Joint project of Algorithms for massive datasets & Statistical methods for ML - 2022-23

Simone Malesardi[978989]

Università Degli Studi di Milano

Abstract. This project is built over the "IBM Transactions for Anti Money Laundering" dataset sourced from Kaggle and released under the Community Data License Agreement. The initial request was to create an algorithm that could predict whether a transaction was laundering or legitimate. For this purpose, a learning algorithm for decision trees was developed from scratch and tested on multiple subsets of the data set that can be loaded into main memory. A random forest algorithm was also implemented using the parallel processing environment provided by Apache Spark, with the aim of providing a comprehensive and scalable system for predicting money laundering transactions.
The proposed solution also addresses the imbalance between classes in the data set. This aspect is particularly important since the rate of illicit transactions is extremely low. To ensure the effectiveness of the proposed solutions, the hardware characteristics of the machine on which the entire project was implemented must also be taken into consideration.

1 Introduction

Money laundering is a significant global challenge involving multi-billion dollar transactions that are difficult to detect. Existing automated algorithms often suffer from high false positive rates, flagging legitimate transactions as suspicious and failing to identify those activities that are actually money laundering. These complications are also due to the fact that criminals are able to mask their illicit activities through numerous transactions.

The datasets provided by Kaggle [1] simulate a virtual world inhabited by individuals, businesses and banks. The dataset taken into account is the "HI-Small_Trans.csv" one. Like the other datasets provided by Erik Altman, it is composed of labeled transactions belonging to the laundering class (1) and legitimate one (0).

In this document, the details of the project will be explored discussing the technologies used, the methodology adopted and the results achieved. Through a fusion of techniques and efficient algorithms, the proposed solution aims to address the challenges associated with detecting money laundering activity within big data.

1.1 Project organization

Upon execution of the script, a folder named "datasets" is generated, wherein the entire dataset from Kaggle is downloaded. This directory contains both CSV datasets and TXT files (one per CSV dataset) containing explanations of patterns within each CSV file.

Several checkpoints have been created in the notebook where the spark dataframes are saved in parquet format. These checkpoints were inserted during development so that you can directly load data from a specific point without having to go through the entire processing flow

again. These checkpoints are strategic, placed after phases of intensive computation or data preprocessing.

Those parquet files are stored in the **datasets** folder.

PySpark is a distributed framework of Apache Spark used in the project which allows you to efficiently manage large files by dividing them across multiple nodes and reducing execution times.

Those are the configurations used to build the PySpark Session:

- **Driver Memory**: this configuration ensures a maximum of **3 GB** of memory for Spark driver. The driver is the main process responsible for communication with the cluster.
- **Executor Memory**: it represents the amount of memory allocated to Spark executors, the processes responsible for the computation on the cluster. In this case, a maximum of **4 GB** of memory is allocated to each executor
- **Arrow Library**: the library is enabled to enhance performance during data exchange operations between PySpark and Pandas. Arrow optimizes data serialization and deserialization, thereby significantly improving overall efficiency
- **JARs packages**: the specific JAR packages that need to be downloaded and employed during the application's runtime are defined. The **GraphFrames** [2] package, version 0.8.1, is incorporated. This version is compatible with Spark 3.0 and Scala 2.12. GraphFrames represents an API tailored for graph-related operations within the Spark framework
- **Local Master Node**: the Spark execution mode is configured to run in the "local" mode with a wildcard asterisk **[*]**. This setup allows for the utilization of all available cores on the machine, optimizing the local execution environment

The notebook containing the project is organized in 3 main parts:

- **dataset analysis**: exploration of the dataset in order to understand the data distribution and the correlation between them
- **calculation of features** part includes data sampling and division into train and test set. It calculates additional features and it also contains the selection of features applied to the dataset with a Lasso linear regression model
- **model building and ML evaluation** part contains the sequential implementation of the decision tree learning algorithm with the HalvingGridSearchCV algorithm for setting hyperparameters. A mock-up code for the implementation of the random forest which distributes the creation of trees on the various worker nodes is proposed

With the use of the NumPy library and random seed, all experiments can be easily reproduced.

All experiments were conducted on a MacBook Pro M1 with an 8-core CPU and 16 GB of memory, providing insight into the solution's performance and scalability on a practical hardware setup.

2 Pre-processing

The initial schema of the dataframe is transformed through a custom function in which the new names of the columns and their corresponding data types are defined since the supplied datasets have redundant column names and the type of the columns is string.

The main objective behind data preprocessing is to improve the quality of the data and adapt them to the specific requirements for the entire data mining process. Data cleaning involves identifying inconsistencies in the data such as missing values and duplicates.

The timestamps, relating to the time when each transaction was executed, are converted into a timestamp format that is correct for time analysis (**yyyy/MM/dd HH:mm**). This is important to ensure that time data is consistent and interpreted correctly.

Rows containing null values in any column are found and removed. This ensures that the DataFrame is free of missing data that could compromise subsequent analyses.

Redundant copies of the data are found and removed, focusing only on unique information in the DataFrame, i.e. information that does not have the same information in all columns.

An identifier is then assigned to each unique transaction.

column name	**data type**	**column name**	**data type**
timestamp	timestamp	**receiving_currency**	string
from_bank	integer	**amount_paid**	double
from_account	string	**payment_currency**	string
to_bank	integer	**payment_format**	string
to_account	string	**is_laundering**	integer
amount_received	double		

Table 1. Dataframe schema after pre-processing computation

2.1 Analysis

During the analysis phase, I examined three different subsets of the dataset: the complete set of transactions, the set of laundering transactions, and the set of legitimate transactions. All of these transactions took place in September 2022.

Looking at the trends in sums of money, I noticed that in the laundering set, the amounts received and paid are always the same for each transaction. In the legitimate transaction set, however, the amounts sometimes differ. This discrepancy could be an important indicator for detecting potential laundering activity.

Moving on to transactions over time periods, I discovered that in the laundering set, the percentage of laundering transactions is low in the first 10 days of the month, but it increases significantly from the eleventh day, reaching a money laundering rate of 60%. This increase seems to be accompanied by a decrease in legitimate transactions. This suggests that there might be a direct correlation between transactions and days of the month.

Analyzing the days of the week, I observed that on Saturdays and Sundays, the percentage of laundering transactions is slightly higher compared to other days. This might indicate that money launderers try to exploit days when banking activity could be lower.

As for the hours of the day, it appears that the rate of laundering activities is higher during mid-day hours. This could suggest that money launderers prefer to act during times when they might go unnoticed.

Examining payment formats, I noticed that the number of money laundering transactions with the "ACH" payment format is clearly prevalent over all other types. On the other hand,

in legitimate transactions, there is a greater variety of formats, but the "Cheque" format still prevails.

Concerning receiving and payment currencies, the trends are similar between the two transaction classes. However, a key feature is that laundering transactions always have the same currencies, unlike legitimate transactions, which can vary.

Finally, the last consideration that needs to be made is that the classes are highly unbalanced. The rate of money laundering transactions in the entire dataset is around 0.1%, while the rate of legitimate transactions is around 99.9%.

For further analysis on the patterns (fig. 1) found in laundering transactions, please refer to the specific notebook available here. This deeper dive has helped us gain a more comprehensive view of the strategies and patterns used for laundering transactions.

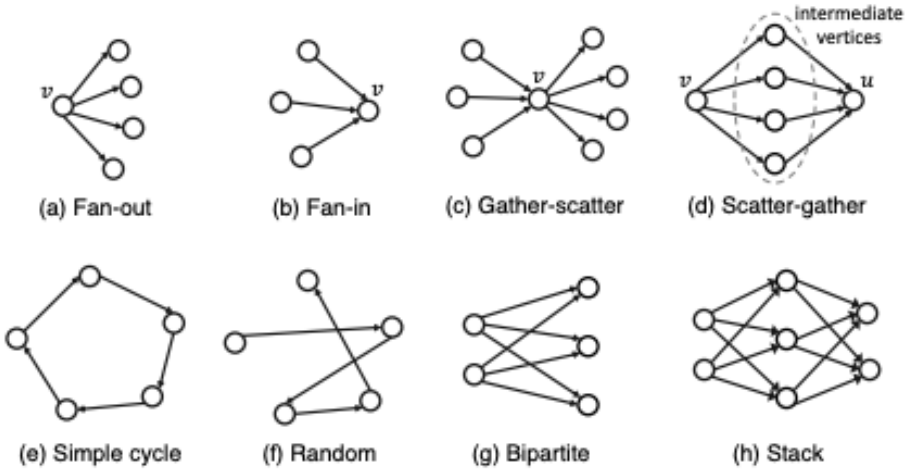

Fig. 1. Types of laundering patterns

In summary, through these analyses, I have identified key trends that help us better understand the behavior of laundering transactions and develop more effective strategies to detect and prevent them.

3 Feature computing

The two classes that handle dataset management and feature creation are FeatureManager and CustomGraph.

FeatureManager implements the code required to calculate independent features, i.e. those concerning the individual transaction.

The features that are calculated on the complete dataset are:

- same_bank
- same_account
- same_currency
- same_amounts
- amount_difference

amount_difference represents the difference between the amount of money paid in a transaction and the amount of money actually received, while the other characteristics take on a truth value, e.g. same_account = 1 if it is a self-payment.

As discussed earlier, the dataset shows significant imbalance between classes.

In datasets characterized by this imbalance, classifiers can often "predict" the most common class, largely ignoring feature analyses, which can result in a high accuracy rate but not necessarily a correct result.

One of the strategies commonly employed to address this imbalance problem is resampling [3]. This technique involves removing samples from the majority class (undersampling) and/or adding additional examples from the minority class (oversampling) (fig. 2).

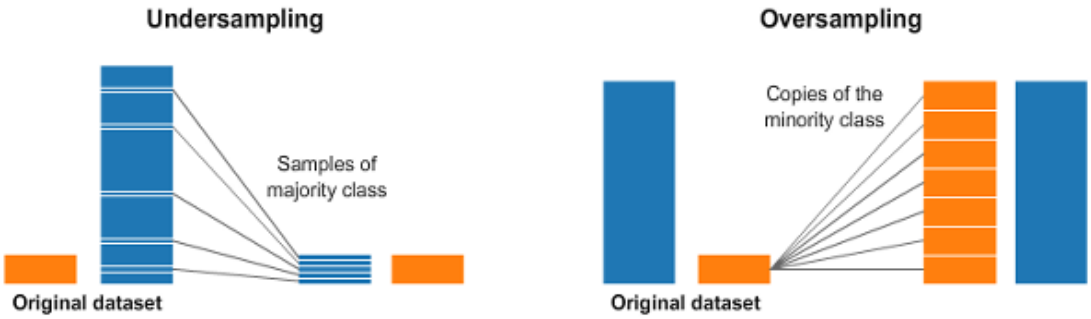

Fig. 2. Class imbalance handling through sampling techniques

The simplest implementation of oversampling is to duplicate random records of the minority class. This can lead to overrepresentation of minority cases in the dataset leading to overfitting of the model. In practice, the model may attribute excessive importance to specific cases in the minority class, compromising the ability to generalize to future data.

On the other hand, subsampling involves creating a training dataset in which samples are removed from the predominant class. This can lead to reduced execution time and improved storage capacity. However, it should be considered that this process could result in the loss of potentially useful transactions that would contribute to classification.

One must also consider that although techniques such as subsampling, but also oversampling, bring advantages, they have their weaknesses (no free lunch).

To address the asymmetry of classes in the dataset, I opted to apply subsampling before splitting between training and test sets. After splitting train and test, new dependent features are calculated. Unlike the previous stage, where the features were independent, in this part of the process the feature values are closely related to the other transactions in the subset.

What is new is the number of transactions made from an account (but also those received) in the same hour, week, day.

The figure 3 shows the correlation between the coefficients of each feature and the label.

The correlation coefficients can vary in a range [-1,1], where the left limit represents a perfect negative correlation and the right limit represents a perfect positive correlation. 0, on the other hand, indicates that there is no correlation We can see that there are no real features

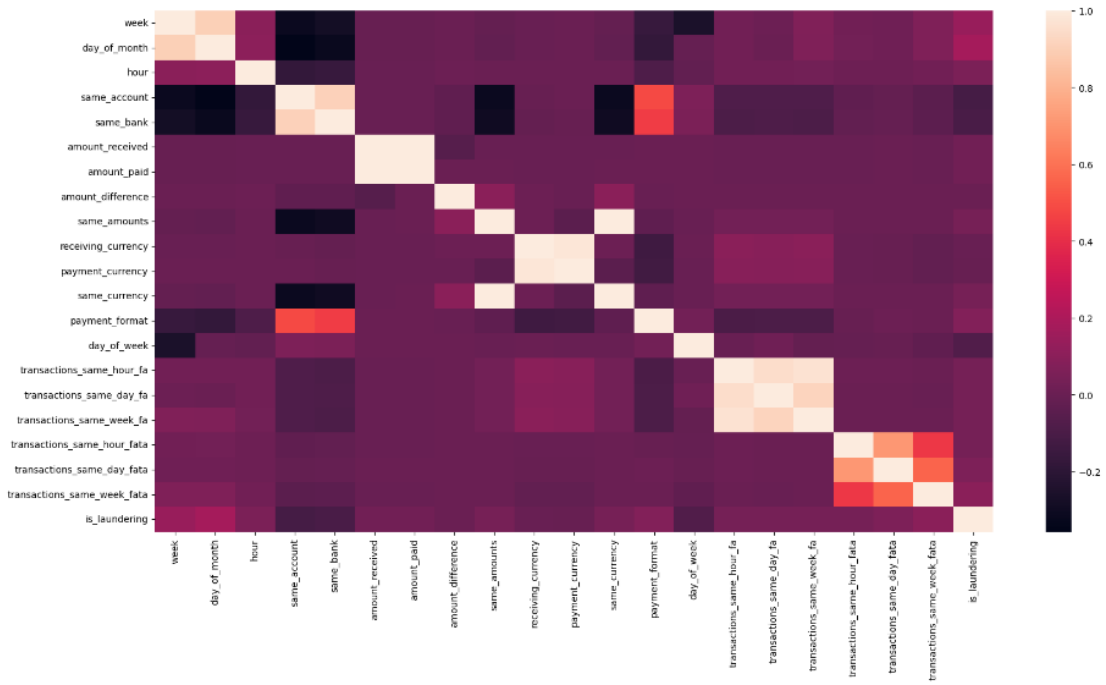

Fig. 3. Correlation matrix of the computed features with FeatureManager

that are closely correlated with the label, but there are features such as amount_difference, receiving_currency and payment_currency that have a correlation close to 0.

3.1 GraphFrames

The dataset represents a virtual world and may not perfectly reflect real-life transactions, but just as in the real world techniques are used to detect recycling patterns, the entire network of transactions can be represented as a real graph, in which the various accounts represent the nodes, while the connecting edges between them represent the real transactions.

An Apache Spark library that allows you to process graphs in a distributed way is GraphFrames [8].

After analyzing the patterns, it would be beneficial to process certain features for each node in the graph to gain valuable insights into the transactions.

- the degrees of connection of the nodes within the graph. Specifically, I need to calculate the number of incoming and outgoing degrees for each node. This allows us to estimate how much money is received from an account and how much money is sent out
- whether a transaction is forwarding. These are transactions in which the recipient sends the same amount of money to another account
- whether there are intermediate transactions between two main transactions. This means that an account could be acting as an intermediary between two transactions, which could be an indicator of nontransparent activity

- PageRank (section 3.1.1) is a key tool for assessing the importance of nodes

Initially, code for managing the pattern cycle was also developed, but due to the too high cost of the algorithm, this part was no longer considered.

The figure 4 shows the correlation matrix representation of the dataset to which the new features calculated with graphframes were added.

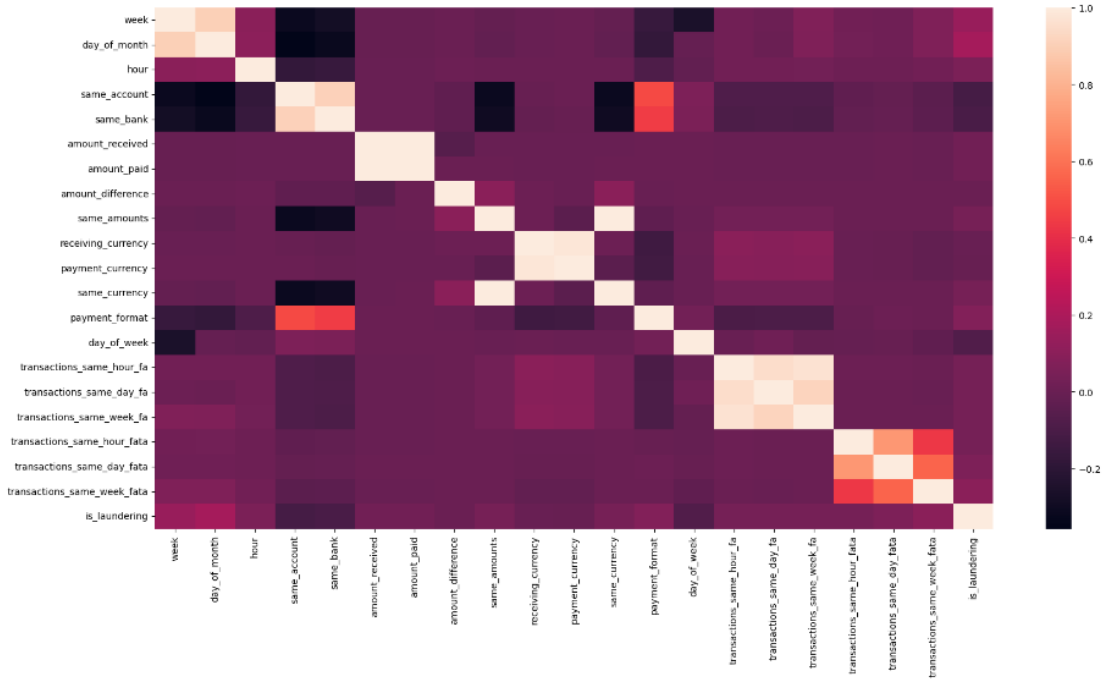

Fig. 4. Correlation matrix of the computed features with GraphFrame

3.1.1 PageRank

It measures the importance of each account based on the number of incoming relationships and the importance of the corresponding source nodes.

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$$

where:

- $PR(p_i)$ is the p_i PageRank
- $M(p_i)$ is the set of nodes that link to p_i
- $L(p_j)$ is the number of outbound links on node v
- N is the total number of nodes
- d is the damping factor (resetProbability)

It is invoked as follows:

```
pageRank(resetProbability=0.15, tol=0.1)
```

where:

- **resetProbability**: is the probability of reset to a random vertex during PageRank calculation. In other words, it represents the probability that the user stops browsing on the current page and starts from a random page.
- **tol**: the PageRank algorithm is run until the tolerance specified in tol has been reached. The tolerance represents the maximum allowable difference between the PageRank scores of two successive iterations. This is useful if you wish to run the algorithm until PageRank converges to a specified precision.

The method returns a new graph with two new columns: "pagerank" for vertices and "weight" for edges.

3.2 LASSO: linear regression with regularization L1

Feature selection is a crucial step when developing a predictive model. It consists of reducing the number of input variables to improve model performance. This is critical because irrelevant or redundant features can compromise the effectiveness of the model, causing errors, reducing accuracy and increasing computational costs.

A very useful technique for feature selection is logistic regression with L1 regularization, also called LASSO regression. This method tries to strike a balance between the simplicity of the model and its accuracy.

It is ideal when you have many features, since it automates the selection of the most important variables.

Here is how it works in a simplified way [4]:

- Linear regression model: I start with a standard linear regression model that assumes a linear relationship between the independent variables (features) and the dependent variable (target)
- L1 adjustment: LASSO regression adds a penalty based on the absolute values of the model coefficients. This penalty depends on a parameter λ
- Objective function: the goal of LASSO is to find the values of the coefficients that minimize the sum of the squared differences between the predicted and actual values, while simultaneously minimizing the L1 penalty
- Reduction of coefficients: through the L1 penalty, LASSO can reduce some coefficients to zero. When λ is sufficiently large, some coefficients are brought to exactly 0, which means the corresponding features are removed from the model
- Tuning parameter λ: the choice of parameter λ is crucial.
  - $\lambda \to 0$: a smaller λ limits the penalty and keeps more features
  - $\lambda \to \infty$: a larger λ increases the penalty and reduces the number of features in the model
- Model fitting: an optimization algorithm, such as Coordinate Descent, is used to estimate the coefficients in the LASSO

4 ML models

The first part of this section presents an overview of the Decision Tree implemented that is subsequently tested on a dataset that fits in memory.

In addition, the model is tested using the breast cancer dataset provided by sklearn.

Subsequently, an algorithm for selection of decision tree hyperparameters is used in such a way as to modify the performance of the model in order to obtain optimal results.

Finally there is the development of a Random Forest class that uses Spark to distribute the creation of trees on the different worker nodes.

4.1 Decision Tree

For the design and implementation of the decision tree algorithm, extensive research was conducted, referring to a variety of sources, including academic publications, analysis of existing implementations, and open source source code [5]. This material enabled the acquisition of a solid understanding of best practices and key techniques in the implementation of decision trees.

The process of creating the tree can be summarized in the following steps:

1. starts with a root node that contains the entire training set
2. finds the best feature in the dataset using a feature selection criterion, such as entropy or Gini index, to find the best feature with which to partition the current node
3. once the best feature is found, it subdivides the node into subsets based on the possible values of the feature
4. creates a decision node containing the best feature
5. for each child node created repeats these steps recursively, starting from step 2. This process continues until a stopping condition is met, such as the maximum depth of the tree or the minimum number of samples in a node to continue the subdivision, or all the sample belongs to the same class
6. at the end, when it is no longer possible to subdivide nodes or when all nodes in a subset have the same target value, the leaves of the tree are reached

The CustomDecisionTree class has a number of essential methods that are executed in a specific order, as follows:

- grow_tree: recursive method that takes as input the training set and the label set, returning as output the entire decision tree. The tree construction process begins with a root node that initially contains the entire input set. For each node under consideration, the following conditions are checked in order to determine whether the node should become a leaf of the tree:
  - are there only unique target classes in the data?
  - has the maximum depth of the tree (max_depth) been reached?
  - is the number of samples less than the minimum value required to continue splitting (min_samples_split)?

  If one of these conditions is met, the node becomes leaf and will contain the most common class among the considered data. Otherwise, if none of the conditions is met, the method searches for the best feature and split threshold by calling best_split, and then splits the node into two child nodes

- best_split: it takes as input the dataset and its labels. Initially, it selects a random number of features to examine, which helps to prevent overfitting, and then it iterates over the selected features to calculate the feature index and split threshold that will allow the parent node to be split into two subsets in order to maximize the information gain (information_gain)
- information_gain: it is a measure used to evaluate the importance of a feature in splitting the data. Higher information gain indicates that the feature is more informative and improves class separation in the data. The gini index and entropy are the criteria used to calculate the information gain [6]. Specifically, the information gain is calculated as the entropy of the parent node minus the sum of the weighted entropies of the child nodes. The weight of a child node is given by the ratio of the number of samples in the child node to the total number of samples among all child nodes. Similarly information gain is calculated with gini score

The results obtained on the model were promising. The following are the main evaluation metrics:

- **Accuracy**: 0.91
- **Precision**: 0.57
- **Recall**: 0.66
- **F1 score**: 0.61

4.1.1 Performance of CustomDecisionTree

Overall, these results suggest that our Decision Tree model is effective in detecting suspicious money laundering transactions, with high accuracy and a good combination of recall and precision. However, further optimization or exploring other models could be considered to further improve performance.

These are the parameters used in the model:

```
{'max_n_features': 'sqrt',
'split_criterion': 'gini',
'max_depth': 30,
'min_samples_split': 2,
'max_thresholds': None}
```

In order to demonstrate the effectiveness of the CustomDecisionTree algorithm, training and testing was conducted using the "breast_cancer" dataset provided by Scikit-Learn. This dataset, similarly to the case of recycling transactions, is a binary dataset.

The model based on CustomDecisionTree obtained an F1 score of 96%.

In addition, to evaluate the performance of the CustomDecisionTree model, a Scikit-Learn DecisionTreeClassifier model was trained and tested with the same data. This comparison showed how well the CustomDecisionTree model performs compared to the standard Scikit-Learn DecisionTreeClassifier model.

4.2 Hyperparameter Tuning

GridSearchCV and RandomizedSearchCV are common hyperparameter search techniques used to optimize machine learning models. However, these techniques were not suitable for this case study:

- **GridSearchCV**: it is a very exhaustive technique as it considers all combinations of hyperparameters, but it can quickly become inefficient with a large number of hyperparameters to test, leading to a high computational load and a large execution time
- **RandomizedSearchCV**: unlike the first method, it does not consider all possible combinations, but it performs the random search on a random subset of the possible combinations. It's faster than GridSearchCV, but it may not fully cover the hyperparameter space and you may miss important combinations

A different approach, but still based on cross validation is the **HalvingGridSearchCV** [7] experimental method provided by sklearn. This approach addresses the limitations of GridSearchCV and RandomizedSearchCV, providing the following advantages:

- Efficient use of resources: while GridSearchCV examines all combinations of hyperparameters, HalvingGridSearchCV intelligently narrows down the search space by iteratively selecting and refining the most promising combinations. This leads to a reduced computational load and shorter execution times
- Scalability: as the dataset grows, HalvingGridSearchCV remains efficient due to its focus on subsets of data and iterative refinement. This ensures that the method scales well even with large datasets
- Enhanced Generalization: by evaluating performance on multiple subsets of data, HalvingGridSearchCV aims to find hyperparameters that perform well across various scenarios, contributing to better model generalization

HalvingGridSearchCV is designed to be efficient when you have a large number of hyperparameters to optimize and it works like this:

1. Divide the training data into smaller subsets
2. Start by testing combinations of hyperparameters on a small number of subsets
3. Maintains only the best combinations of hyperparameters based on average performance on subsets of data
4. Repeat the previous steps with an increasing number of subsets and a decreasing number of hyperparameter combinations

In conclusion, this algorithm simplifies the hyperparameter tuning process, making it fast and efficient. Through this optimization, the predictive accuracy and generalization capabilities of the decision tree model are improved.

By giving the following parameters as input to HalvingGridSearchCV

```
{'max_n_features': ["sqrt", "log2", None],
'split_criterion': ['entropy','gini'],
'max_depth': [5, 10, 20, 30, 40, None],
'min_samples_split': [2, 5, 10],
'max_thresholds': [2, 5, 10, 15, None]}
```

the model selects the following parameters:

```
{'max_depth': 10, 'max_n_features': None,
'max_thresholds': None, 'min_samples_split': 10,
'split_criterion': 'gini'}
```

4.3 Random Forest

The last step concerns the implementation of the random forest which allows the distribution of the work on different worker nodes. The worker nodes that are used are directly proportional to the number of processor cores on which the program is running.

The CustomRandomForest class represents a custom implementation of the Random Forest algorithm, designed to support distributed computing using Spark RDD.

These are the parameters used in the model [11]:

```
{'n_estimators': 40, 'split_criterion': 'gini',
'max_depth': 20, 'max_n_features': 'sqrt'}
```

4.3.1 Train the classifier

This implementation offers several customization options for the user, but one of the most important attributes is bootstrap [10], which determines whether random sampling with replacement is performed, on the training set, during the creation of each tree. This means that to create each tree, a set of training data is randomly selected from the data population with the possibility that some data may be selected multiple times and others may be omitted.

The data are distributed within the RDD into a specified number of partitions, indicated by n_estimators. For each data partition, a function named train_tree is applied using the mapPartitions method. The train_tree function returns a CustomDecisionTree object, which represents the trained decision tree.

The generated trees are obtained when the collect() method is called. This method allows iterating over each partition of the data, processing it independently to train a decision tree and collecting all the estimators (decision trees) generated by each partition. This list of decision trees represents the final result of the training.

4.3.2 Test the classifier

The test set is broadcast to all worker nodes using PySpark's broadcast mechanism.

Each decision tree trained in the previous step is parallelized on the worker nodes. The nodes perform the predictions using the local test set.

The intermediate prediction results from each worker node are collected and aggregated to the driver node. A majority voting mechanism is used to obtain the final predictions.

The results obtained on the model were promising. The following are the main evaluation metrics:

- **Accuracy**: 0.92
- **Recall**: 0.58
- **Precision**: 0.59
- **F1 score**: 0.58

Finally, the classifier, is compared with pyspark's RandomForestClassifier, achieving very similar performance.

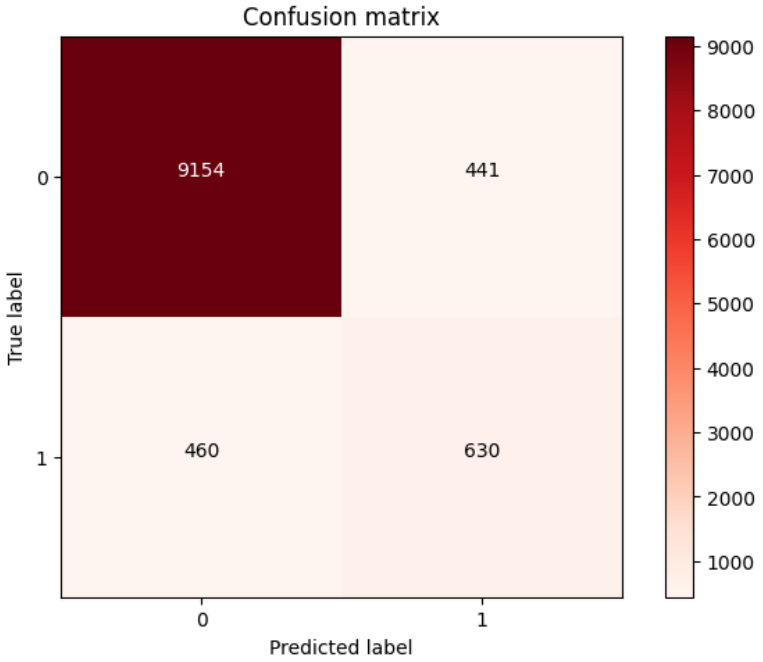

Fig. 5. Confusion matrix of the Random Forest

5 Final consideration

The use of undersampling mitigated class imbalance by providing a more balanced training base and reducing the dimensionality of the data. This allowed the dataset to fit in memory, optimizing the efficiency of the training process.

The use of Apache Spark and, in particular, GraphFrames, enabled efficient processing of the data. Feature analysis, including the application of algorithms such as PageRank, made it possible to extract relevant information from connections between transactions, thus enriching the dataset with network information crucial for laundering detection.

L1 Lasso optimized model efficiency and reduced computational complexity by selecting the most informative variables, thus enabling faster and more cost-effective analysis.

Tuning the decision tree hyperparameters with HalvingGridSearchCV improved the ability to generalize to test data.

Finally, the distributed implementation of Random Forest with PySpark emphasized the scalable approach of the project. The ability to handle large datasets and the diversification of patterns through bootstrap sampling were key factors in achieving results with some quality.

In summary, the project is a demonstration of how the integration of these techniques enables the development of a scalable and efficient money laundering detection system capable of handling big data in main memory.

6 Declaration

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

References

1. Erik Altman. *IBM Transactions for Anti Money Laundering (AML)* (2023). link
2. Graphframes. *GraphFrames JAR.* link
3. Analytics Vidhya. *The problem with class imbalance in machine learning* (2020). link
4. Great Learning Team. *A Complete understanding of LASSO Regression* (2023). link
5. Mısra Turp. *Decision Tree Algorithm* (2022). link
6. thatascience. *Gini index vs Entropy* (2020). link
7. EDUCBA. *Decision Tree Hyperparameters* (2023). link
8. Xiangrui Meng. *Graphframes package* (2020). link
9. Wikipedia. *PageRank* (2001). link
10. Celso M. Cantanhede Silva. *So you want to bootstrap with PySpark* (2020). link
11. Saurabh Gupta. *Hyperparameters of Random Forest Classifier* (2021). link