UNIVERSITY OF PISA

DEPARTMENT OF INFORMATION ENGINEERING

Data Mining and Machine Learning

Credit Card Fraud Detection

Work group:

Valentina Bertei

Francesco Tarchi

ACADEMIC YEAR 2024/2025

Contents

1 Introduction **4**
1.1 GitHub Repository 4

2 Dataset and Preprocessing **5**
2.1 Dataset 5
2.2 Preprocessing 6

3 Classifiers **9**
3.1 Grid Search 9
3.2 Training & Testing 9
3.3 Evaluation metrics 10
3.3.1 Acceptable Level of Performance 11

4 Experimental Results **12**
4.1 Results of the Models 12
4.2 Comparison of the Models 13
4.3 Explainability of the Models 17
4.3.1 Feature Importances 17
4.3.2 SHAP Values 19
4.3.3 Permutation Feature Importance 22
4.3.4 Rule Extraction via Surrogate Models 25

5 Real-world Application **28**

6 Conclusions **30**

List of Figures

2.1 Distribution of transactions: *fraudulent* vs *legitimate*. 6
2.2 Comparison of the class distribution in the training set before (left) and after (right) the application of SMOTE. 7
2.3 Preprocessing pipeline. 8

4.1 Comparison of the *accuracy* among all the classifiers. 13
4.2 Comparison of the *balanced accuracy* among all the classifiers. 13
4.3 Comparison of the *recall* among all the classifiers. 14
4.4 Comparison of the *precision* among all the classifiers. 14
4.5 Comparison of the *f1* among all the classifiers. 15
4.6 *PR Curves* of all the classifiers. 15
4.7 *ROC Curves* of all the classifiers. 15
4.8 Thresholds used to compute the *ROC* values plotted against the *TPR* values: the dots on the curves represents the *ALP* for each classifier. . 16
4.9 Thresholds used to compute the *ROC* values plotted against the *FPR* values: the dots on the curves represents the *ALP* for each classifier. . 16
4.10 Top 20 features ranked by intrinsic feature importance as computed by the AdaBoost classifier on the preprocessed training set. 17
4.11 Top 20 features ranked by intrinsic feature importance as computed by the Decision Tree classifier on the preprocessed training set. 17
4.12 Top 20 features ranked by intrinsic feature importance as computed by the Random Forest classifier on the preprocessed training set. . . . 18
4.13 Top 20 features ranked by intrinsic feature importance as computed by the XGBoost classifier on the preprocessed training set. 18
4.14 SHAP summary plot for the XGBoost model, showing the global impact of the top features on the model predictions. Positive values indicate a contribution towards predicting fraud, while negative values indicate a contribution towards predicting legitimate transactions. 19
4.15 SHAP summary plot for the Decision Tree classifier. 19
4.16 SHAP summary plot for the Gaussian NB classifier. 20
4.17 SHAP summary plot for the Random Forest classifier. 20
4.18 SHAP summary plot for the AdaBoost classifier. 21
4.19 Permutation feature importances for the Decision Tree classifier. . . . 22
4.20 Permutation feature importances for the Gaussian Naive Bayes classifier. 22
4.21 Permutation feature importances for the XGBoost classifier. 23
4.22 Permutation feature importances for the Random Forest classifier. . . 23
4.23 Permutation feature importances for the AdaBoost classifier. 24
4.24 Permutation feature importances for the K-NN classifier. 24

5.1 Screenshot of the Streamlit interface used for real-time fraud detection. Users can input a transaction and see predictions from all models along with interpretability explanations. 28
5.2 Screenshot of the application displaying explanations for each classifier. For tree-based models (Decision Tree, Random Forest, XGBoost) SHAP values highlight feature contributions, AdaBoost uses a Kernel SHAP explainer, Naive Bayes shows posterior probabilities, and K-Nearest Neighbors presents the most similar examples from the training set. 29

1. Introduction

Credit Card Fraud Detection is a simple security application which allows a user to manually input transaction details, which are then classified as Fraudulent or Legitimate by a set of classifiers. The key idea behind this project is to provide both performance comparison and explainability of the classification process. This way, the user can not only see which transactions are flagged as fraudulent, but also understand the reasoning behind each classification.

In particular, the application implements six classifiers: Decision Tree, Random Forest, Naive Bayes, K-Nearest Neighbors (KNN), AdaBoost, and XGBoost. For each classifier, the system provides visualizations of performance metrics as well as explanations for the classification decisions, allowing the user to compare the models not only in terms of accuracy, precision, recall, and F1-score, but also in terms of interpretability.

The application is developed using Streamlit, providing an interactive and user-friendly interface for entering transactions and observing the results.

1.1 GitHub Repository

All the codes and the raw materials (apart from the dataset) can be found at the following repository.

`https://github.com/francetarchi/CreditCardFraudDetection`

2. Dataset and Preprocessing

2.1 Dataset

The dataset used for this application comes from the *IEEE-CIS Fraud Detection* competition on Kaggle [1]: it contains historical online transaction data enriched by behavioral signals and device information. The dataset includes 590,540 transactions (1.08 GB), of which 20,663 are labeled as fraudulent (approximately 3.5%). The binary target variable is `isFraud`, where 1 represents a *fraudulent* transaction and 0 a *legitimate* one.

The dataset appears strongly imbalanced due to the small proportion of fraudulent transactions. As a consequence, proper evaluation metrics were used to assess the performance of the models, instead of relying solely on *accuracy*.

The dataset contains 432 features, grouped according to their type and origin.

- **Transaction features:** These include transaction amount (`TransactionAmt`) and a timedelta feature (`TransactionDT`) representing the time difference from a reference date. Product codes are included via `ProductCD`.
- **Address features:** Billing and mailing information, such as `addr1`, `addr2`, and distances between addresses (`dist` features).
- **Email domain features:** Recipient and purchaser email domains (`Remaildomain` and `Pemaildomain`).
- **Card features:** Payment card information including type, category, issuing bank, and country (`card1`–`card6`).
- **Counting features:** Features `C1-C14` representing counts, with masked meaning for privacy reasons.
- **Timedelta features:** Features `D1-D15` representing time differences between events.
- **Match features:** Features `M1-M9` describing matches, e.g., between cardholder name and address.
- **Identity features:** Features `id01-id38` and `Device`, `DeviceInfo`, capturing device information, behavioral signals, and security ratings.
- **Vesta engineered features:** Features `V1-V339`, numeric features engineered by Vesta, including rankings, counts, and entity relations.

The input to the models consists of multivariate features including transaction details, card and address information, identity data, and engineered Vesta features. The output is the binary class label `isFraud` $\in \{0, 1\}$.

A graphical representation of the dataset distribution (fraudulent vs legitimate[1] transactions) is reported in Figure 2.1.

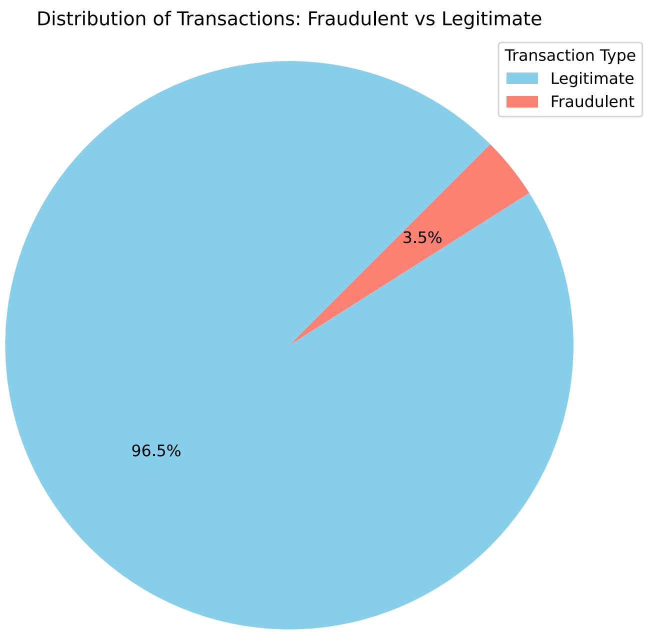


Figure 2.1: Distribution of transactions: *fraudulent* vs *legitimate*.

2.2 Preprocessing

The raw dataset underwent several preprocessing steps to prepare it for the training phase.

1. First, we removed three **unnecessary columns** (an index column, the `TransactionID_x` column and the `TransactionID_y` column) and engineered additional **temporal features** from the `TransactionDT` variable, which originally represented a cumulative count of seconds. To better capture temporal patterns, we extracted the day, the hour of the transaction, and the day of the week, together with their cyclic encodings using sine and cosine transformations.

2. The dataset was then split into **training** and **testing** sets, preserving the label distribution through *stratified sampling*.

3. **Missing values** in the numerical columns were handled using a *median imputation*.

4. Then, a `RobustScaler` was applied to reduce the influence of outliers during the training phase.

[1]In next paragraphs, legitimate transactions are often assessed as *nonfraudulent* transactions.

5. **Feature selection** was performed in two steps: first, a *variance threshold filter* was applied to remove features with very low variability; then, we applied `SelectKBest` with the *Mutual Information score* to select the `k=120`[2] most informative features for the classification task. This allowed us to reduce the dimensionality of the dataset and retain only the features that contribute most to the discrimination between fraudulent and legitimate transactions.

6. Since the dataset is highly imbalanced (only about 3.5% of the samples are fraudulent), we applied the **Synthetic Minority Oversampling Technique** (**SMOTE**) to the training set. This procedure increased the proportion of fraudulent transactions by generating synthetic samples. In our experiments, we applied SMOTE oversampling only on the training set, as oversampling the test set would artificially alter the real-world class distribution and lead to biased evaluation. We decided to apply SMOTE with a sampling strategy of 0.2, which generates a number of synthetic fraudulent transactions equal to 20% of the majority class size. As a result, the proportion of fraudulent transactions in the training set increased from 3.5% to approximately 16.7%. A stronger oversampling (e.g., up to a balanced 50-50 distribution) was tested, but it did not improve the evaluation metrics while unnecessarily inflating the dataset size and training time.

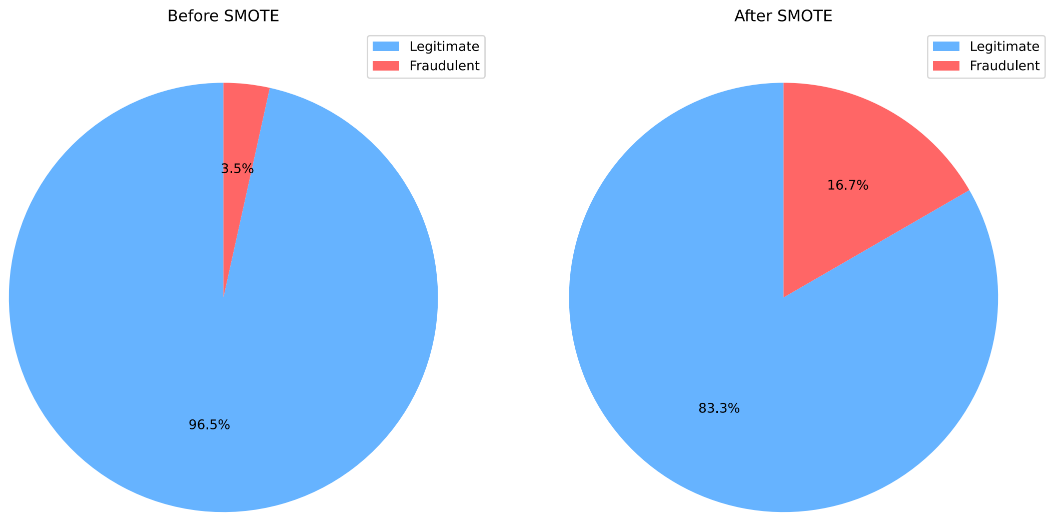


Figure 2.2: Comparison of the class distribution in the training set before (left) and after (right) the application of SMOTE.

[2]The value of `k` has not been chosen randomly: we plotted a graph with all the features ordered from the less informative to the most informative, based on *Mutual Information scores*. Such graph had an elbow shape around the 120th feature, which also was around the threshold of `MI-score = 0.01`.

Finally, the preprocessed training, testing, and resampled datasets were saved for later use, together with the fitted preprocessing objects (imputer, scaler, feature selectors), so that the same transformations could be consistently applied during the evaluation phase.

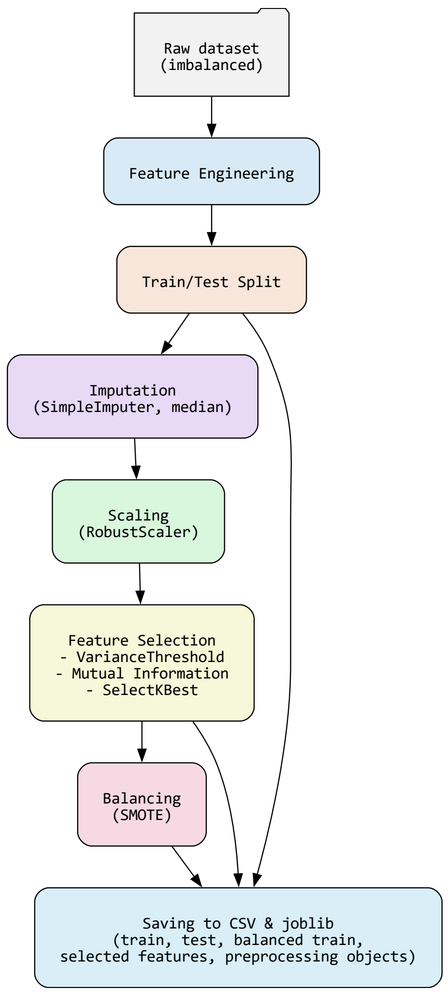


Figure 2.3: Preprocessing pipeline.

3. Classifiers

One of the main goals of this project was to compare several classifiers in order to verify which one had the best performances on the dataset. We chose to compare the following models:

- *K-Nearest Neighbours*, possibly referred from here on as *KNN*;
- *Naive Bayes*, possibly referred from here on as *NB*;
- *DecisionTree*, possibly referred from here on as *DT*;
- *RandomForest*, possibly referred from here on as *RF*;
- *AdaBoost*, possibly referred from here on as *ADA*;
- *XGBoost*, possibly referred from here on as *XGB*.

3.1 Grid Search

In order to take each classifier to his own best performance on our dataset, we applied a *Grid Search* to find the best hyper-parameters for each of them. We used the `GridSearchCV` class provided by the `scikit-learn` library.

We have set the `cv` parameter of the `GridSearchCV` class to `5`, meaning that a *Cross-Fold Validation* with `k=5` has been done during this process to obtain a more robust choice of the best hyper-parameters.

We have chosen *f1* as scoring metric of the `GridSearchCV` class because we needed a metric that suffered less the strongly imbalanced dataset: the *accuracy* was not the right choice for this exact reason and the *f1* was the perfect balance between *precision* and *recall* (the two metrics on which we focused during the examination of the results).

3.2 Training & Testing

After the identification of the best hyper-parameters, each classifier has been trained on the *rebalanced train set* (*SMOTE* at 20% ratio).

During the training phase, another *Cross-Fold Validation* has been applied in order to obtain more robust validation metrics, this time with the number of folds `k=10` (for the *Grid Search* we used `k=5` to speed up a bit the process).

After the training phase, each classifier has been tested on the *imbalanced test set* and the results of the predictions have been used to obtain the evaluation metrics.

3.3 Evaluation metrics

Before listing all the metrics we used, we need to clarify that we wanted our classifier to correctly classify the fraudulent transactions as main goal (increase *TP* and decrease `FN`); after that, we wanted them to do so with the smallest possible number of non-fraudulent transactions misclassified (decrease *FP*).

For these reasons, the evaluation metrics on which we mainly focused were *precision* and *recall*, even though we have calculated the following evaluation metrics for each classifier.

- *Confusion matrix*: they have been used to obtain a quick look to the performances of each classifier and as "main metric" to obtain more specific ones.

$$\begin{bmatrix} TN & FP \\ FN & TP \end{bmatrix}$$

- *Accuracy*: even if it is not so suitable for strongly imbalanced datasets (as ours is), it is however a good metric to quickly evaluate the general performance of a classifier.
- *Balanced accuracy*: defined as *(specificity + recall)/2*, it is more robust to imbalanced datasets then the "normal" *accuracy*.
- *Precision*: one of the two metrics on which we focused to improve classifiers performances.
- *Weighted precision*: the weighted version of *precision*.
- *Recall*: the metric on which we firstly focused to improve classifiers performances.
- *Weighted recall*: the weighted version of *recall*.
- *F1*: mainly used as scoring metric of the *Grid Search* to order models considering both *recall* and *precision*.
- *Weighted f1*: the weighted version of *f1*.
- *ROC_AUC*: the area under the *ROC Curve* of each classifier, mainly used at the end of the project to compare all the classifiers' performances.
- *PR_AUC*: the area under the *PR Curve* of each classifier, mainly used at the end of the project to compare all the classifiers' performances.

3.3.1 Acceptable Level of Performance

The *ROC Curves* graph and the *PR Curves* graph (as we will see in section 4.1) gave us some interesting results about the adaptation of the classifiers to different values of the *decision threshold*[1]: if we modify such threshold, all the classifiers improve their own *TPR*, but always worsening *FPR*. Consequently, we decided that we consider "a good classifier" a model that has at least a *TPR* of `0.8` (that is a classifier which can identify correctly the 80% of fraudulent transactions), because the main goal of a fraud detector should be to identify as much frauds as possible, even accepting a considerable amount of misclassified non-fraud transactions[2].

We called **Acceptable Level of Performance** (**ALP**) the condition in which a classifier reaches `tpr = 0.8`.

We called **ALP_threshold** the threshold at which a classifier reaches the *ALP*.

We called **ALP_FPR** the *FPR* of a classifier when it reaches *ALP*.

We made a further analysis of the *ROC Curves* graph to obtain the *ALP* for each classifier[3] and find the classifier with the smallest *ALP_FPR*, plotting a *Thresholds VS TPR* graph and a *Thresholds VS FPR* graph. Such graphs helped us to identify the model with the best trade-off between *TPR* and *FPR* at *ALP*.

[1]With *decision threshold* we mean the value `t` used by each classifier to consider as 1 or 0 the probability of a tuple of being positive. By default, `scikit-learn` uses `t=0.5`, meaning that, if a tuple has a probability `p >= 0.5` of being positive, it will be classified as positive (label 1). From now on, we will refer to the *decision threshold* simply as *threshold*.

[2]Obviously, the *ALP* has been chosen reasonably to obtain good performances, but is not mandatory to choose our value: it is just an example to show that the classifiers can reach even better performances than those with the default threshold. The *ALP* could be increased more to obtain even better *recall* (but with more *FP*), or could be decreased to obtain less *FP* (but with less *recall*), depending on what the final user of the application wants.

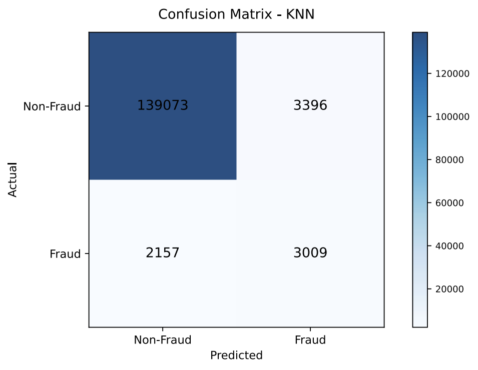
[3]Due to a lack of time, the classifier have not been re-trained using their own *ALP_threshold*: we have just found the *ALP* for each classifier and reported them in the present paper. The models that the user finds in our real world application are trained with the `scikit-learn` default threshold `t = 0.5`.

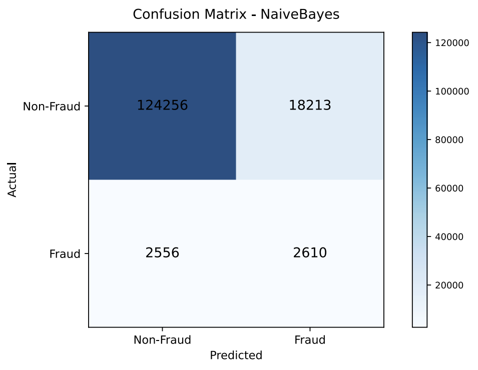
4. Experimental Results

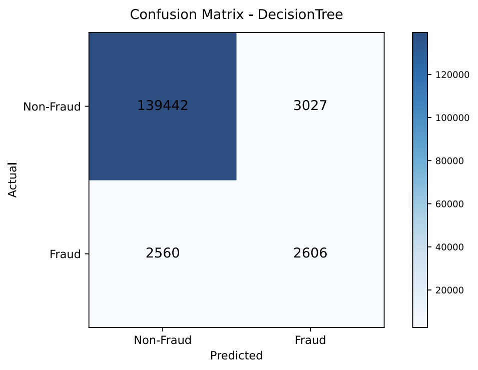
In this chapter we include both the results of each classifier (with the comparison among all of them) and the results of the *XAI methods* we used to explain the classifiers.

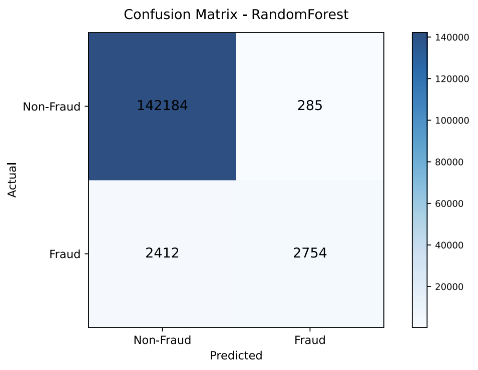
4.1 Results of the Models

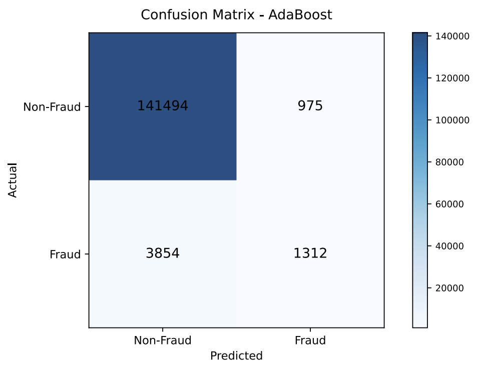
The results of the model are presented individually as *confusion matrices* of their predictions.

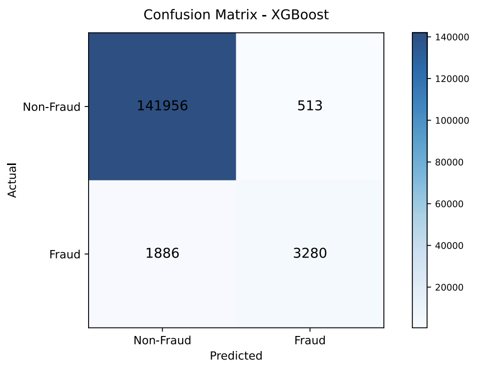












4.2 Comparison of the Models

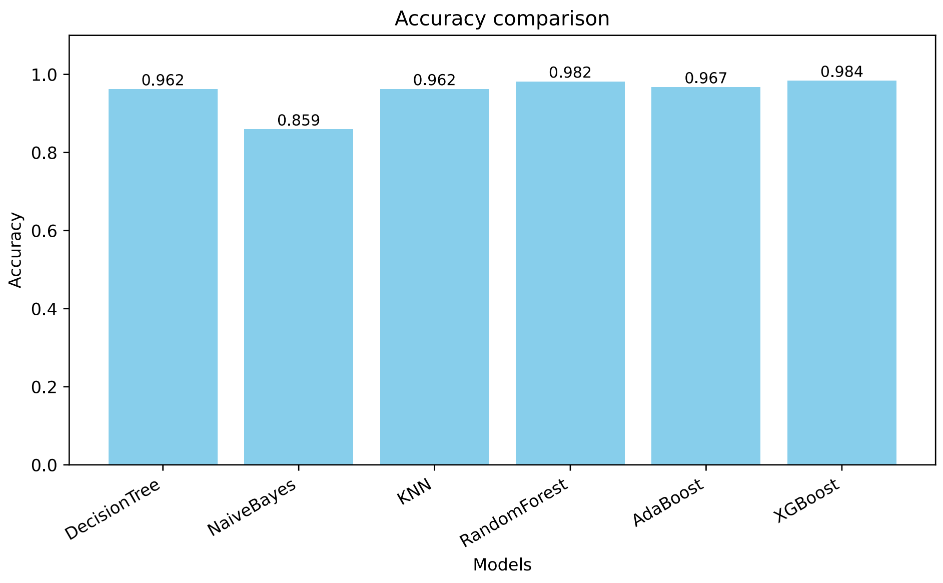


Figure 4.1: Comparison of the *accuracy* among all the classifiers.

We can easily see how the *accuracy* metric is really high in all the classifiers, with only NaiveBayes a bit lower than the others.

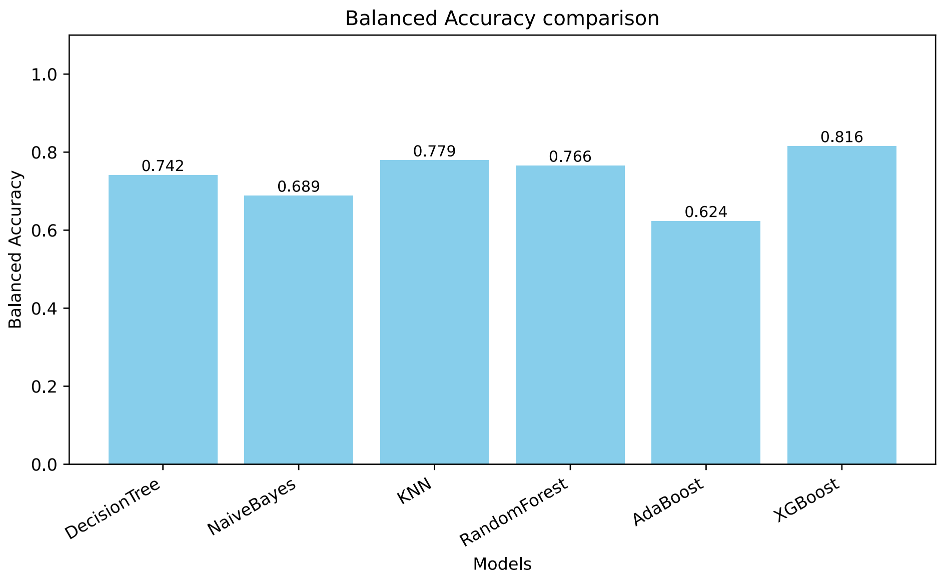


Figure 4.2: Comparison of the *balanced accuracy* among all the classifiers.

We can easily see how the *balanced accuracy* metric is lower in all the classifiers w.r.t the "normal" *accuracy*: the advantage of **XGBoost** on the other classifiers increases and AdaBoost becomes the worst.

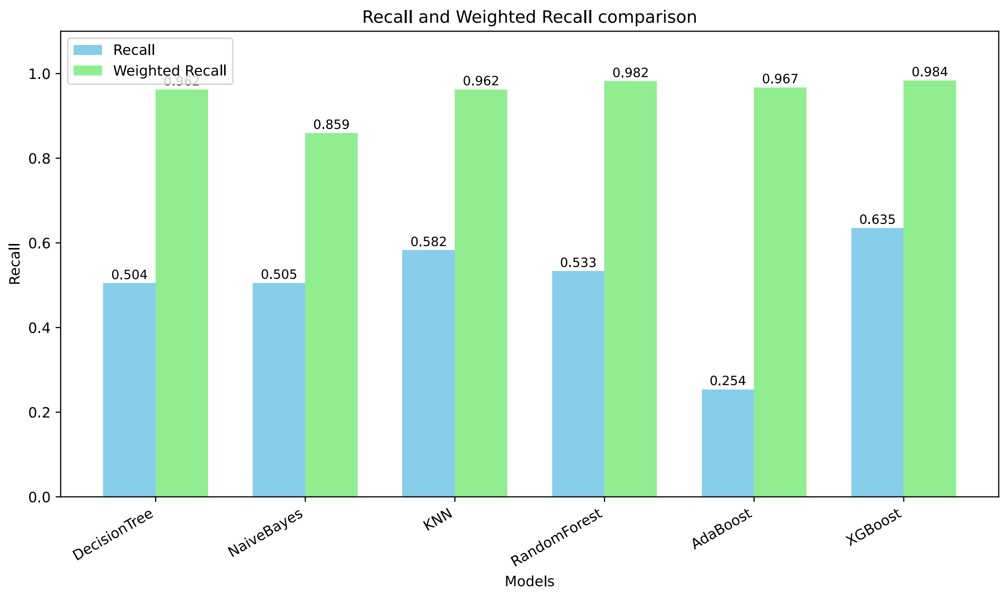


Figure 4.3: Comparison of the *recall* among all the classifiers.

We can see how **XGBoost** has the best *recall*, followed by KNN and Random-Forest. Instead, all the classifiers are comparable in the *weighted recall*, with only NaiveBayes a bit worse than the others.

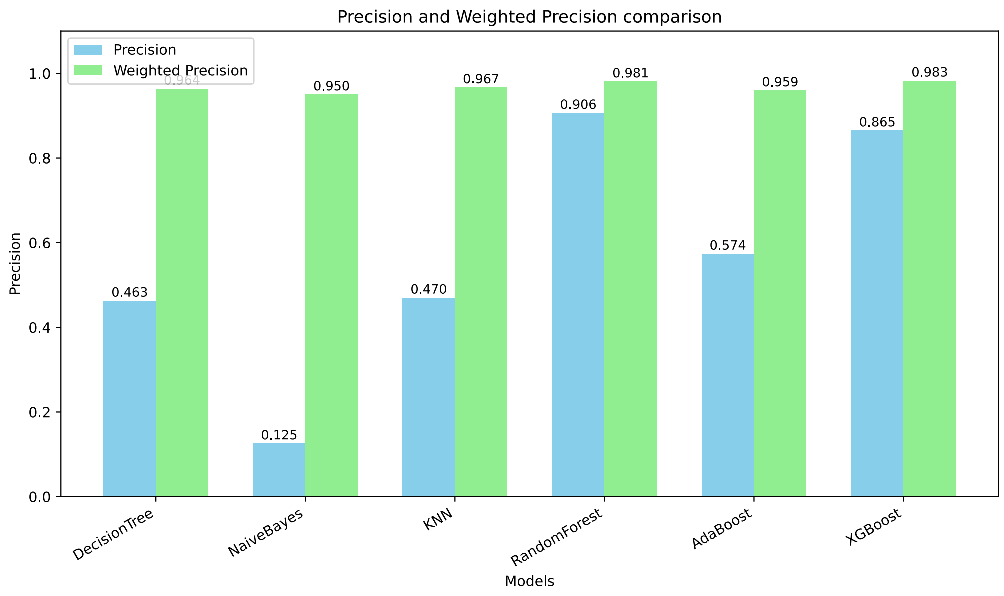


Figure 4.4: Comparison of the *precision* among all the classifiers.

We can see how **RandomForest** has the best *precision*, followed only by XG-Boost, while the other classifiers have much lower metrics, especially NaiveBayes. Instead, all the classifiers are comparable in the *weighted precision*.

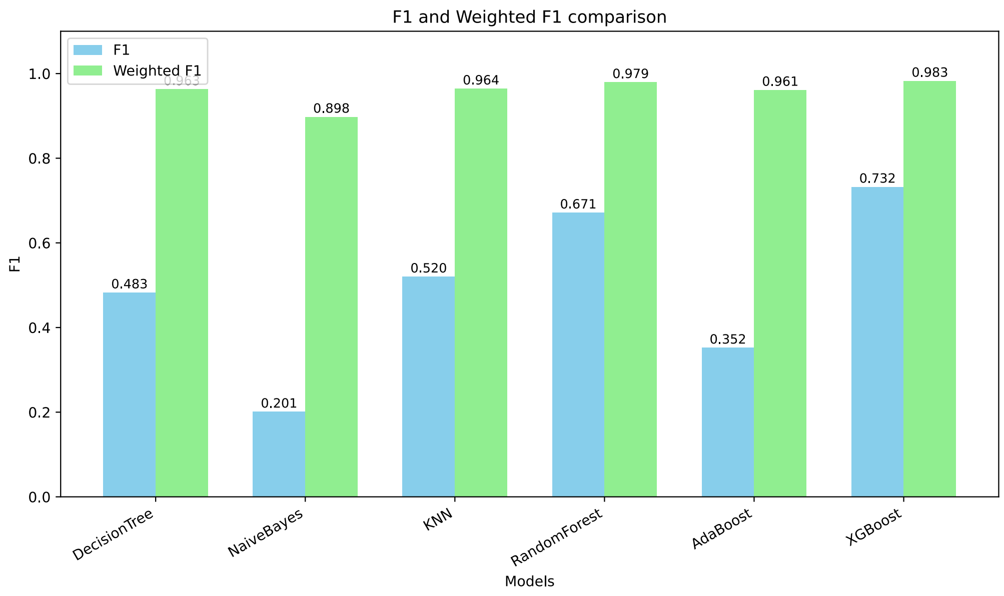


Figure 4.5: Comparison of the *f1* among all the classifiers.

We can see how **XGBoost** has the best *f1* score, closely followed only by RandomForest, while NaiveBayes and AdaBoost are by far the worst (KNN and DecisionTree are about in the middle between the best models and the worst models). Instead, all the classifiers are comparable in the *weighted f1*.

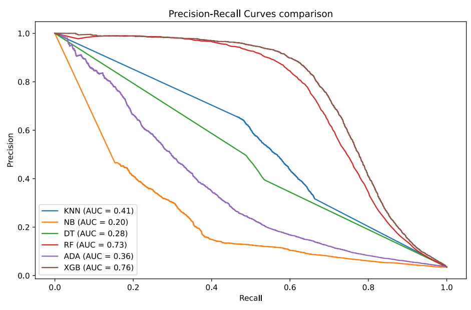


Figure 4.6: *PR Curves* of all the classifiers.

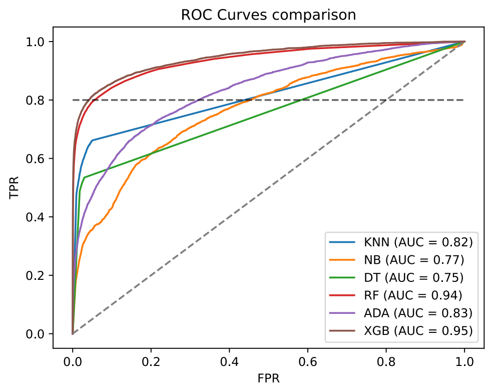


Figure 4.7: *ROC Curves* of all the classifiers.

Both in *PR Curves* and in *ROC Curves*, we see that the the two best[1] classifiers are by far **XGBoost** and **RandomForest**, with the first one with a small advantage on the second one (clearly visible both from the graph and the AUC values).

[1]We recall that "the best" means:

- for the *PR Curves*, the curve that is the nearest to the upper right angle of the graph;
- for the *ROC Curves*, the curve that is the nearest to the upper left angle of the graph;
- for both *PR Curves* and *Roc Curves*, the curve which has the largest area underlying, that is the biggest AUC value.

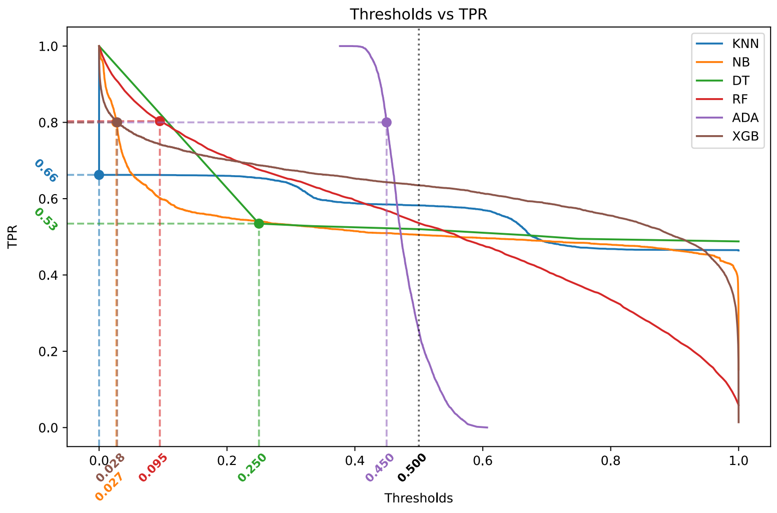


Figure 4.8: Thresholds used to compute the ROC values plotted against the TPR values: the dots on the curves represents the ALP for each classifier.

We can see how KNN and DecisionTree cannot reach the ALP[2]. We then see that AdaBoost reaches ALP really quickly ($ALP\_threshold$ near 0.5), while XGBoost and RandomForest need a very low $ALP\_threshold$.

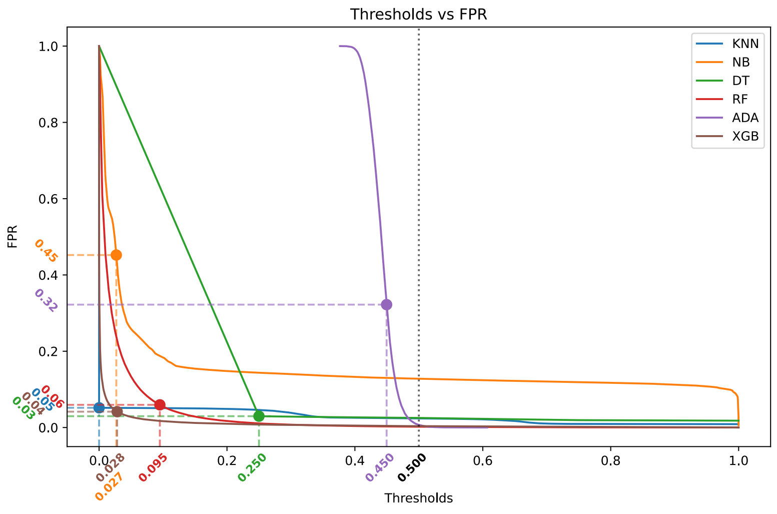


Figure 4.9: Thresholds used to compute the ROC values plotted against the FPR values: the dots on the curves represents the ALP for each classifier.

We can see that at the $ALP\_thresholds$ the classifiers with the lowest ALP_FPR are **XGBoost**, **RandomForest**, KNN and DecisionTree, but the last two classifiers could not reach ALP.

[2]The straight lines from the blue and green dots the point `(0,1)` depend on the implementation of `matplotlib`, which draws on the graphs all the input points and then connects them with straight lines.

4.3 Explainability of the Models

To better understand the predictions of the models and provide insights into the factors influencing the classification, several explainability techniques were applied to all the trained classifiers.

4.3.1 Feature Importances

For models supporting intrinsic feature importance metrics, such as DecisionTree, RandomForest and XGBoost, the top features contributing to the classification were computed and visualized. Horizontal bar plots highlight the most relevant features, providing an immediate understanding of which variables drive the predictions.

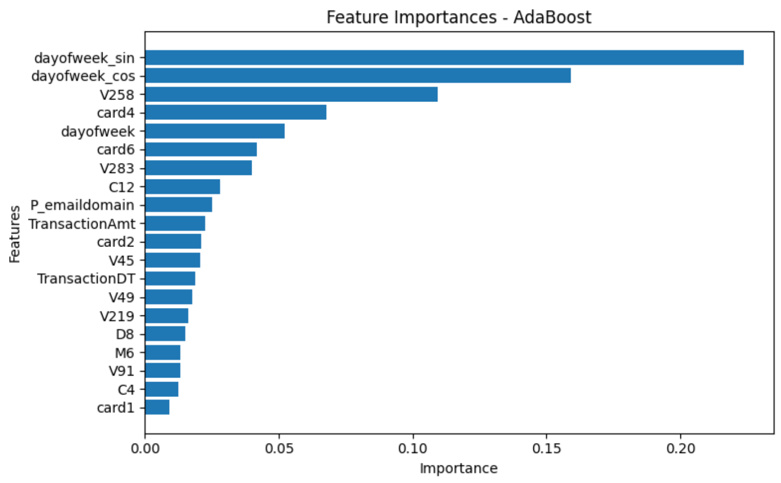


Figure 4.10: Top 20 features ranked by intrinsic feature importance as computed by the AdaBoost classifier on the preprocessed training set.

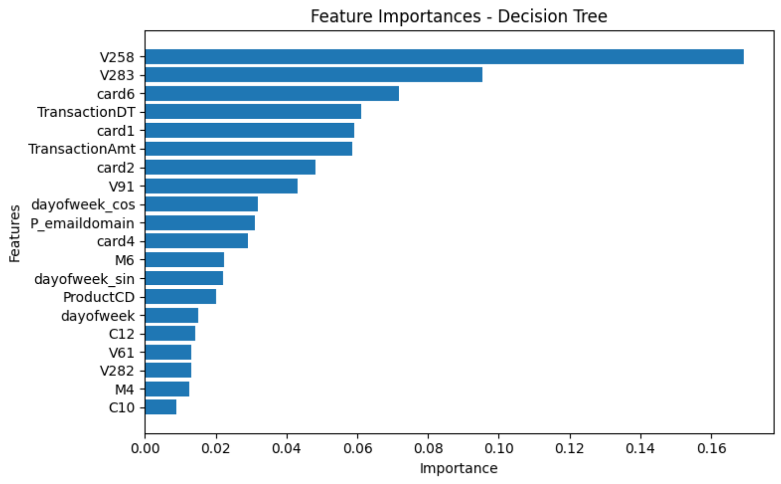


Figure 4.11: Top 20 features ranked by intrinsic feature importance as computed by the Decision Tree classifier on the preprocessed training set.

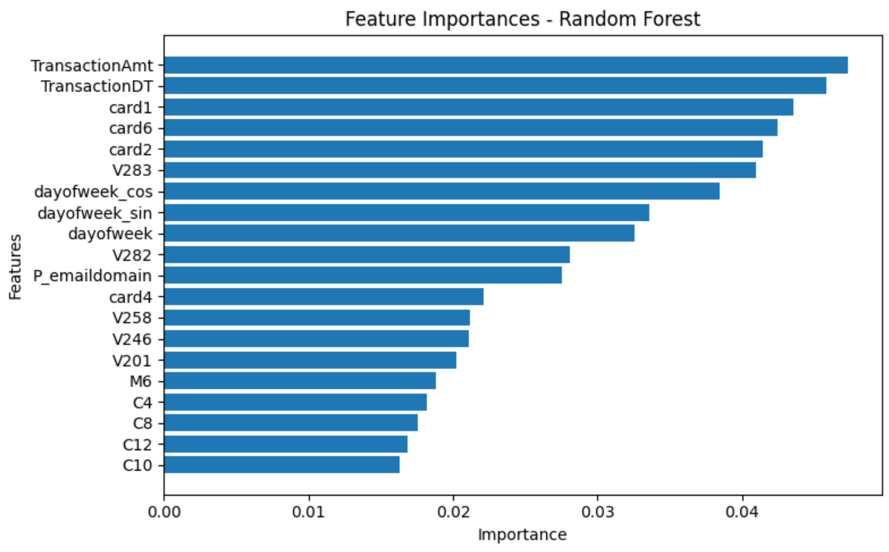


Figure 4.12: Top 20 features ranked by intrinsic feature importance as computed by the Random Forest classifier on the preprocessed training set.

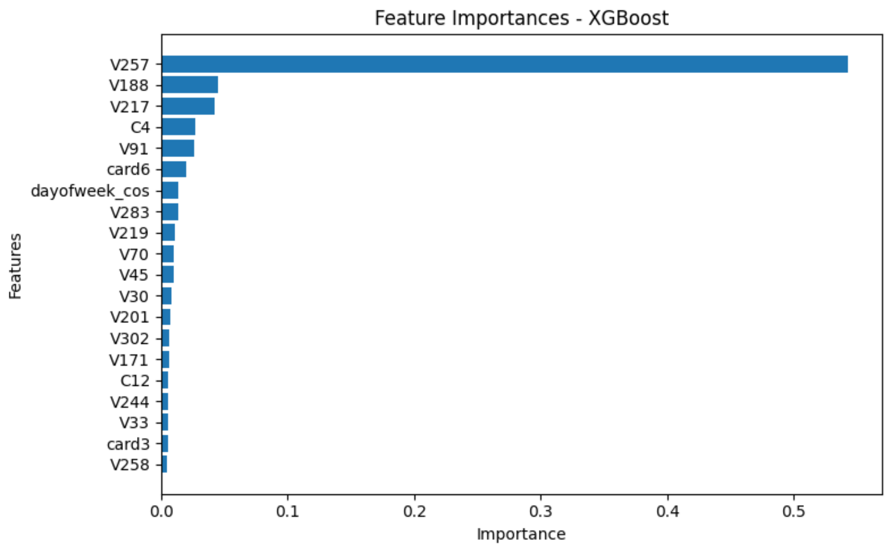


Figure 4.13: Top 20 features ranked by intrinsic feature importance as computed by the XGBoost classifier on the preprocessed training set.

4.3.2 SHAP Values

SHAP (SHapley Additive exPlanations) values were computed, whenever feasible, using a stratified sample of the testing set. SHAP values allow to quantify the contribution of each feature to individual predictions as well as to the overall behavior of the model. Summary plots were generated to illustrate the global impact of features across the dataset.

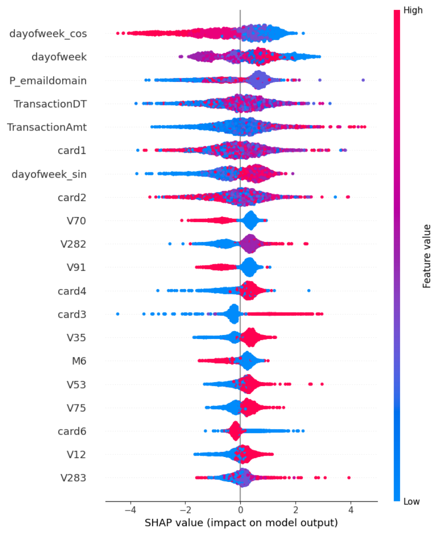


Figure 4.14: SHAP summary plot for the XGBoost model, showing the global impact of the top features on the model predictions. Positive values indicate a contribution towards predicting fraud, while negative values indicate a contribution towards predicting legitimate transactions.

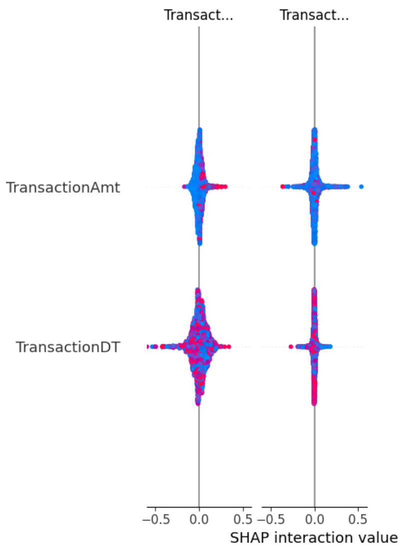


Figure 4.15: SHAP summary plot for the Decision Tree classifier.

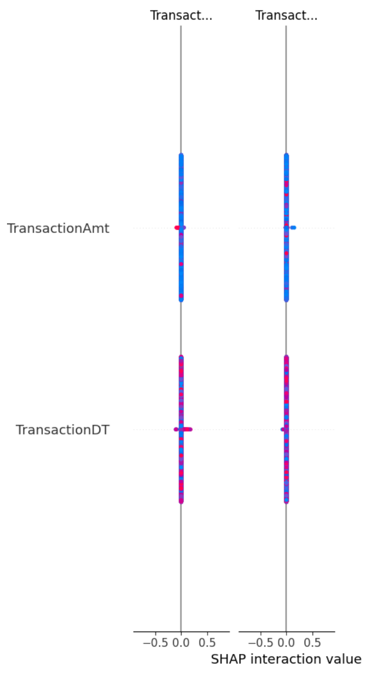


Figure 4.16: SHAP summary plot for the Gaussian NB classifier.

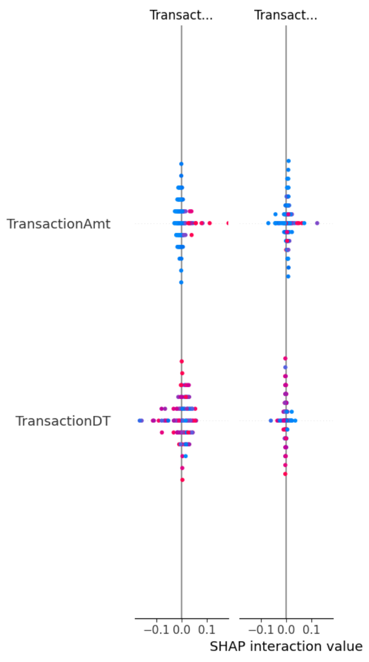


Figure 4.17: SHAP summary plot for the Random Forest classifier.

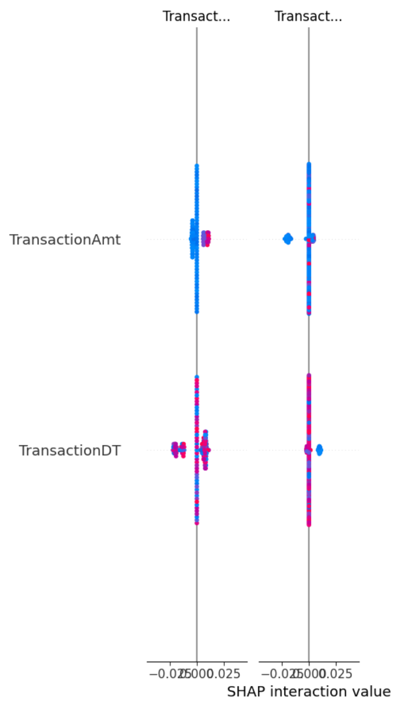


Figure 4.18: SHAP summary plot for the AdaBoost classifier.

4.3.3 Permutation Feature Importance

Permutation importance was also employed to estimate the effect of features on the model performance. By randomly shuffling the values of one feature at a time and measuring the decrease in prediction accuracy, the importance of features is inferred. This approach is model-agnostic and complements both intrinsic feature importance and SHAP analyses.

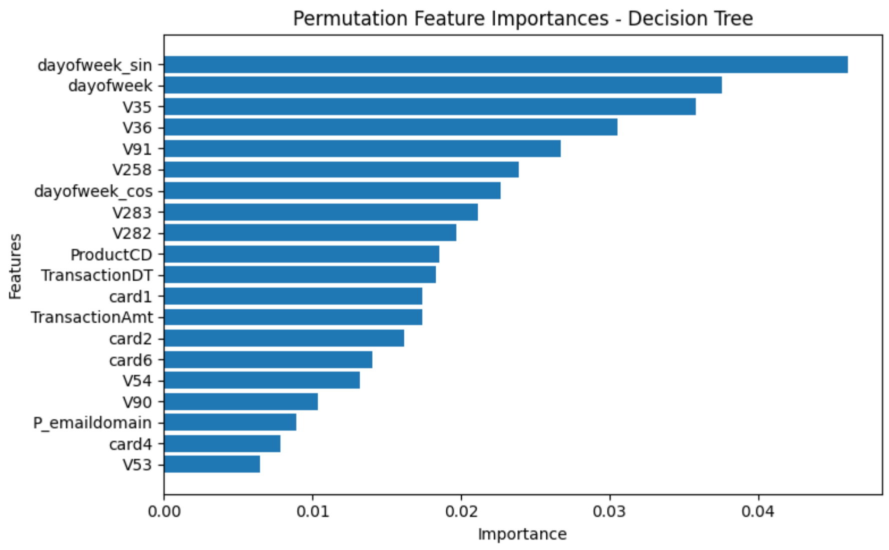


Figure 4.19: Permutation feature importances for the Decision Tree classifier.

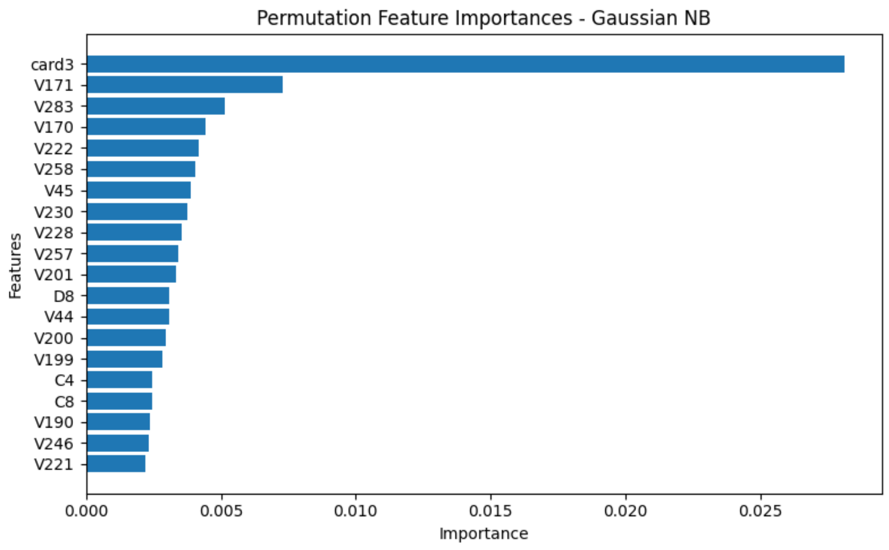


Figure 4.20: Permutation feature importances for the Gaussian Naive Bayes classifier.

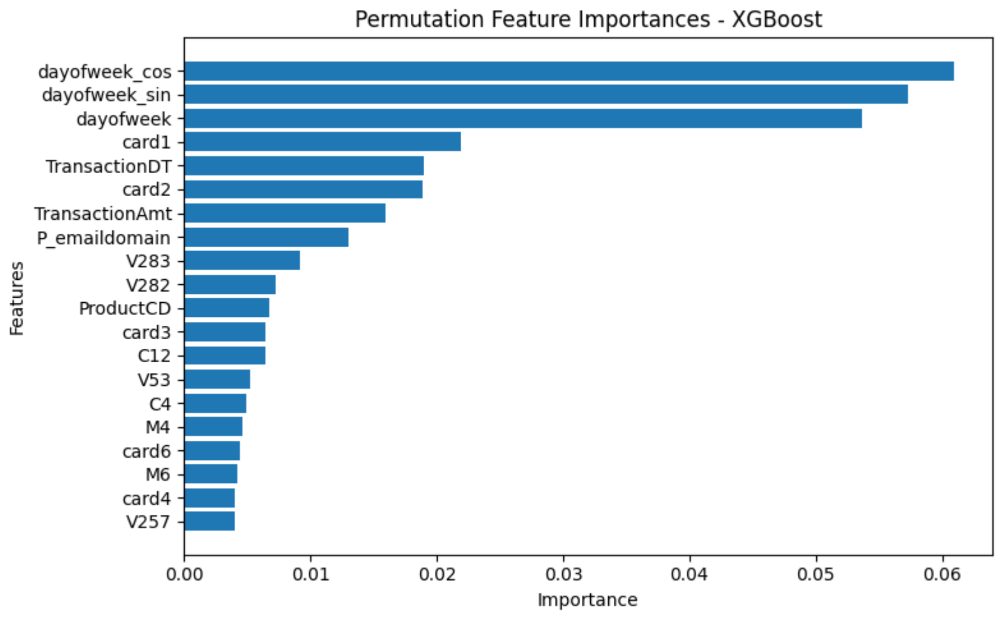


Figure 4.21: Permutation feature importances for the XGBoost classifier.

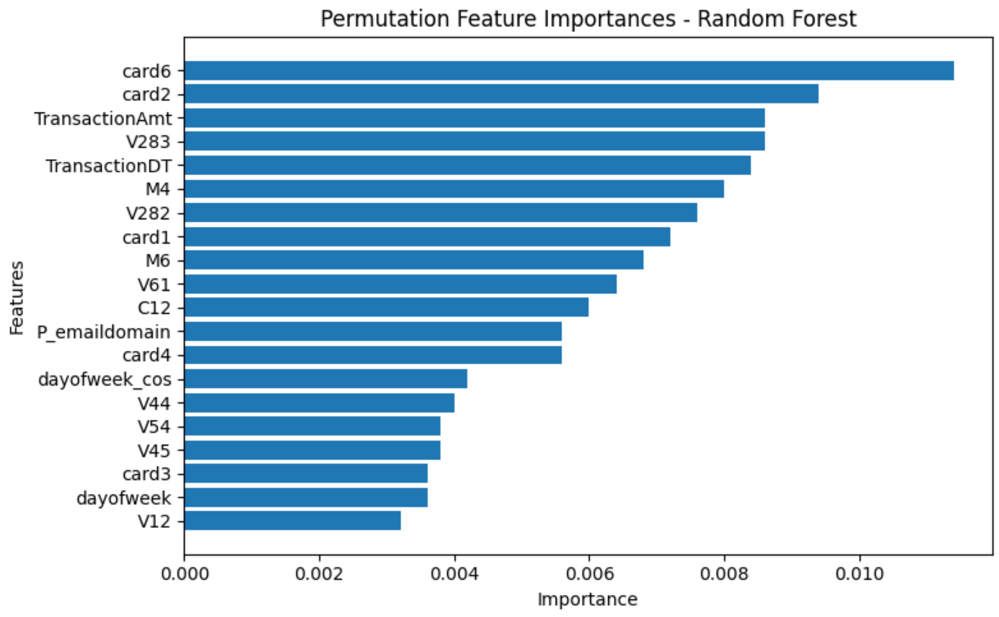


Figure 4.22: Permutation feature importances for the Random Forest classifier.

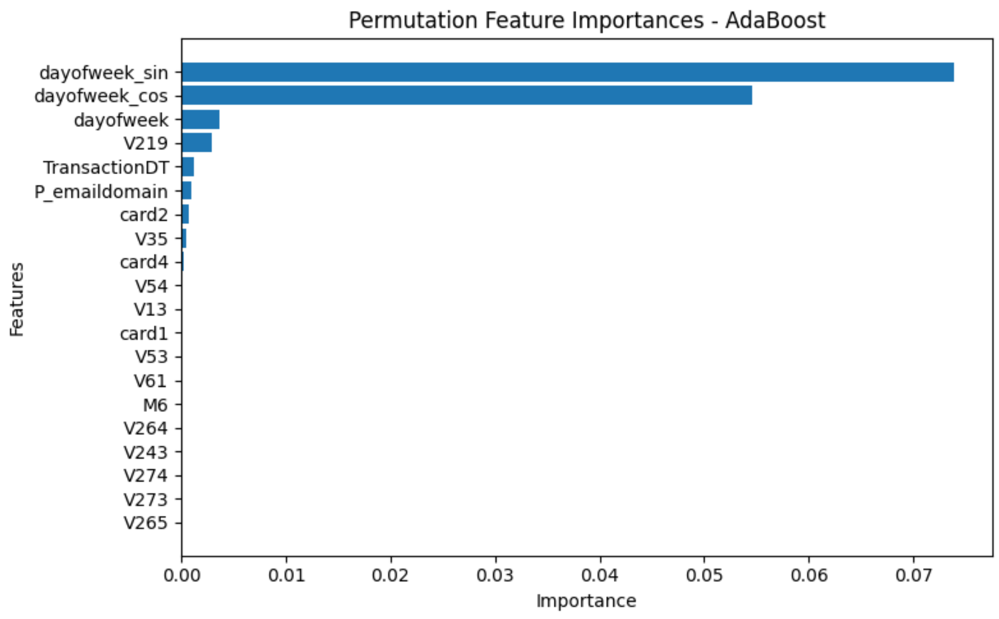


Figure 4.23: Permutation feature importances for the AdaBoost classifier.

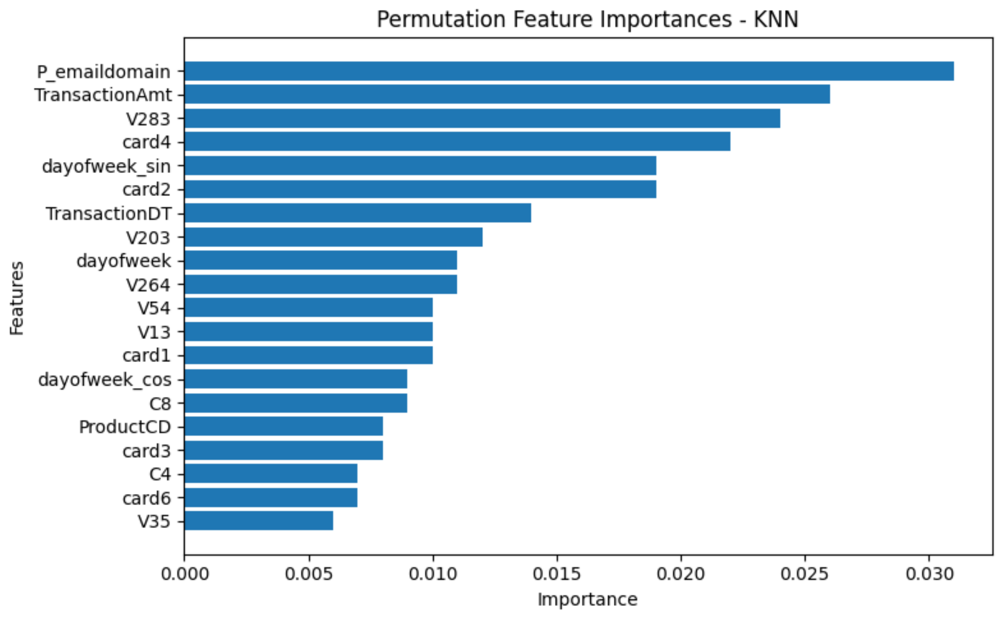


Figure 4.24: Permutation feature importances for the K-NN classifier.

4.3.4 Rule Extraction via Surrogate Models

To further enhance interpretability, surrogate decision trees of depth 3 were trained to approximate the classifiers predictions. These surrogate models allow for extracting simple decision rules that can explain the original complex model behavior in an understandable form. The rules were saved as text files.

```
IF V258 <= 0:
    IF V283 <= 0:
        IF C7 <= 0: class = 0
        ELSE: class = 0
    ELSE:
        IF V283 <= 1: class = 1
        ELSE: class = 0
ELSE:
    IF V258 <= 1:
        class = 1
    ELSE:
        IF V265 <= 5725: class = 1
        ELSE: class = 0
```

Listing 4.1: Example of extracted decision rules from the trained Decision Tree model for fraud detection.

```
IF V258 <= 0:
    IF V283 <= 0:
        IF C7 <= 0: class = 0
        ELSE: class = 0
    ELSE:
        IF V283 <= 1: class = 1
        ELSE: class = 0
ELSE:
    IF V258 <= 1:
        class = 1
    ELSE:
        IF V265 <= 5725: class = 1
        ELSE: class = 0
```

Listing 4.2: Example of extracted decision rules from the trained XGBoost model for fraud detection.

```
IF ProductCD <= -3.01:
    IF D8 <= 737.08:
        IF V76 <= -0.13: class = 1
        ELSE: class = 1
    ELSE:
        IF V170 <= 1.50: class = 0
        ELSE: class = 1
ELSE:
    IF V219 <= 0.00:
        IF V283 <= 5.78: class = 0
        ELSE: class = 1
    ELSE:
        IF V200 <= 0.36: class = 0
        ELSE: class = 1
```

Listing 4.3: Example of extracted decision rules from the trained Gaussian NB model for fraud detection.

```
IF C4 <= 1.03:
    IF V188 <= 0.19:
        IF V54 <= 0.26: class = 0
        ELSE: class = 1
    ELSE: class = 1
ELSE:
    IF V243 <= 0.00:
        IF card2 <= 0.61: class = 1
        ELSE: class = 0
    ELSE: class = 1
```

Listing 4.4: Example of extracted decision rules from the trained KNN model for fraud detection.

```
IF C4 <= 1.03:
    IF card6 <= -0.15:
        IF card6 <= -0.73: class = 0
        ELSE: class = 1
    ELSE:
        IF V171 <= 2.00: class = 0
        ELSE: class = 1
ELSE:
    IF V258 <= 0.00:
        IF V259 <= -0.05: class = 1
        ELSE: class = 0
    ELSE: class = 1
```

Listing 4.5: Example of extracted decision rules from the trained AdaBoost model for fraud detection.

```
IF C4 <= 1.01:
    IF V190 <= 0.19:
        IF card6 <= -0.05: class = 0
        ELSE: class = 0
    ELSE:
        IF dayofweek_sin <= 0.26: class = 1
        ELSE: class = 0
ELSE:
    IF V282 <= -0.80:
        IF C7 <= 6.50: class = 0
        ELSE: class = 1
    ELSE:
        IF P_emaildomain <= -4.14: class = 0
        ELSE: class = 1
```

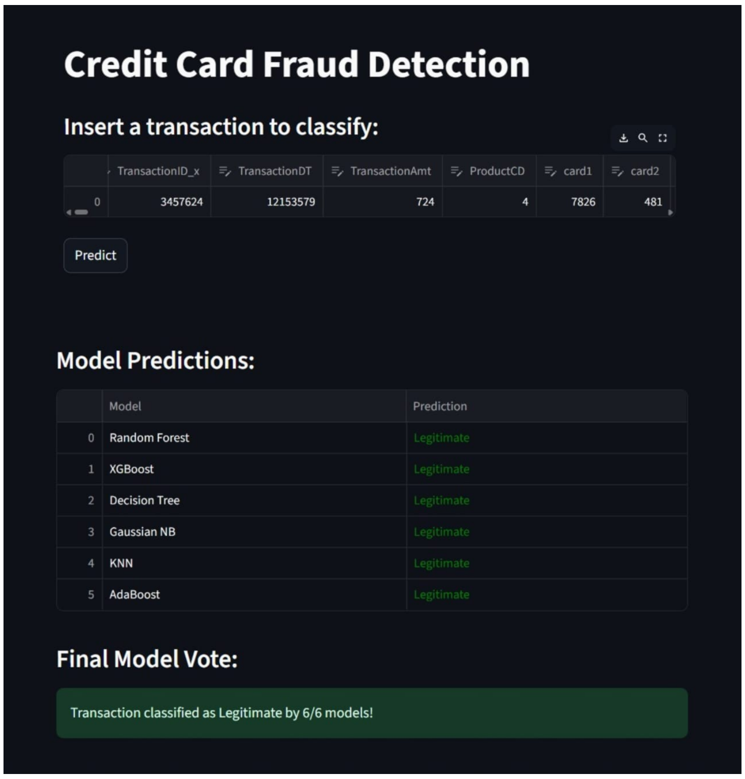
Listing 4.6: Example of extracted decision rules from the trained Random Forest model for fraud detection.

Overall, the combination of these techniques enables both global and local interpretability of the classifiers, making it possible to analyze not only which features are important, but also how they contribute to specific predictions. Figures generated for each method are saved in the `graphs/explanations` folder.

5. Real-world Application

A practical scenario for the application of the models developed in this project is an online payment system, where transactions are continuously performed and the risk of fraudulent activity is high. The aim of the application is to assist financial institutions, e-commerce platforms, or payment processors in detecting potentially fraudulent transactions in real time.

The implemented prototype is a simple web interface realized with `Streamlit`, allowing a user to manually input transaction details.



Credit Card Fraud Detection

Insert a transaction to classify:

	TransactionID_x	TransactionDT	TransactionAmt	ProductCD	card1	card2
0	3457624	12153579	724	4	7826	481

Model Predictions:

	Model	Prediction
0	Random Forest	Legitimate
1	XGBoost	Legitimate
2	Decision Tree	Legitimate
3	Gaussian NB	Legitimate
4	KNN	Legitimate
5	AdaBoost	Legitimate

Final Model Vote:

Transaction classified as Legitimate by 6/6 models!

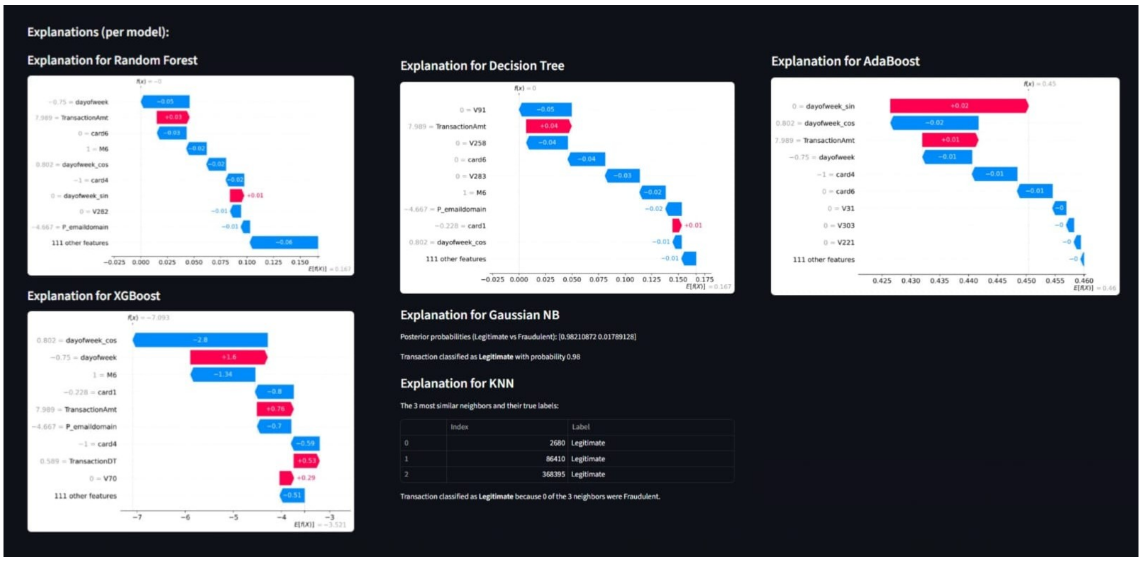
Figure 5.1: Screenshot of the Streamlit interface used for real-time fraud detection. Users can input a transaction and see predictions from all models along with interpretability explanations.

Once the transaction data is submitted, the application performs the following steps.

- **Pre-processing:** The input is transformed using the same pre-processing pipeline applied to the training data. This includes imputation of missing values, scaling, temporal feature engineering (e.g., hour and day of the week), and feature selection based on variance threshold and mutual information.
- **Classification:** Six different classifiers are applied simultaneously: *Decision Tree*, *Random Forest*, *Naive Bayes*, *K-Nearest Neighbors*, *AdaBoost*, and *XG-*

Boost. Each model outputs a prediction indicating whether the transaction is *Legitimate* or *Fraudulent*.

- **Ensemble decision:** A majority voting mechanism aggregates the predictions from all models to produce a final decision.
- **Explainability:** For each model, the application provides explanations of the prediction:
  - Tree-based models (Decision Tree, Random Forest, XGBoost) utilize SHAP values to show the contribution of each feature to the prediction.
  - AdaBoost uses a kernel SHAP explainer with a small background sample from the training set.
  - Naive Bayes displays posterior probabilities for each class.
  - K-Nearest Neighbors shows the most similar examples from the training set and their labels, helping the user understand the decision.


Explanations (per model):
Explanation for Random Forest
Explanation for Decision Tree
Explanation for AdaBoost
Explanation for XGBoost

Explanation for Gaussian NB

Posterior probabilities (Legitimate vs Fraudulent): [0.98210872 0.01789128]

Transaction classified as **Legitimate** with probability 0.98

Explanation for KNN

The 3 most similar neighbors and their true labels:

	Index	Label
0	2680	Legitimate
1	86410	Legitimate
2	368395	Legitimate

Transaction classified as **Legitimate** because 0 of the 3 neighbors were Fraudulent.

Figure 5.2: Screenshot of the application displaying explanations for each classifier. For tree-based models (Decision Tree, Random Forest, XGBoost) SHAP values highlight feature contributions, AdaBoost uses a Kernel SHAP explainer, Naive Bayes shows posterior probabilities, and K-Nearest Neighbors presents the most similar examples from the training set.

This application demonstrates how the developed models can be employed in a real-world context to detect and explain potentially fraudulent transactions, providing both automated classification and interpretable insights to support decision-making.

6. Conclusions

This project aimed to develop and compare a set of classifiers for credit card fraud detection, with a special emphasis not only on performance but also on the interpretability of the models. After an initial preprocessing phase (which included temporal feature engineering, handling of missing values, feature selection, and balancing the training set using the SMOTE technique) it was possible to train six different classification models: *DecisionTree*, *RandomForest*, *Naive Bayes*, *K-Nearest Neighbors*, *AdaBoost*, and *XGBoost*.

- The analysis of the experimental results highlighted the **superiority of ensemble based models**, particularly **XGBoost** and **Random Forest**, with a slight advantage of the first one. These latter models achieved the best performance across most of the evaluation metrics considered, such as *precision*, *recall*, *F1-score*, and *AUC*, proving to be more effective in identifying fraudulent transactions while minimizing false positives.

- The comparison between the classifiers in all the metrics highlighted the **uselessness** of the *weighted* versions of the metrics and of the *accuracy* for credit card fraud detection: the vast majority of the datasets of fraud and non-fraud transactions are strongly imbalanced, with much more non-fraud transactions respect to fraud transactions to represent real world scenarios, in which frauds are a tiny part of the total number of transactions. The aforementioned metrics tend to flatten towards the values of the majority class.

- The introduction of the *Acceptable Level of Performance* (*ALP*) concept, defined as a classifier's ability to correctly identify at least 80% of frauds, allowed for a more in-depth analysis of the trade-off between *TPR* and *FPR*. This analysis revealed that models like **XGBoost** and **Random Forest** can reach this performance level with a **limited number of false alarms**.

- The thresholds analysis showed that KNN and DecisionTree **could not reach the peak performances** as the other classifiers on our dataset. KNN seemed a pretty good model in some metrics, but just could not keep the pace of the best ones. Moreover, it is also the slowest model both to train and to test.

- AdaBoost was the most "conservative" and "safe" model in terms of *ALP_threshold*: it reached *ALP* with the smallest deviation of the threshold from 0.5, but this came along with the biggest *ALP_FPR*, which led it to be one of the worst choice.

- Overall, **XGBoost** turned out to be the **best choice** for a credit card fraud detection application.

- Overall, **Naive Bayes** turned out to be the **worst choice** for a credit card fraud detection application.

- A key aspect of the project was the study of model interpretability (*XAI*). The use of techniques such as intrinsic *feature importance analysis*, *SHAP values*, and *permutation feature importance* helped to find the factors that most influence predictions, increasing the **transparency** and **trustworthiness** of the classifiers. Furthermore, *rule extraction* via surrogate models offered a simplified and understandable view of the behavior of the more complex models.

- Finally, the project culminated in the development of an **interactive Streamlit application** that simulates a real-world use case. This prototype not only classifies transactions in real time but also provides detailed explanations for each model's prediction, offering a practical tool for analysis and decision support. The application integrates the six classifiers and a majority voting mechanism for the final decision, demonstrating how different machine learning techniques can be combined to create a robust, high-performing, and interpretable fraud detection system.

Bibliography

[1] Ieee-cis fraud detection. https://www.kaggle.com/c/ieee-fraud-detection. Accessed: 2025-07-20.