Shaheed Zulfiqar Ali Bhutto Institute of Science and Technology

Department of Computer Science

Proposed Title of the Project

Credit Card Fraud Detection

Group Members Names

Touseef ahmed

Supervisor's Name

Dr. Tehreem Khan

Program: MS Data Science

Department

Computer Science

Institute

SZAIST Islamabad

Date: 1/25/2023

Credit Card Fraud Detection

Credit card fraud detection is a critical task in the financial industry, as it helps protect both cardholders and issuers from financial losses. Machine learning techniques can be used to detect fraudulent activities by analyzing patterns in transaction data.

Data Pre Processing

Pre-processing of data sets is an important step in the machine learning pipeline, as it helps to ensure that the data is in a suitable format for the chosen model. In Python, the pandas library is commonly used for data pre-processing tasks.

Some common pre-processing steps include:

Handling missing values: Missing values can be handled by dropping the rows or columns that contain missing values, or by imputing the missing values with a suitable value such as the mean or median.

Data cleaning: Data cleaning involves removing any irrelevant or duplicate data that may be present in the dataset.

Data transformation: This step involves normalizing or scaling the data to ensure that all features are on the same scale. This is important as some models are sensitive to the scale of the input features.

Encoding categorical variables: Categorical variables are variables that take on a limited number of values. They need to be encoded as numerical values before they can be used as inputs to a machine learning model. This can be done by using techniques such as one-hot encoding or label encoding.

Data split: Data split is used to divide the dataset into training and testing datasets. This is important as it allows the model to be trained and evaluated on separate data.

Data balancing: Some datasets might be imbalanced, meaning that the number of samples in each class is not equal. In such cases, techniques such as oversampling and under sampling can be used to balance the data.

These are just some examples of the pre-processing steps that can be performed on a dataset in Python using the pandas library. The specific pre-processing steps required will depend on the characteristics of the dataset and the requirements of the chosen model.

Modelling and Results – MLPClassifier and XGBoost

One approach to credit card fraud detection is to use a Multi-Layer Perceptron (MLP) classifier, which is a type of neural network that can be used for supervised learning tasks. The MLP classifier can be trained on a dataset of labeled transaction data, where the labels indicate whether a transaction is fraudulent or not. The classifier can then be used to predict the likelihood of fraud for new transactions based on their features.

Another approach is to use XGBoost (eXtreme Gradient Boosting), which is an ensemble learning method that can be used for both regression and classification tasks. XGBoost is an implementation of gradient boosting, which is a technique that combines the predictions of multiple simpler models to improve overall performance.

The Python libraries scikit-learn and xgboost can be used to implement these approaches in a credit card fraud detection system.

To create the dataset, you can use various features like the transaction amount, location of transaction, time of transaction, etc. Then, the dataset is split into training and testing datasets. The training dataset is used to train the MLPClassifier and XGBoost models. And then, the models are evaluated using the testing dataset. The model which performs better is chosen as the final model.

In the testing phase, the final model is used to predict the likelihood of fraud for new transactions. These predictions can be used to flag potentially fraudulent transactions for further investigation.

It is important to note that, credit card fraud detection is a difficult task and it requires a lot of data and computing power. In addition to this, there is a high risk of over fitting in these models, which can lead to poor performance on new data. To avoid over fitting, it is important to use techniques such as crossvalidation and regularization during model training.

Overall, MLPClassifier and XGBoost are powerful machine learning techniques that can be used to detect credit card fraud. They can be implemented in Python using the scikit-learn and XGBoost libraries, respectively. However, it is important to keep in mind the limitations and challenges of these methods, such as the risk of over fitting, when using them in practice.

Program Code:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt import seaborn as sns import
warnings
warnings.filterwarnings("ignore")

# Import packages to plot/calculate ROC and PR-AUC

from sklearn.metrics import (
precision_recall_curve,     roc_auc_score,
   roc_curve,
auc,
   brier_score_loss,
)
from sklearn.calibration import calibration_curve

data = pd.read_csv('../input/creditcardfraud/creditcard.csv')

## Custom Functions

def plt_roc_pr(y_true, predicted_proba):
   """
   Plot precision-recall and ROC curves, also
displaying the area values under each curve.
   Arguments:
      y_true: true target label
      predicted_proba: the predicted probability for the positive label
Output:
      auc_pr: the area under the precision-recall curve
      roc_auc_score_value: the area under the roc curve
   """

   # Create subplot     f, (ax1,
ax2) = plt.subplots(
      1, 2, gridspec_kw={"width_ratios": [1, 1]}, figsize=(18, 6)
   )

   # Compute precision-recall curve
   precision, recall, _ = precision_recall_curve(y_true, predicted_proba)

   # Compute area under precision-recall curve
   auc_pr = auc(recall, precision)
   print(f"PR AUC: {auc_pr:.4f}")

   # Plot PR curve
   ax1.step(recall, precision, alpha=1)
   no_gain_theoretical = np.sum(y_true) / len(y_true)

   ax1.plot(
      [no_gain_theoretical, no_gain_theoretical],
      linestyle="--",
lw=2,        color="r",
alpha=0.8,
label="No Skill",
   )
   ax1.set_xlabel("Recall")
ax1.set_ylabel("Precision")
ax1.set_title(f"Precision-Recall")
   ax1.legend(loc="upper right")

   # Compute roc curve
   fpr, recall, _ = roc_curve(y_true, predicted_proba)

   # Compute area under roc
   roc_auc_score_value = roc_auc_score(y_true, predicted_proba)
print(f"ROC AUC: {roc_auc_score_value:.3f}")

   # Plot roc curve
ax2.step(fpr, recall, alpha=1)
   ax2.plot(
      [0, 1], [0, 1], linestyle="--", lw=2, color="r", alpha=0.8, label="No Skill"
   )
   ax2.set_xlabel("FPR - false positive rate")
ax2.set_ylabel("Recall")
   ax2.title.set_text(f"ROC")

   plt.show()

   return auc_pr, roc_auc_score_value
```

<H2> Dataset exploration</H2>

```
data.head()

data.info()

data.describe()
```

Checking Null and Unique values of each column in dataset

```
data.isnull().sum()

data.nunique()
```

Spliting Data for further analysis

```
df_X = data.drop('Class',axis = 1)
df_y = data['Class']
```

# Imbalanced dataset: under-sampling data

sns.countplot(x = data['Class'])
data['Class'].value_counts()

As we can see, the data is not balanced! So let's balance it using undersampling approach

#Calculating Percentage

print("Percentage of 0 is: {zero}".format(zero = round(int(data['Class'].value_counts()[0]) / len(data['Class'])*100,2))) print('')
print("Percentage of 1 is: {one}".format(one = round(int(data['Class'].value_counts()[1]) / len(data['Class'])*100,2)))

# Resampling

from imblearn.under_sampling import NearMiss
nm = NearMiss()
X,y = nm.fit_resample(df_X,df_y)

sns.countplot(x = y)
print(X.shape,y.shape)

Now data is balanced let's implement the model

# Data Split and Scaling

from sklearn.model_selection import train_test_split from sklearn.preprocessing import RobustScaler

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

X_train.shape, X_test.shape, y_train.shape, y_test.shape

Here we can't fit to test data, or our model will have some 'spoilers' of the data that should be all new to it. So we'll just transform the train dataset

rs = RobustScaler()
X_train = rs.fit_transform(X_train)
X_test = rs.transform(X_test)

```
# Modelling and Results - MLPClassifier

from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score

mlp_model = MLPClassifier()
mlp_model.fit(X_train,y_train)

# Using the predict to calculate the confusion matrix

mlp_pred = mlp_model.predict(X_test)

# Using the predict_proba to calculate the ROC curve and PR-AUC

mlp_pred_proba = mlp_model.predict_proba(X_test) mlp_pred_proba[:,1]

# Confusion matrix report

mlp_cm = confusion_matrix(y_test,mlp_pred)
print(classification_report(y_test,mlp_pred))

# Confusion matrix heatmap

sns.heatmap(mlp_cm, annot = True, fmt = 'g', cbar = False, cmap = 'icefire')
plt.title('Confusion Matrix') plt.ylabel('Actal Values')
plt.xlabel('Predicted Values')

print(round(accuracy_score(y_test,mlp_pred),4))

# ROC curve and PR-AUC

MLP_auc_pr, MLP_roc_auc_score_value = plt_roc_pr(y_test, mlp_pred_proba[:,1])

# Modelling and Results - XGBoost

from xgboost import XGBClassifier
xgb_model = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss')

xgb_model.fit(X_train, y_train)

xgb_pred = xgb_model.predict(X_test)

from sklearn.metrics import f1_score
```

```
f1_score(y_test, xgb_pred, average='macro')

# Confusion matrix report

xgb_cm = confusion_matrix(y_test,xgb_pred)
print(classification_report(y_test,xgb_pred))

# Confusion matrix heatmap

sns.heatmap(xgb_cm, annot = True, fmt = 'g', cbar = False, cmap = 'icefire')
plt.title('Confusion Matrix') plt.ylabel('Actal Values')
plt.xlabel('Predicted Values')

# Using the predict_proba to calculate the ROC curve and PR-AUC

xgb_pred_proba = xgb_model.predict_proba(X_test) xgb_pred_proba[:,1]

# ROC curve and PR-AUC

XGB_auc_pr, XGB_roc_auc_score_value = plt_roc_pr(y_test, xgb_pred_proba[:,1])
```