=(14,14))

plt.title('Credit Card Transactions features correlation plot (Pearson)')

corr = data\_df.corr()

sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns, linewidths=.1, cmap="Reds")

plt.show()

**Visualization:**

**Explanation:**

* data\_df.corr(): Computes the correlation matrix.
* sns.heatmap(): Creates a heatmap to visualize correlations.

**Linear Regression Plots**

Show relationships between features and the transaction amount for different classes using linear regression plots.

python

Copy code

sns.lmplot(x='V20', y='Amount', data=data\_df, hue='Class', fit\_reg=True, scatter\_kws={'s': 2})

sns.lmplot(x='V7', y='Amount', data=data\_df, hue='Class', fit\_reg=True, scatter\_kws={'s': 2})

plt.show()

sns.lmplot(x='V2', y='Amount', data=data\_df, hue='Class', fit\_reg=True, scatter\_kws={'s': 2})

sns.lmplot(x='V5', y='Amount', data=data\_df, hue='Class', fit\_reg=True, scatter\_kws={'s': 2})

plt.show()

**Visualization:**

**Explanation:**

* sns.lmplot(): Creates linear regression plots to show relationships between features and the transaction amount.

**Features Density Plot**

Create density plots to compare the distribution of each feature for fraudulent and non-fraudulent transactions.

python

Copy code

var = data\_df.columns.values

t0 = data\_df.loc[data\_df['Class'] == 0]

t1 = data\_df.loc[data\_df['Class'] == 1]

sns.set\_style('whitegrid')

plt.figure()

fig, ax = plt.subplots(8, 4, figsize=(16, 28))

for i, feature in enumerate(var):

plt.subplot(8, 4, i + 1)

sns.kdeplot(t0[feature], bw=0.5, label="Class = 0")

sns.kdeplot(t1[feature], bw=0.5, label="Class = 1")

plt.xlabel(feature, fontsize=12)

locs, labels = plt.xticks()

plt.tick\_params(axis='both', which='major', labelsize=12)

plt.show()

**Visualization:**

**Explanation:**

* sns.kdeplot(): Creates density plots to compare the distribution of each feature for different classes.

**Step 6: Predictive Modeling**

**Data Preparation**

Split the data into training and testing sets.

python

Copy code

X = data\_df.drop(['Class'], axis=1)

y = data\_df['Class']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

**Explanation:**

* drop(['Class'], axis=1): Removes the 'Class' column from the features.
* train\_test\_split(): Splits the data into training and testing sets.

**RandomForestClassifier**

Train and evaluate a RandomForest classifier.

python

Copy code

rf = RandomForestClassifier(n\_estimators=100, random\_state=2018)

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

pred\_rf = rf.predict(X\_test)

print("RandomForest ROC AUC Score:", roc\_auc\_score(y\_test, pred\_rf))

**Explanation:**

* RandomForestClassifier(): Initializes the RandomForest classifier.
* fit(): Trains the model.
* predict(): Makes predictions.
* roc\_auc\_score(): Evaluates the model using ROC AUC score.

**AdaBoostClassifier**

Train and evaluate an AdaBoost classifier.

python

Copy code

adb = AdaBoostClassifier(n\_estimators=100, random\_state=2018)

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

pred\_adb = adb.predict(X\_test)

print("AdaBoost ROC AUC Score:", roc\_auc\_score(y\_test, pred\_adb))

**Explanation:**

* AdaBoostClassifier(): Initializes the AdaBoost classifier.
* The rest of the process (fit, predict, evaluate) is the same as for RandomForest.

**CatBoostClassifier**

Train and evaluate a CatBoost classifier.

python

Copy code

catb = CatBoostClassifier(iterations=100, random\_state=2018)

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

pred\_catb = catb.predict(X\_test)

print("CatBoost ROC AUC Score:", roc\_auc\_score(y\_test, pred\_catb))

**Explanation:**

* CatBoostClassifier(): Initializes the CatBoost classifier.
* The rest of the process (fit, predict, evaluate) is the same as for RandomForest.

**XGBoost**

Train and evaluate an XGBoost classifier.

python

Copy code

xgb\_model = xgb.XGBClassifier(n\_estimators=100, random\_state=2018)

xgb\_model.fit(X\_train, y\_train)

pred\_xgb = xgb\_model.predict(X\_test)

print("XGBoost ROC AUC Score:", roc\_auc\_score(y\_test, pred\_xgb))

**Explanation:**

* XGBClassifier(): Initializes the XGBoost classifier.
* The rest of the process (fit, predict, evaluate) is the same as for RandomForest.

**LightGBM**

Train and evaluate a LightGBM classifier.

python

Copy code

lgb\_model = lgb.LGBMClassifier(n\_estimators=100, random\_state=2018)

lgb\_model.fit(X\_train, y\_train)

pred\_lgb = lgb\_model.predict(X\_test)

print("LightGBM ROC AUC Score:", roc\_auc\_score(y\_test, pred\_lgb))

**Explanation:**

* LGBMClassifier(): Initializes the LightGBM classifier.
* The rest of the process (fit, predict, evaluate) is the same as for RandomForest.

**Step 7: Model Performance Comparison**

Compare the ROC AUC scores of all models.

python

Copy code

models = ['RandomForest', 'AdaBoost', 'CatBoost', 'XGBoost', 'LightGBM']

roc\_scores = [roc\_auc\_score(y\_test, pred\_rf), roc\_auc\_score(y\_test, pred\_adb),

roc\_auc\_score(y\_test, pred\_catb), roc\_auc\_score(y\_test, pred\_xgb),

roc\_auc\_score(y\_test, pred\_lgb)]

plt.figure(figsize=(10, 5))

sns.barplot(x=models, y=roc\_scores)

plt.title('ROC AUC Score Comparison')

plt.ylabel('ROC AUC Score')

plt.xlabel('Model')

plt.show()

**Visualization:**

**Conclusion**

The dataset's high imbalance posed a challenge for predictive modeling. Various models were trained and evaluated based on their ROC AUC scores, with each model demonstrating different strengths. The ensemble methods (RandomForest, AdaBoost, CatBoost, XGBoost, LightGBM) all performed well, indicating their robustness in handling imbalanced datasets.

This analysis underscores the importance of thorough EDA and the use of robust modeling techniques in detecting fraud. Future work could involve fine-tuning these models further and exploring additional techniques such as anomaly detection and deep learning.