DEVELOPMENT PART OF CREDIT CARD FRUAD DETECTION

**Introduction**

From food delivery apps to online clothing stores, the internet made it easier to purchase whatever we want whenever we want with the convenience of using our credit cards to do so. Credit cards are useful for a bunch of things, it saves us from the inconvenience of having to carry large amounts of cash with us to each place we go and it also allows us to advance a purchase that can be paid over time.

According to Statista, the number of worldwide transactions rose from 195 billion to 468 billion per year from 2014 to 2020.

# Core Features:

The system stores previous transaction patterns for each user.

Based upon the user spending ability and even country, it calculates user’s characteristics.

More than 20 -30 %deviation of users transaction(spending history and operating country) is considered as an invalid attempt and system takes action.

# The first step, of course, is importing all the necessary libraries for development,

import pandas as pd, numpy as np, matplotlib.pyplot as plt, seaborn as sns, plotly.express as px# Sklearn's preprocessing libraryfrom sklearn.preprocessing import StandardScaler# Importing train and test data splitfrom sklearn.model\_selection import train\_test\_split# Sklearn's metrics to evaluate our modelsfrom sklearn.metrics import accuracy\_score, precision\_score, confusion\_matrix, recall\_score, f1\_score# Classifiersfrom sklearn.ensemble import RandomForestClassifierfrom sklearn.ensemble import AdaBoostClassifierfrom sklearn.ensemble import GradientBoostingClassifierfrom sklearn.tree import DecisionTreeClassifier# Setting theme style and color palette to seabornsns.set\_theme(context = 'notebook', style='darkgrid',palette='muted')a

**Now, let’s check the number of occurrences of each class label and plot the information using matplotlib**.

non\_fraud = len(dataframe[dataframe.Class == 0])

fraud = len(dataframe[dataframe.Class == 1])

fraud\_percent = (fraud / (fraud + non\_fraud)) \* 100

print("Number of Genuine transactions: ", non\_fraud)

print("Number of Fraud transactions: ", fraud)

print("Percentage of Fraud transactions: {:.4f}".format(fraud\_percent))

**Let’s plot the above information using matplotlib.**

import matplotlib.pyplot as plt

labels = ["Genuine", "Fraud"]

count\_classes = dataframe.value\_counts(dataframe['Class'], sort= **True**)

count\_classes.plot(kind = "bar", rot = 0)

plt.title("Visualization of Labels")

plt.ylabel("Count")

plt.xticks(range(2), labels)

plt.show()

**Output:**



# Let’s apply scaling techniques on the “Amount” feature to transform the range of values. We drop the original “Amount” column and add a new column with the scaled values. We also drop the “Time” column as it is irrelevant.

import numpy as np

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

dataframe["NormalizedAmount"] = scaler.fit\_transform(dataframe["Amount"].values.reshape(-1, 1))

dataframe.drop(["Amount", "Time"], inplace= **True**, axis= 1)

Y = dataframe["Class"]

X = dataframe.drop(["Class"], axis= 1)

# Now, it’s time to split credit card data with a split of 70-30 using train\_test\_split().

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

(train\_X, test\_X, train\_Y, test\_Y) = train\_test\_split(X, Y, test\_size= 0.3, random\_state= 42)

print("Shape of train\_X: ", train\_X.shape)

print("Shape of test\_X: ", test\_X.shape)

# **Output:**

# initial data split

# Let’s visualize the scores of each of our credit card fraud classifiers.

decision\_tree.fit(train\_X, train\_Y)

predictions\_dt = decision\_tree.predict(test\_X)

decision\_tree\_score = decision\_tree.score(test\_X, test\_Y) \* 100

random\_forest.fit(train\_X, train\_Y)

predictions\_rf = random\_forest.predict(test\_X)

random\_forest\_score = random\_forest.score(test\_X, test\_Y) \* 100

print("Random Forest Score: ", random\_forest\_score)

print("Decision Tree Score: ", decision\_tree\_score)

# **Output:**

# initial classification models

The Random Forest classifier has slightly an edge over the Decision Tree classifier.

Let’s create a function to print the metrics: accuracy, precision, recall, and f1-score.

from sklearn.metrics import accuracy\_score, precision\_score, confusion\_matrix, recall\_score, f1\_score

**def** metrics(actuals, predictions):

print("Accuracy: {:.5f}".format(accuracy\_score(actuals, predictions)))

print("Precision: {:.5f}".format(precision\_score(actuals, predictions)))

print("Recall: {:.5f}".format(recall\_score(actuals, predictions)))

print("F1-score: {:.5f}".format(f1\_score(actuals, predictions)))

# Let’s visualize the confusion matrix and the evaluation metrics of our Decision Tree model.

confusion\_matrix\_dt = confusion\_matrix(test\_Y, predictions\_dt.round())

print("Confusion Matrix - Decision Tree")

print(confusion\_matrix\_dt)

plot\_confusion\_matrix(confusion\_matrix\_dt, classes=[0, 1], title= "Confusion Matrix - Decision Tree")

# **Output:**

# confusion matrix decision tree

|  |  |
| --- | --- |
|  | # Building Models |
|  |  |
|  | # 1. Decision Tree |
|  |  |
|  | tree\_model = DecisionTreeClassifier(max\_depth = 4, criterion = 'entropy') |
|  | tree\_model.fit(X\_train, y\_train) |
|  | tree\_yhat = tree\_model.predict(X\_test) |
|  | # 2. Logistic Regression |
|  |  |
|  | lr = LogisticRegression() |
|  | lr.fit(X\_train, y\_train) |
|  | lr\_yhat = lr.predict(X\_test) |
|  |  |
|  | # 3. K-Nearest Neighbors |
|  |  |
|  | n = 5 |
|  |  |
|  | knn = KNeighborsClassifier(n\_neighbors = n) |
|  | knn.fit(X\_train, y\_train) |
|  | knn\_yhat = knn.predict(X\_test) |
|  |  |
|  |  |
|  | # 4. SVM |
|  |  |
|  | svm = SVC() |
|  | svm.fit(X\_train, y\_train) |
|  | svm\_yhat = svm.predict(X\_test) |
|  |  |
|  | # 5. Random Forest Tree |
|  |  |
|  | rf = RandomForestClassifier(max\_depth = 4) |
|  | rf.fit(X\_train, y\_train) |
|  | rf\_yhat = rf.predict(X\_test) |
|  |  |
|  | # 6. XGBoost |
|  |  |
|  | xgb = XGBClassifier(max\_depth = 4) |
|  | xgb.fit(X\_train, y\_train) |
|  | xgb\_yhat = xgb.predict(X\_test) |
|  |  |
|  | #7. LGBMClassifier |
|  | estimator = LGBMClassifier(learning\_rate=0.22,n\_estimators =100 |
|  | ,scale\_pos\_weight= 3.5,max\_depth=16,num\_leaves=50,reg\_lambda=0.3,)# parameters for the algorithm this was done by manual tuning |
|  |  |
|  | estimator.fit( X\_train, y\_train, )# fitting on train data |
|  | estimator\_yhat = estimator.predict(X\_test) |

# EVALUATION

plt.plot(history['loss'])

plt.plot(history['val\_loss'])

plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper right');

groups = error\_df.groupby('true\_class')

fig, ax = plt.subplots()

**for** name, group **in** groups:

ax.plot(group.index, group.reconstruction\_error, marker='o', ms=3.5, linestyle='',

label= "Fraud" **if** name == 1 **else** "Normal")

ax.hlines(threshold, ax.get\_xlim()[0], ax.get\_xlim()[1], colors="r", zorder=100, label='Threshold')

ax.legend()

plt.title("Reconstruction error for different classes")

plt.ylabel("Reconstruction error")

plt.xlabel("Data point index")

plt.show();

# Conclusions:

We learned how to develop our credit card fraud detection model using machine learning. We used a variety of ML algorithms, including ANNs and Tree-based models. At the end of the training, out of 85443 validation transaction, XGBoost performs better than other models:

* Correctly identifying 111 of them as fraudulent
* Missing 9 fraudulent transactions
* At the cost of incorrectly flagging 25 legitimate transactions