CREDICT CARD FRAUD DETECTION USING MACHINE LEARNING

Problem definition:

The problem is to develop a machine learning-based system for real-time credit card fraud detection. The goal is to create a solution that can accurately identify fraudulent transactions while minimizing false positives. This project involves data preprocessing, feature engineering, model selection, training, and evaluation to create a robust fraud detection system.

Data source:

The dataset was retrieved from an open-source website, Kaggle.com. it contains data of transactions that were made in 2013 by credit card users in Europe, in two days only. The dataset consists of 31 attributes, 284,808 rows. 28 attributes are numeric variables that due to confidentiality and privacy of the customers have been transformed using PCA transformation, the three remaining attributes are “Time” which contains the elapsed seconds between the first and other transactions of each attribute, “Amount” is the amount of each transaction, and the final attribute “Class” which contains binary variables where “1” is a case of fraudulent transaction, and “0” is not as case of fraudulent transaction.

Dataset Link: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

Data preprocessing:

Data preprocessing is a process of preparing the raw data and making it suitable for a machine learning model. The structure of the dataset where all attributes are shown, with their type, in addition to glimpse of the variables within each attribute, as shown at the end of the figure the Class type is integer which I needed to change to factor and identify

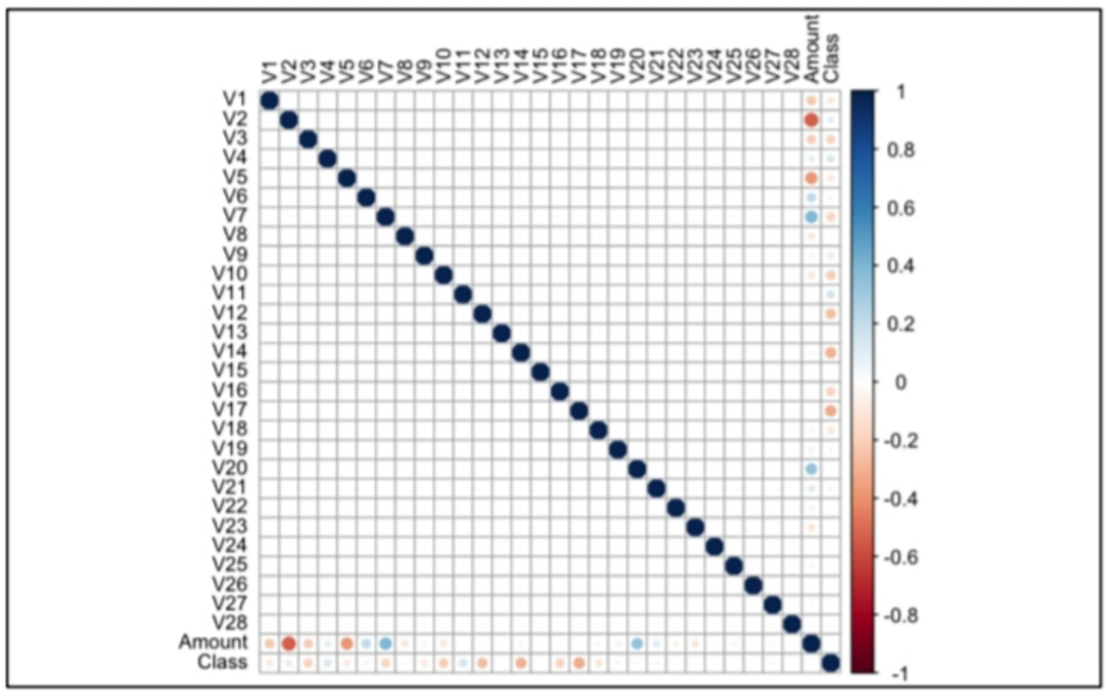
the 0 as Not Fraud and the 1 as Fraud to ease the process of creating the model and obtain visualizations .

```
'data.frame':   284807 obs. of  31 variables:
 $ Time  : num  0 0 1 1 2 2 4 7 7 9 ...
 $ V1    : num  -1.36 1.192 -1.358 -0.966 -1.158 ...
 $ V2    : num  -0.0728 0.2662 -1.3402 -0.1852 0.8777 ...
 $ V3    : num  2.536 0.166 1.773 1.793 1.549 ...
 $ V4    : num  1.378 0.448 0.38 -0.863 0.403 ...
 $ V5    : num  -0.3383 0.06 -0.5032 -0.0103 -0.4072 ...
 $ V6    : num  0.4624 -0.0824 1.8005 1.2472 0.0959 ...
 $ V7    : num  0.2396 -0.0788 0.7915 0.2376 0.5929 ...
 $ V8    : num  0.0987 0.0851 0.2477 0.3774 -0.2705 ...
 $ V9    : num  0.364 -0.255 -1.515 -1.387 0.818 ...
 $ V10   : num  0.0908 -0.167 0.2076 -0.055 0.7531 ...
 $ V11   : num  -0.552 1.613 0.625 -0.226 -0.823 ...
 $ V12   : num  -0.6178 1.0652 0.0661 0.1782 0.5382 ...
 $ V13   : num  -0.991 0.489 0.717 0.508 1.346 ...
 $ V14   : num  -0.311 -0.144 -0.166 -0.288 -1.12 ...
 $ V15   : num  1.468 0.636 2.346 -0.631 0.175 ...
 $ V16   : num  -0.47 0.464 -2.89 -1.06 -0.451 ...
 $ V17   : num  0.208 -0.115 1.11 -0.684 -0.237 ...
 $ V18   : num  0.0258 -0.1834 -0.1214 1.9658 -0.0382 ...
 $ V19   : num  0.404 -0.146 -2.262 -1.233 0.803 ...
 $ V20   : num  0.2514 -0.0691 0.525 -0.208 0.4085 ...
 $ V21   : num  -0.01831 -0.22578 0.248 -0.1083 -0.00943 ...
 $ V22   : num  0.27784 -0.63867 0.77168 0.00527 0.79828 ...
 $ V23   : num  -0.11 0.101 0.909 -0.19 -0.137 ...
 $ V24   : num  0.0669 -0.3398 -0.6893 -1.1756 0.1413 ...
 $ V25   : num  0.129 0.167 -0.328 0.647 -0.206 ...
 $ V26   : num  -0.189 0.126 -0.139 -0.222 0.502 ...
 $ V27   : num  0.13356 -0.00898 -0.05535 0.06272 0.21942 ...
 $ V28   : num  -0.0211 0.0147 -0.0598 0.0615 0.2152 ...
 $ Amount: num  149.62 2.69 378.66 123.5 69.99 ...
 $ Class : int  0 0 0 0 0 0 0 0 0 0 ...
```

Dataset Structure

Correlation between attributes “Image from R”:

The correlations between all the of the attributes within the dataset are presented in the figure below.



Correlations

Feature Engineering:

In order to create effective features for fraud detection, it is necessary to understand the underlying data and the problem that the model is trying to solve (first or third-party fraud, as well as the types of fraud therein). This often involves a combination of domain expertise, data exploration, and experimentation to identify the most relevant and informative features to use as input (often accomplished through exploratory data analysis, or “EDA”). Once these features have been identified, they may need to be transformed or combined in order to make them more useful for training a machine learning model (also included in EDA). This process of feature engineering is crucial for achieving good performance in fraud detection and other machine learning tasks.

Some of the most common patterns of features for fraud detection include:

- Affinity features built on top of rolling counter-based features — how many times an event of some kind happened in rolling windows (e.g., how many times a user has made an ATM withdrawal in this city in the last 6 months or the average transaction amount for a user with a particular credit card).
- Velocity features — how quickly events happen (e.g., the ratio of the purchases the user made in the last hour to the average number of transactions they made per hour in the last 30 days). These are also ratios of some counters, even though the focus is on capturing the velocity of the actions taken vs. the affinity between two entities.
- Reputation features — some “reputation” score for various things like the email domain of the user, the IP address the activity is coming from, the vendor the purchase was made from, etc. Some of these are, again, using counters of past behavior.
- External API-based features — e.g., the credit score of the user or location of an IP, etc.
- Relatively static profile features — e.g., the zip code from which the request originated or the age of the user’s account, etc.

As mentioned above, determining which of these features is most useful involves a lot of industry knowledge and experimentation (since the normal log and wait process would require you to wait months for results, due to the nature of the data being received and the rate a which it is received). When doing experimentation via exploratory data analysis (EDA), the engineer uses different statistics and visualizations of the data to find the best data points and combinations by looking for patterns, anomalies, and relationships between data (which can also inform the engineer that only one piece of data from the relationship needs to be used).

Model Selection:

After making sure that the data is ready to get modeled the four models were created using both Weka and R. the model SVM was created using Weka only, as for KNN, Logistic Regression and NaïveBayes they were created using R and Weka.

- ► KNN:

The k-nearest neighbors classifier (kNN) is a non-parametric supervised machine learning algorithm. It's distance-based: it classifies objects based on their proximate neighbors' classes. kNN is most often used for classification, but can be applied to regression problems as well. The K-Nearest Neighbor algorithm (KNN) is a supervised ML technique that can be applied in both scenario instances, classification instances along with regression instances.

- ► Navie Bayes:

Naïve Bayes is a classification algorithm that consider the being of a certain trait within a class is unrelated to the being of any different feature, the main use of it is for clustering and classifications, depending on the conditional probability of happening,

- ► Logistic Regression:

Logistic Regression model is statical model where evaluations are formed of the connection among dependent qualitative variable (binary or binomial logistic regression) or variable with three

values or higher (multinomial logistic regression) and one independent explanatory variable or higher whether qualitative or quantitative.

► Support Vector Machine:

Support Vector machine is a supervised ML technique with connected learning algorithms which inspect data used for both classification and regression analyses, it also performs linear classification, additionally to non-linear classification by creating margins between the classes, which are created in such a fashion that the space between the margin and the classes is maximum which minimizes the error of the classification.

Model Training:

Model training is at the heart of the data science development lifecycle where the data science team works to fit the best weights and biases to an algorithm to minimize the loss function over prediction range. Loss functions define how to optimize the ML algorithms. A data science team may use different types of loss functions depending on the project objectives, the type of data used and the type of algorithm.

When a supervised learning technique is used, model training creates a mathematical representation of the relationship between the data features and a target label. In unsupervised learning, it creates a mathematical representation among the data features themselves.

► How To Train a Machine Learning Model:

Split the Dataset:

Your initial training data is a limited resource that needs to be allocated carefully. Some of it can be used to train your model, and some of it can be used to test your model – but you can’t use the same data for each step. You can’t properly test a model unless you have given it a new data set that it hasn’t encountered before. Splitting the training data into two or more sets allows you to train and then validate the model using a single source of data. This allows you to see if the model is overfit, meaning that it performs well with the training data but poorly with the test data.

A common way of splitting the training data is to use cross-validation. In 10-fold cross-validation, for example, the data is split into ten sets, allowing you to train and test the data ten times. To do this:

1.Split the data into ten equal parts or folds.

2. Designate one fold as the hold-out fold.

3. Train the model on the other nine folds.

4. Test the model on the hold-out fold

5. Repeat this process ten times, each time selecting a different fold to be the hold-out fold. The average performance across the ten hold-out folds is your performance estimate, called the cross-validated score.

► Fit and Tune Models:

Now that the data is prepared and the model's hyperparameters have been determined, it's time to start training the models. The process is essentially to loop through the different algorithms using each set of hyperparameter values you've decided to explore. To do this:

1.Split the data

2. Select an algorithm.

3. Tune the hyperparameter values.

4. Train the model. 5. Select another algorithm and repeat steps 3 and 4..

▸ Choose the Best Model:

Now it's time to test the best versions of each algorithm to determine which gives you the best model overall

1. Make predictions on your test data.

2. Determine the ground truth for your target variable during the training of that model.

3. Determine the performance metrics from your predictions and the ground truth target variable.

4. Run each finalist model with the test data.

Evaluation:

The last stage of the CRISP-DM model is the evaluation and deployment stage, as presented in table 2 below all models are being compared to each other to figure the best model in identifying fraudulent credit card transactions. Accuracy is the overall number of instances that are predicted correctly, accuracies are represented by confusion matrix where it showed the True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN). True Positive represents the transactions that are fraudulent and was correctly classified by the model as fraudulent. True Negative represents the not fraudulent transactions that were correctly predicted by the model as Not fraudulent. The third rating is False positive which represents the transaction that are fraudulent but was misclassified as not fraudulent. And finally False Negative which are the not fraudulent transactions that were identified as fraudulent, table 1 below shows the confusion matrix.

Actual/Predicted	Positive	Negative
positive	TP	FN
Negative	FP	TN

Confusion Matrix

The table above shows all the components to calculate an accuracy of a model which is displayed in the below equation.

$$Accuracy = TP + TN / TP + TN + FP + FN$$