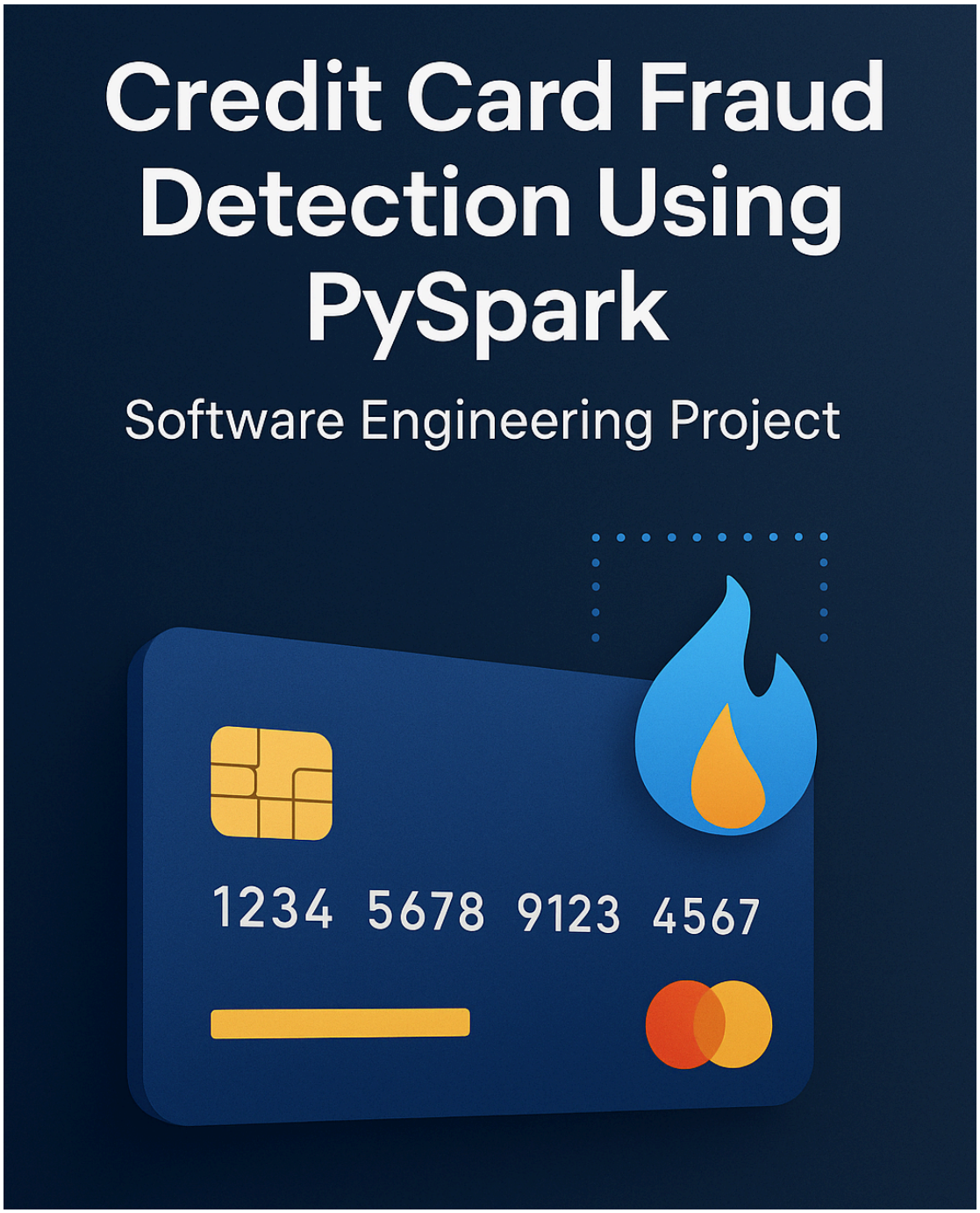
Credit Card Fraud Project (using PySpark)

By Ángel Pérez Castro



Credit card fraud detection is a critical problem in the financial sector, where millions of transactions are processed every day and even a small fraction of fraudulent activity can cause significant economic losses. In this project, I develop a complete data analysis and machine learning pipeline using PySpark to detect fraudulent credit card transactions. The work is based on the well-known Credit Card Fraud Detection dataset from Kaggle, which contains 284,807 transactions and exhibits a highly imbalanced class distribution: only 492 transactions (approximately 0.17%) are labeled as fraud.

The pipeline is implemented using two Jupyter notebooks. The first notebook focuses on exploratory data analysis (EDA) and preprocessing, including schema inspection, descriptive statistics, and data type casting. The second notebook builds and evaluates two classification models using Spark MLlib: Logistic Regression and Random Forest. Both models are trained on standardized features assembled into a single feature vector, and evaluated using Accuracy, F1-score, and ROC–AUC. Despite the strong class imbalance, both models achieve very high performance metrics on the test set, with ROC–AUC values of 0.9569 for Logistic Regression and 0.9504 for Random Forest, and overall accuracy of 0.9992 for both.

The report describes the dataset, the methodology adopted, the modeling process, and the results obtained, and discusses the limitations related to class imbalance and the use of global metrics. Finally, potential improvements and extensions are outlined, such as using class-weighting, resampling techniques, and more advanced evaluation strategies.

1. Introduction

In modern digital economies, credit card payments are one of the most common methods for conducting transactions. While convenient for customers and merchants, this payment method is also an attractive target for fraudulent activity. Detecting fraudulent transactions in real time is a challenging problem due to the large volume of data, the evolving nature of fraudulent strategies, and the very low proportion of fraudulent cases compared to legitimate ones.

Machine learning techniques provide a powerful approach to address this challenge by automatically learning complex patterns from historical transaction data. However, fraud detection systems must be both accurate and computationally efficient, especially when deployed in large-scale environments. This is where PySpark, the Python API for Apache Spark, becomes particularly useful: it allows distributed data processing and scalable machine learning through Spark MLlib.

The objective of this project is to build a complete fraud detection pipeline using PySpark, from data loading and exploratory analysis to model training and evaluation. The work is structured around two notebooks:

- *Notebook 01 – EDA and Preprocessing:*
  Creation of the SparkSession, loading of the Kaggle credit card dataset, inspection of the schema, computation of basic statistics, verification of missing values, and preparation of a clean numeric dataset.

- *Notebook 02 – Modeling and Evaluation:*
  Construction of a feature vector and standardization, train–test split, training of Logistic Regression and Random Forest models using MLlib, and evaluation with several metrics (ROC–AUC, Accuracy, F1-score).

The rest of this report is organized as follows. Section 2 describes the dataset. Section 3 explains the exploratory data analysis and preprocessing steps. Section 4 presents the modeling approach. Section 5 shows the experimental results and evaluation. Section 6 discusses the findings and limitations, and Section 7 concludes the work.

2. Dataset Description

The project uses the Credit Card Fraud Detection dataset, which contains 284,807 anonymized credit card transactions. Each transaction is represented by 31 columns:

30 features:

- Time: the elapsed time (in seconds) between the current transaction and the first transaction in the dataset.
- V1–V28: numerical features obtained via a Principal Component Analysis (PCA) transformation. These components result from a dimensionality reduction and anonymization process to protect customer identities and original feature definitions.
- Amount: the transaction amount.

1 label:

- Class: the target variable indicating whether a transaction is legitimate or fraudulent.
    - Class = 0 → legitimate transaction
    - Class = 1 → fraudulent transaction

According to the output of df.count() and len(df.columns), the dataset contains 284,807 rows and 31 columns. The class distribution is extremely imbalanced:

- Legitimate transactions (Class = 0): 284,315 instances
- Fraudulent transactions (Class = 1): 492 instances

This means that only about 0.17% of the data is labeled as fraud, which is typical for real-world fraud detection tasks and makes the classification problem particularly challenging. Standard evaluation metrics, such as accuracy, can be misleading in this setting, since a trivial model that always predicts the majority class would already achieve very high accuracy.

The descriptive statistics computed for some key variables provide further insight into the dataset. For example, for the variables Time and Amount:

Time (seconds):

- Count: 284,807
- Mean: 94,813.86
- Standard deviation: 47,488.15
- Minimum: 0.0
- Maximum: 172,792.0

Amount (transaction value):

- Count: 284,807
- Mean: 88.35
- Standard deviation: 250.12
- Minimum: 0.0
- Maximum: 25,691.16

These statistics indicate that transaction amounts are highly variable, ranging from very small purchases to relatively large payments. The Time variable spans approximately two days of transactions.

Finally, an important characteristic of this dataset is the absence of missing values. A dedicated null-check using Spark functions confirmed that all columns have zero missing or NaN values. This simplifies the preprocessing stage, since no imputation is required.

3. Exploratory Data Analysis and Preprocessing

The exploratory analysis and preprocessing steps were implemented in the first notebook, using a local Spark session. With a configuration that forces Spark to run in local mode using all available cores, and explicitly binds the driver to 127.0.0.1 to avoid host name issues on Windows.

3.1 Schema Inspection

The dataset was loaded from a CSV file with headers and automatic type inference. The inferred schema, obtained with df.printSchema(), showed that:

- All feature columns (Time, V1–V28, Amount) were recognized as double.
- The Class column was initially read as numeric but was explicitly cast to integer to reinforce its role as a label.

The resulting cleaned schema is:

- Time: double
- V1–V28: double
- Amount: double
- Class: integer

This fully numeric schema is ideal for Spark MLlib algorithms, which expect numeric feature vectors and numeric labels.

3.2 Class Distribution and Imbalance

Using df.groupBy("Class").count().show(), the following counts were obtained:

- Class 0: 284,315 samples
- Class 1: 492 samples

This confirms a heavily imbalanced dataset, where the minority class represents less than one out of every 500 transactions. This imbalance has a direct impact on the choice and interpretation of evaluation metrics, and will be discussed further in the evaluation and discussion sections.

3.3 Descriptive Statistics and Missing Values

The describe() method was applied to selected numerical columns (Time, Amount). As mentioned earlier, the statistics showed no anomalies such as negative amounts and confirmed a wide range of values.

To verify data completeness, the code used a combination of isNull and isnan over all columns. The resulting row of zeros for all columns indicates that no column contains missing or NaN values. Consequently, no explicit imputation or row filtering was necessary; the entire dataset could be used for training and evaluation.

3.4 Type Casting and Clean DataFrame

For consistency and to ensure compatibility with MLlib, the following transformations were applied:

- The Class column was cast to IntegerType.
- All other columns were cast to DoubleType (even though they were already inferred as double, this step guarantees consistency).

The final cleaned DataFrame, df_clean, preserves all 284,807 rows and 31 columns, and is fully numeric. This cleaned DataFrame is used as the basis for the modeling notebook.

4. Modeling Approach

The modeling work is implemented in the second notebook. It reuses the same loading and cleaning logic to construct a clean DataFrame, and then builds a machine learning pipeline using Spark MLlib.

4.1 Feature Engineering

The set of feature columns is defined as all columns except the label:

- feature_cols = [c for c in df_clean.columns if c != "Class"]

Two main feature-processing stages are used:

- VectorAssembler:
  Combines the individual numeric columns into a single feature vector called features_raw.
- StandardScaler:
  Standardizes the features by scaling them to unit standard deviation, producing a new vector column called features. Standardization is particularly important for algorithms like Logistic Regression, which can be sensitive to the scale of the input features.

4.2 Train–Test Split

To evaluate model performance on unseen data, the cleaned dataset is randomly split into training and test subsets:

- Train set: 80% of the data → 228,225 rows
- Test set: 20% of the data → 56,582 rows

A fixed random seed (seed=42) is used to ensure reproducibility of the split.

4.3 Models Considered

Two classification models are trained and compared:

Logistic Regression
A linear model suited for binary classification. The configuration used is:

- featuresCol = "features"
- labelCol = "Class"
- maxIter = 10
- regParam = 0.0
- elasticNetParam = 0.0

Logistic Regression is a natural baseline for this kind of problem, and its probabilistic interpretation is convenient for ranking transactions by fraud risk.

Random Forest Classifier
An ensemble of decision trees that can capture non-linear relationships and interactions between features. The configuration used is:

- featuresCol = "features"
- labelCol = "Class"
- numTrees = 100
- maxDepth = 5
- seed = 42

The relatively shallow depth (maxDepth=5) controls model complexity and helps reduce overfitting, while 100 trees provide a reasonably strong ensemble.

Both models are wrapped into ML pipelines:

- pipeline_lr = Pipeline(stages=[assembler, scaler, lr])
- pipeline_rf = Pipeline(stages=[assembler, scaler, rf])

Each pipeline applies feature assembly, scaling, and then the classifier.

5. Evaluation and Results

5.1 Evaluation Metrics

Given the class imbalance, it is important to look beyond simple accuracy. This project uses:

ROC–AUC (Area Under the Receiver Operating Characteristic Curve):
Measures the ability of the model to rank positive instances higher than negative ones over all possible classification thresholds. Values closer to 1 indicate better performance.

Accuracy:
The proportion of correctly classified instances. On imbalanced datasets, accuracy can be artificially high if the model predicts mostly the majority class.

F1-score:
Harmonic mean of precision and recall. In this project, Spark's MulticlassClassificationEvaluator with metric name "f1" is used, which yields a weighted F1-score across classes.

5.2 Logistic Regression Results

For Logistic Regression, the following metrics were obtained on the test set:

- ROC–AUC: 0.9569
- Accuracy: 0.9992
- F1-score: 0.9991

These values indicate very strong performance. The high ROC–AUC suggests that the model is able to distinguish well between fraudulent and legitimate transactions when considering ranking or threshold-based decisions. The accuracy and F1-score are both above 0.999, which reflects that the vast majority of transactions are classified correctly.

5.3 Random Forest Results

For the Random Forest classifier, the metrics on the test set are:

- ROC–AUC: 0.9504
- Accuracy: 0.9992
- F1-score: 0.9992

Compared to Logistic Regression, the Random Forest model achieves a slightly lower ROC–AUC (0.9504 vs. 0.9569), but a slightly higher F1-score (0.9992 vs. 0.9991). The overall accuracy is identical for both models.

5.4 Comparative Summary

The final comparative table for both models is:

Model	ROC–AUC	Accuracy	F1-score
Logistic Regression	0.9569	0.9992	0.9991
Random Forest	0.9504	0.9992	0.9992

Both models perform extremely well on the test set, with very high ROC–AUC and near-perfect accuracy and F1-score. Logistic Regression appears slightly stronger in terms of ranking (ROC–AUC), whereas Random Forest has a marginally higher F1-score.

6. Discussion

The results obtained are impressive at first sight: both models achieve ROC–AUC above 0.95 and overall accuracy above 0.999. However, these numbers must be interpreted with caution, mainly due to the extreme class imbalance of the dataset.

Because fraudulent transactions represent less than 0.2% of all samples, a model that correctly classifies almost all legitimate transactions but still misses some frauds can still achieve very high accuracy and a high weighted F1-score. Without inspecting class-specific metrics (such as recall for the minority class) or confusion matrices, it is difficult to know exactly how many fraudulent transactions are being detected versus how many are being missed.

Furthermore, no explicit techniques were applied to handle class imbalance, such as:

- Class weighting (giving more importance to the minority class)
- Oversampling of fraud cases
- Undersampling of legitimate cases
- Synthetic sample generation (e.g., SMOTE-type methods)

As a result, the models may be biased toward the majority class. In a real-world fraud detection system, missing frauds (false negatives) is often more costly than occasionally flagging legitimate transactions as suspicious (false positives). Therefore, focusing on metrics such as recall and precision for the fraud class would be essential.

Despite these limitations, the project demonstrates that PySpark and Spark MLlib provide a powerful and flexible environment for building scalable classification pipelines. The use of a pipeline with VectorAssembler and StandardScaler makes it easy to incorporate additional preprocessing steps or replace the classifier with more advanced algorithms.

7. Conclusions

This project implemented a complete fraud detection pipeline using PySpark on the Kaggle Credit Card Fraud Detection dataset. The work included:

- Loading and inspecting a large, highly imbalanced dataset with 284,807 transactions;
- Performing exploratory data analysis and confirming the absence of missing values;
- Preparing a fully numeric dataset with appropriate types for machine learning;
- Constructing a feature engineering and modeling pipeline using Spark MLlib;
- Training and evaluating Logistic Regression and Random Forest models on an 80/20 train–test split.

Both models achieved very strong performance on the test set, with ROC–AUC values above 0.95 and accuracy above 0.999. Logistic Regression achieved a slightly higher ROC–AUC, while Random Forest obtained a slightly better F1-score.

At the same time, the project highlights the importance of carefully handling class imbalance and choosing evaluation metrics that reflect the real goals of fraud detection. Future work could focus on improving detection of the minority class through advanced resampling strategies, class weighting, threshold tuning, or the use of more sophisticated models. Additionally, analyzing confusion matrices and class-specific metrics would provide a clearer picture of how many fraudulent cases are actually being detected.

Overall, this project shows that PySpark is a suitable tool for building scalable fraud detection models and provides a solid foundation for more advanced and production-ready systems.