DECISION TREE AND RANDOM FOREST CLASSIFIERS

Matias Salaris Giuseppe Tallini

Settembre 2023

Contents

1 Introduction

The analysis and recognition of money laundering and illicit transactions are very common problems in today's financial landscape. Banks, governments, and private entities are constantly researching new methods and improving their ability to find a solution.

Our project centers around the analysis of a comprehensive financial transaction database that accurately reflects real-world transactions. Our primary goal was to develop a robust classification model capable of identifying money laundering transactions within the database by exploiting the capabilities of decision tree classifiers and random forest algorithms.

We approached this project by dividing the job into various tasks. First, we analyzed the dataset, in order to gain a general idea of the patterns and characteristics of the transactions. In this phase we realized the main problems to solve would have been: (i) transforming the categorical variables into numerical variables and (ii) dealing with a highly unbalanced dataset.

We then moved to the phase of pre-processing and feature engineering, where we started modeling the database into a more suitable configuration. This process involved considering various factors, such as (i) how to handle outliers, (ii) choosing suitable categorical variables encodings, (iii) choosing the variables to drop and (iv) choosing the proportion on positive and negative examples to use at training time.

After finishing the feature engineering part, we began writing and testing our decision tree classifier, choosing though cross validation the best hyperparameters.

The decision tree classifier was used as a building block for a random forest. Every tree in the forest was trained using a different subset of the original dataset. By parallelizing the construction of the single trees, we were able to train the whole forest in one third of the time.

Finally, we tested our model against a baseline composed of a random classifier, showing significant improvements. In order to quantify these improvements we built a cost model for the money laundering problem. According to this model, our forest reduces the bank cost to 1/41 with respect to not identifying illicit transactions and to 1/50 with respect to the random classifier.

2 Dataset Description

The "IBM Transactions for Anti Money Laundering" dataset is a collection of databases containing more than 300M financial transactions created through a generator based on real-world data and observations.

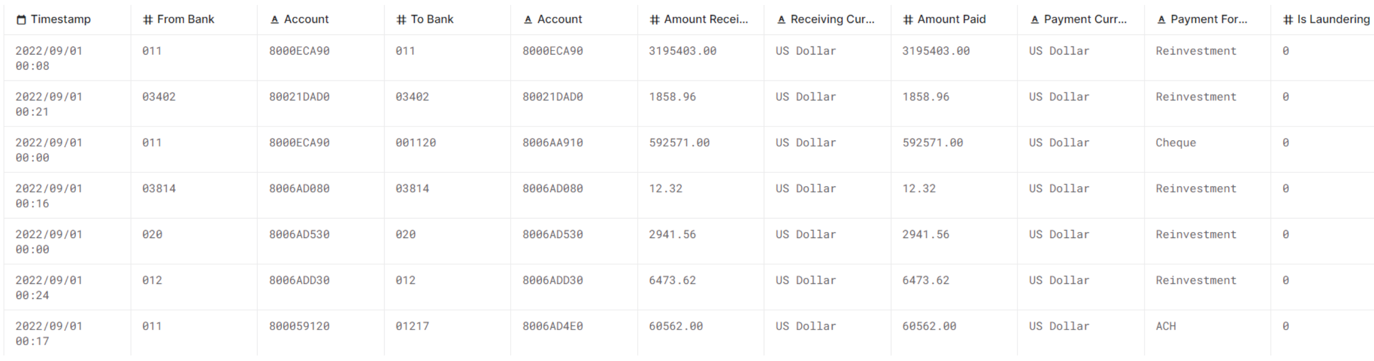
The dataset contains 6 databases classified by their dimension (Large, Medium, Small) and the relative ratio of money laundering transactions (High, low).

To be compliant with our time possibilities we chose to work on a small database ("LI-Small") containing around 5M transactions. During our development, we found it to be useful to force a pre-determined ratio of the labels in the training set, for this reason using either one or the other dataset didn't affect our results. Our final result works efficiently on the database we used and can scale well even on larger databases (or on databases with a different label ratio). Each database is

composed of 11 columns:

1. **Timestamp:** Year/Month/Day Hour/Minute
2. **From Bank:** Numeric code for bank where transaction originates
3. **Account Sender:** Hexadecimal code for account where transaction originates
4. **To bank:** Numeric code for bank where transaction ends
5. **Account Receiver:** Hexadecimal code for an account where the transaction ends

6. **Amount Received:** Monetary amount received (in currency units of the next column)
7. **Receiving Currency:** Currency such as dollars, euros, etc of From account
8. **Amount Paid:** Monetary amount paid (in currency units of the next column)
9. **Payment Currency:** Currency such as dollars, euros, etc of From account
10. **Payment Format:** How transaction was conducted, e.g. cheque, ACH, wire, credit cards, etc.
11. **Is Laundering:** 0/1 value with 1 = Transaction is Laundering, 0 = Not



Timestamp	From Bank	Account	To Bank	Account	Amount Recei...	Receiving Cur...	Amount Paid	Payment Curr...	Payment For...	Is Laundering
2022/09/01 00:08	011	8000ECA90	011	8000ECA90	3195403.00	US Dollar	3195403.00	US Dollar	Reinvestment	0
2022/09/01 00:21	03402	80021DAD0	03402	80021DAD0	1858.96	US Dollar	1858.96	US Dollar	Reinvestment	0
2022/09/01 00:00	011	8000ECA90	001120	8006AA910	592571.00	US Dollar	592571.00	US Dollar	Cheque	0
2022/09/01 00:16	03814	8006AD080	03814	8006AD080	12.32	US Dollar	12.32	US Dollar	Reinvestment	0
2022/09/01 00:00	020	8006AD530	020	8006AD530	2941.56	US Dollar	2941.56	US Dollar	Reinvestment	0
2022/09/01 00:24	012	8006ADD30	012	8006ADD30	6473.62	US Dollar	6473.62	US Dollar	Reinvestment	0
2022/09/01 00:17	011	800059120	01217	8006AD4E0	60562.00	US Dollar	60562.00	US Dollar	ACH	0

Figure 1: First rows of the LI-Small table

3 Data Exploration

3.1 Null Value Analysis

We started by checking for null values within the database. Fortunately, we found no instances of missing data, as there were 0 null values across all features.

3.2 Outlier Detection

To identify potential outliers relative to the amount of money transferred, we employed the **k-means** distribution method. This technique allowed us to analyze the distribution of data points and identify any observations that significantly deviated from the expected patterns. We then analyze the resulting outliers whose distance from the mean of the general distribution was higher than a certain threshold to check if they had a significant relevance, but they did not.

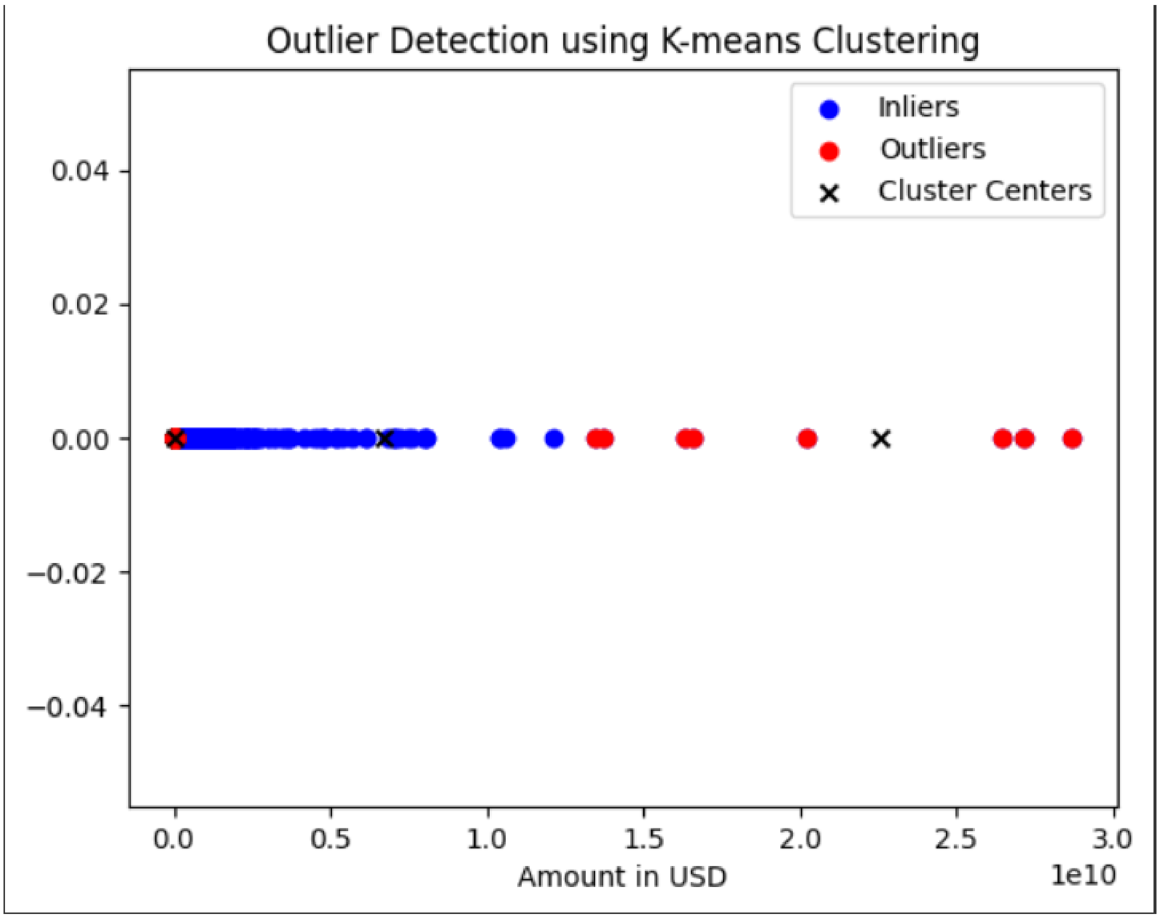


Figure 2: Detected Outliers

3.3 Density Distributions

We also examined the density distributions among several features within the database. By visualizing the distributions, we gained insights into the spread and concentration of values for these features. The distribution of transactions per account reveals significant variations, indicating an unbalanced pattern. Some accounts exhibit a high number of transactions, while others have a comparatively lower count. This uneven distribution suggests that certain accounts play a more prominent role in shaping the overall outcome. For example, figure 3 shows the first ten accounts per number of transactions. The same applies to the number of illicit transactions made from the same account, as shown in figure 4, this characteristic also appears to be unevenly distributed among accounts.

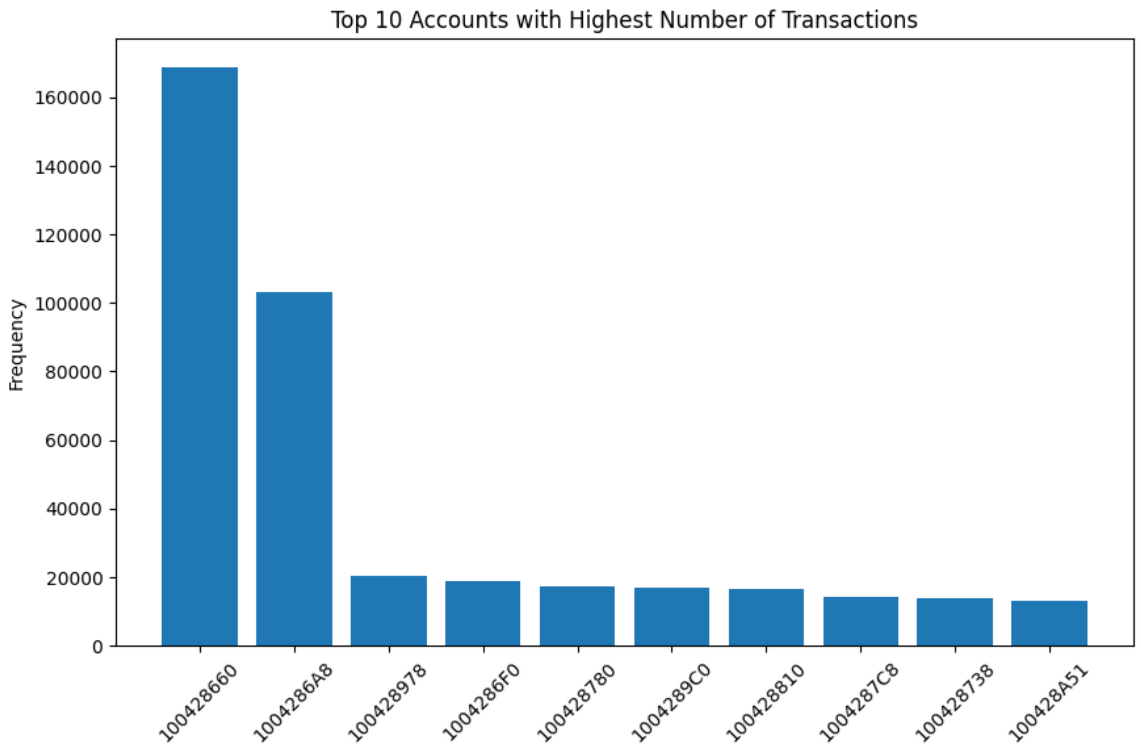


Figure 3: Number Of Transactions per Account

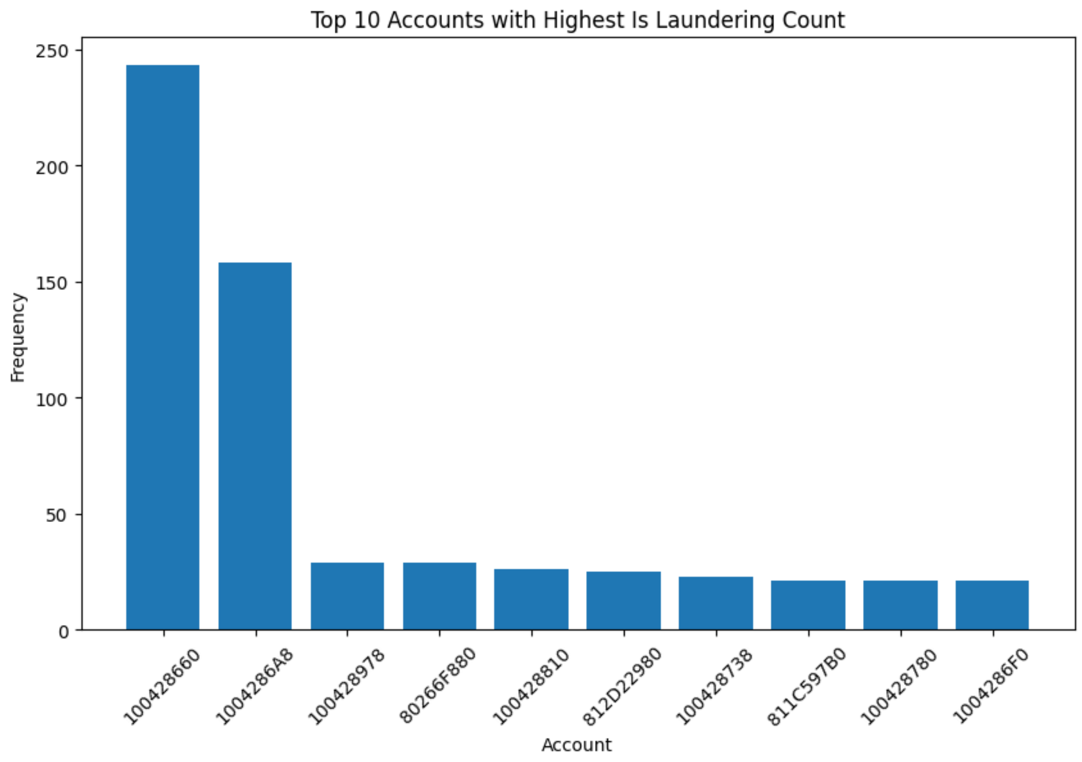


Figure 4: Recidivist Accounts

3.4 Pattern Analysis

During the exploration process, we discovered a noteworthy pattern related to transactions involving four particular features: (From Bank, To Bank, Account sender, Account Receiver). One particular observation that allowed us to identify a correlation between attribute values and the laundering positivity of the transaction is the recidivism of the malicious party. In other words, we observed that if an illicit transaction came from a given number of accounts or from a given bank, the probability of finding other illicit transactions for that same account or for the same bank increased. On the other hand, we observed a lack of correlation between the values of other features and the label, e.g., the ”Timestamp” attribute describing the date and time of the transaction appears to be non-influential for classification purposes

Overall, through our data exploration process, we gained a comprehensive understanding of the database, ensuring its reliability and enabling us to make informed decisions for subsequent feature engineering and modeling tasks.

4 Preprocessing

4.1 Outliers Removal

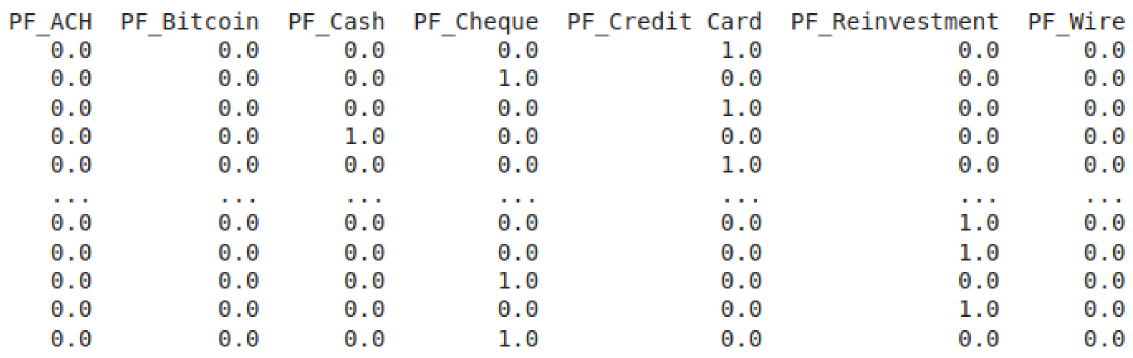
As mentioned in the data exploration section, we detected outliers using the k-means distribution method. To ensure the integrity of our analysis, we removed these outliers from the dataset. By eliminating these extreme values, we aimed to reduce the execution time of our classifier algorithm and most of all the variance in the attribute indicating the amount of money in a transaction.

4.2 Normalization and Columns Removal

Next, we focused on normalizing the "amount paid" column to USD dollars. This normalization allowed us to bring all payment amounts to a common currency for better comparability and analysis. Additionally, we decided to remove the columns related to "amount received" and "receiving currency," as we recognized that they were almost identical to the respective features "Amount Paid" and "Payment currency", and the only times there was a discrepancy between the 2, it didn't influence the outcome of the transaction. However, we retained a general column named "Currency" to capture the currency used in each transaction, as it could provide valuable information.

4.3 Features Engineering

For the categorical features "Paying method" and "Currency", which only consisted of a small pool of unique values, we employed one-hot encoding (figure 5). This encoding technique created binary columns for each unique category, allowing us to represent the categorical variables as binary vectors. This approach prevents any ordinal relationship assumption and allows the decision tree to handle a numerical representation of non-numerical attributes.



```
PF_ACH  PF_Bitcoin  PF_Cash  PF_Cheque  PF_Credit Card  PF_Reinvestment  PF_Wire
   0.0         0.0      0.0        0.0             1.0              0.0      0.0
   0.0         0.0      0.0        1.0             0.0              0.0      0.0
   0.0         0.0      0.0        0.0             1.0              0.0      0.0
   0.0         0.0      1.0        0.0             0.0              0.0      0.0
   0.0         0.0      0.0        0.0             1.0              0.0      0.0
   ...         ...      ...        ...             ...              ...      ...
   0.0         0.0      0.0        0.0             0.0              1.0      0.0
   0.0         0.0      0.0        0.0             0.0              1.0      0.0
   0.0         0.0      0.0        1.0             0.0              0.0      0.0
   0.0         0.0      0.0        0.0             0.0              1.0      0.0
   0.0         0.0      0.0        1.0             0.0              0.0      0.0
```

Figure 5: One-Hot-Encoding for "Payment Method"

On the other hand, for categorical features with a high cardinality of values such as "From bank," "To bank," and "Account" we explored multiple encoding techniques. We initially experimented with feature hashing, which converts each value into a fixed-size hash code. However, we observed that feature hashing led to either the loss of information or to the curse of dimensionality, resulting in a significant increase in the number of features and the requirement for exponential growth of training data needed in order to not face a degrade in the performance of our models. Therefore, we decided not to proceed with feature hashing for these numerical features.

Instead, we implemented two alternative encoding techniques: frequency encoding and target encoding. Frequency encoding replaces the values of the categorical attributes with the frequency of their occurrences within the dataset, capturing the relative importance of each category based on its frequency. Target encoding, on the other hand, replaces the values of the categorical features with the mean target value (in this case, the attribute "Is Laundering") for training data. In the following sections, we will present a comparison of the two approaches, leading us to the conclusion that the first approach is better suited to the dataset we worked with. The reason is that Target encoding relies too much on the target variable to encode the features. In fact, by incorporating the target variable into the encoding process, we inadvertently introduce information from the variable we are trying to predict (target leakage). Through these preprocessing steps, we ensured the dataset was suitably prepared for training our model.

5 Model Implementation: Decision Tree

5.1 Node Classifier

We started by creating a simple mono-dimensional classifier made of a single node (root). This node received as input an array of values (X) and an array of labels (Y), it counted the number of instances of each label class (1 or 0) and took the value of the max between the two as its value.

5.2 Tree Classifier

We then implemented a splitting algorithm to create child nodes. We started by sampling a random value from the array and using it as a threshold. All the data points in the array with values lower than the threshold would be passed to the left child as input, while the values higher than the threshold would be passed to the right child node. Then the process would continue recursively on each child node. To stop the process, we decided to add a maximum depth to the tree. Each node would receive its depth and pass it to the children nodes at depth + 1. The process stops when the current node's depth reaches the maximum depth value. We kept this parameter constant throughout the process, using it as a hyperparameter to maximize the classifier's performance while avoiding overfitting. Another parameter used as a stopping condition is the minimum number of samples for a split. In this case, we do not consider it as a hyperparameter, and we set it equal to 2. It is worth to note that we also developed an iterative version of the Decision Tree classifier in which we simulate the recursion with a stack. At the end, we adopted the recursive version because the algorithm seeems to us more readable.

5.3 Multi Dimensionality

We then implemented the multi-dimensionality aspect of a tree classifier. In fact, for every split, we had to perform both the selection of the best threshold among the feature values and the selection of the best feature on which the decision node splits itself. To do so, we decided to take inspiration from the classical CART decision tree classifier and use the Information Gain metric to identify the best feature and the best threshold for the split. Information gain quantifies the reduction of entropy (or gain of information) that a specific split on a specific attribute would cause. For a binary tree it is given by the formula:

$$\text{Information Gain}(v) = \text{Entropy}(S) - \left(\frac{|S_v^l|}{|S|}\right) \cdot \text{Entropy}(S_v^l) + \left(\frac{|S_v^r|}{|S|}\right) \cdot \text{Entropy}(S_v^r), \tag{1}$$

Where:

- S is the dataset before the split.
- S_v^l represents the subset of S whose elements value $v^{'}$ are smaller then value v of the feature being evaluated.
- S_v^r represents the subset of S whose elements value $v^{'}$ are greater then value v of the feature being evaluated.
- $|S_v^l|$ denotes the number of instances in subset S_v^l.
- $|S_v^r|$ denotes the number of instances in subset S_v^r.
- $|S|$ represents the total number of instances in the original dataset S.

Entropy is calculated using the formula:

$$\text{Entropy(S)} = -\sum_{v \in S} p(v) \cdot \log_2(p(v)) \tag{2}$$

Where $p(v)$ is the probability of an instance in S belonging to class v.

The problem with the Entropy metric is that it has to perform the logarithmic operation $O(n \cdot f \cdot 2h)$ times, where n is the number of features, f is the number of possible thresholds for each feature, and h is the max depth of the binary tree. So we decided to use a less time-consuming metric that is often used in this type of processes: the **Gini index**. The gini index is defined as:

$$\text{Gini}(S) = 1 - \sum_{v \in S} (p(v))^2 \tag{3}$$

and it is faster to calculate while still giving good results for our model.

So our final solution to calculate information gain is the following:

$$\text{Information Gain}(v) = \text{Gini}(S) - \left(\frac{|S_v^l|}{|S|}\right) \cdot \text{Gini}(S_v^l) + \left(\frac{|S_v^r|}{|S|}\right) \cdot \text{Gini}(S_v^r), \tag{4}$$

Once the highest information gain is determined, the tree checks if the current depth is still less than or equal to the maximum depth, and if the information gain is higher than 0 (which means that we are not in a pure node), it splits the current dataset into two child nodes on the selected threshold for the selected feature.

5.4 Dealing with Imbalanced Dataset

Due to the high imbalance between the labels, we decided to alter the function to calculate the value of a leaf. Instead of a simple majority voting, we chose to use a weighted count function for each class and select the class with the overall maximum weight. We assigned a weight of 1 to the data points labeled as ”not laundering” and a weight whose value was determined by parameter ”weight_1” to the data points labeled as ”laundering”.

5.5 Model Training and Hyperparameter Tuning

To optimize the Decision Tree's performance we decided to evaluate some important hyperparameters, which are:

- Weight for values equal to 1 in the attribute ”Is Laundering”
- Max depth of the trees
- Number of thresholds

We ran a cross-validation process with multiple values to find the best combination of hyperparameters. For the number of thresholds, we had to balance between a higher probability of finding better cut points and the impact that the growing number of thresholds has on the classifier's temporal performance. To mitigate this problem, we implemented a solution: every time we have to evaluate the best split point for a node, the algorithm automatically chooses the minimum between the number of possible thresholds and the unique values for the feature in the current dataset. After some tests we realized that whatever value we used for the number of thresholds did not change the performance of the decision tree that much, but high values increased the execution time of the algorithm by a lot. For this reason we decided to set the value of the thresholds number to 10.

6 Model Implementation: Random Forest

In addition to the decision tree classifier, we also implemented a Random Forest model to further enhance the predictive capabilities of our analysis. The Random Forest algorithm utilizes the concept of ensemble learning, where multiple decision trees are trained independently on different subsets of the data. The final prediction is then determined by aggregating the predictions of all the trees.

Algorithm	Data	Data for each tree	Number of trees	Max depth of the trees	Thresholds	Time
Sequential	2000000	100000	100	10	10	**1951 sec ≈ 33 min**
Parallel	2000000	100000	100	10	10	**645 sec ≈ 11 min**

Table 1: Computation time Sequential vs Parallel Random Forest

6.1 Bootstrapping

An important aspect of training a random forest is the method used to create each tree. In our project in order to have trees trained on different data we decided to implement a bootstrapping technique. Bootstrapping is a statistical resampling technique used for estimating the distribution of a statistic by repeatedly resampling, with replacement, from the observed data. In our case, for every tree in the forest, we sample a number of rows based on a hyperparameter called "Fraction", which determines the fractions of the given dataset that the trees utilize for the training. For example, if the input dataset contains 100000 datapoints and the value of "fraction" is 0.2, then each forest tree receives 20000 data points as input. The training samples are chosen at random and with replacement, meaning that multiple trees could potentially be trained on overlapping pools of datapoints.

```
# ROWS BOOTSTRAP (With Replacement)
training_sample = df.sample(withReplacement=True, fraction = self.fraction)
```

To introduce more variation between each tree we decided to implement variations also on the attributes. For each tree, a random number between 1 and 5 of random attributes is not considered.

```
# columns BOOTSTRAP
# choose the N of columns to not consider
n = {random.randint(1, 5)}
# randomly choose which are those N columns
random_features = {random.sample(training\_X.columns, n)}
# drop the chosen columns
bootstrapped\_training\_X = training\_X.drop(*random\_features)
```

6.2 Distributed Execution

One advantage of the Random Forest algorithm is its ability to execute the training and prediction process in a distributed manner. Since each decision tree in the Random Forest operates independently, they can be trained and evaluated simultaneously, taking full advantage of distributed computing architectures and reducing the overall training time.

We opted to implement a map function that takes a parameter 'n,' representing the number of samples, where 'n' corresponds to the number of trees in the forest. For each 'n,' the function creates and fits a Tree Classifier structure, resulting in a trained tree. These trained trees are saved and later employed for making predictions.

To validate the distributed nature of our program, we conducted a performance comparison between the computation times of creating and training the random forest in both distributed and sequential environments. The results, as depicted in table 1, demonstrate a remarkable reduction in computation time by one-third with the distributed program.

6.3 Output Aggregation

During the prediction phase, each tree returns an array of labels corresponding to each (unlabeled) input data point. In order to aggregate the result we decided for each index to count the occurrences of both classes and chose the one with a higher count. We later decided to change this simple count and implement a count based on a different weight between classes. In the case of a more comprehensive investigation, we believe that these weights (or the weight ratio between the two classes) could have served as valuable hyperparameters for the forest. However, due to constraints such as limited computational resources and training time required to thoroughly cross-evaluate all the parameters, we made the decision to maintain them at a fixed value determined through prior analysis and basic testing.

6.4 Model Training and Hyperparameter Tuning

To optimize the Random Forest model's performance we decided to evaluate some important hyperparameters:

- Number of Trees
- Max depth of the trees
- Weight for values equal to 1 in the attribute "Is Laundering" for each tree

Since the optimal values were found for the last two parameters (weight_1 and max_depth) during cross-validation of the decision tree. The choice of the best hyperparameter was limited to the number of trees.

7 Model Evaluation and Analysis of Results

7.1 Evaluation Metrics

For this particular project, we had to optimize the ability of our classifier to assign the correct label to each data frame. But in the dataset that we analyzed and used to train our Tree Classifier the labels were highly unbalanced. This means that the "Laundering" to "Non-laundering" ratio was about 1/1000. It is also important to contextualize the meaning of our system. In systems of this nature, assigning the correct label of "Laundering" to a transaction holds greater significance in terms of costs compared to accurately labeling a transaction as "Non-Laundering." We expand this concept in 7.1.1. For this reason the first metric that we took into consideration is the **Recall**. Recall, also known as the true positive rate or sensitivity, measures the proportion of positive instances correctly identified by the classifier. Given the rarity of positive outcomes in our dataset, maximizing Recall becomes essential to minimize the number of false negatives. The Recall is calculated as:

$$Recall = \frac{TP}{TP + FN}$$

For our institution, the foremost priority is minimizing the number of false negatives. In our analysis, we identified a straightforward method to optimize this metric by increasing the parameter associated with the weight of the "Laundering" class. However, this approach would result in a classifier heavily biased toward labeling data points as positive, potentially misclassifying a significant number of negative transactions as positive. For this reason we decided to also try to optimize our classifier based on a more balanced metric, we chose the F1 score.

$$F1Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

The F1 score is calculated by taking the harmonic mean of precision and recall, where precision and recall are multiplied by 2 and divided by their sum.

We recall that Precision is calculated as:

$$Precision = \frac{TP}{TP + FP}$$

By employing the F1 score, we aimed to find a balance between these two aspects, seeking a classifier that demonstrates strong performance in identifying true positives while keeping false positives under control. For the fact that our dataset is strongly imbalanced we did not consider a simple metric such accuracy.

7.1.1 Illicit Transactions Estimated Costs

The cost of an illicit transaction to a bank can vary significantly depending on several factors, including the nature and scale of the illicit activity, the jurisdiction in which the bank operates, the regulatory environment, and the bank's risk management and compliance measures. Illicit transactions can encompass a wide range of activities, such as money laundering, fraud, sanctions violations, and terrorist financing.

Here are some potential costs and consequences that a bank may incur due to illicit transactions:

1. **Financial Penalties:** Regulatory authorities may impose substantial fines and penalties on banks found to be involved in or facilitating illicit transactions. These fines can range from millions to billions of dollars.
2. **Legal Costs:** Legal expenses related to investigations, litigation, and compliance efforts can be substantial.
3. **Reputation Damage:** Being associated with illicit activities can severely damage a bank's reputation, leading to loss of customer trust and business.
4. **Loss of Correspondent Banking Relationships:** Banks engaging in illicit transactions may find it challenging to maintain correspondent banking relationships with other financial institutions, limiting their ability to facilitate international transactions.
5. **Increased Compliance Costs:** Banks may need to invest heavily in compliance programs, technology, and personnel to prevent, detect, and report illicit transactions, adding to their operational expenses.
6. **Customer Loss:** Some customers may choose to close their accounts with a bank implicated in illicit transactions, leading to a loss of business.
7. **Regulatory Scrutiny:** Banks involved in illicit transactions may face enhanced regulatory oversight and audits, leading to additional compliance costs and restrictions.
8. **De-risking:** Banks may adopt a conservative approach and "de-risk" by closing accounts or refusing services to customers in high-risk industries or regions to minimize exposure to illicit activities. This can result in missed revenue opportunities.

It's important to note that the cost of an illicit transaction can be extremely high, not only in financial terms but also in terms of reputational damage and long-term consequences for the bank's business. On the other hand, the cost of a false positive is limited to an apporfunded check by the bank against the transaction and all data related to it, including the bank account holder. The costs of such a check are not low, but certainly orders of magnitude less than those associated with a false negative. Under these hypothesis we can assume that the **cost for a bank of false negative is one-thousand times the cost of a false positive**.

Our goal was to minimize the total cost for a bank involved in this kind of problem. The total cost is a computed as:

$$TotalCost = FP \times CFP + FN \times CFN \tag{5}$$

Where:

- CFP is the unitary cost for a false positive
- CFN is the unitary cost for a false negative
- CFN = 1000 $\times CFP$

Later on, we will show through our results that **maximizing the Recall is the right choice for minimizing the total cost**.

7.2 Test Results

To find the best results for this problem we have performed these actions:

1. **Cross Validation for the decision tree classifier**
2. **Training and test on the entire dataset with the best hyperparameters for the decision tree**
3. **Choice of the best hyperparameters for the random forest**
4. **Training and test on the entire dataset with the best hyperparameters for the random forest**

7.2.1 Cross Validation for the Decision Tree Classifier

In this phase, we conducted a 3-Fold Cross-Validation to identify the pair of hyperparameters that would maximize recall and the pair that would maximize precision. At the time, we were uncertain which of these two metrics would minimize the overall cost to a bank. The Cross-Validation was performed with a dataset consisting of two million data points. The two hyperparameters under consideration were the Max Depth of the Tree and the Weight assigned to the label ”1”. The values for the first one were 7 and 10, and the values for the second one were 5, 10 and 100. At the end of the process, we chose the pair (7,100) to maximize the recall. We reproduced the same validation for the dataset with target encoding and we obtained the following pairs (10,10) for maximizing the recall.

7.2.2 Test for the Decision Tree Classifier on the Full Dataset

Subsequently, we trained our Decision Tree Classifier with all the datapoints (five million) and with the best hyperparameters. **From the results we realized that the metric to be maximized in order to minimize the total cost is recall.** We show the final result in Table 2. In Figures 6,7, 8 we sum up all the results showing the comparison between Frequency Encoding and Target Encoding for each metric.

Encoding	MaxDepth	Weight1	Recall	Precision	F1
Frequency	7	100	**0.6**	0.019	0.036
Target	10	10	0.022	0.027	0.024

Table 2: Best Test Results for Frequency and Target encoding. This Table shows how Frequency encoding is better than Target encoding.

7.2.3 Random Forest Hyperparameters Search

From the results of the decision tree tests, we inferred that the best scores came from choosing the max_depth parameter equal to 7. As for the weight_1 parameter, we kept the optimal value, which is 100 for maximizing recall. The other considered hyperparameter was the number of trees in the forest with values 20, 100 and 250. Unlike the approach taken for the decision tree, we did not conduct a Cross-Validation in this case. In random forests, there is no need for cross-validation to obtain an unbiased estimate of the test set error, as it is internally estimated during the run. For this reason, we have chosen another approach. First of all, we sampled a subset of the original dataset with 2 milion data points. Then, we trained and validated our model for each combination of hyperparameters on the same subset of datapoints. At the end of the process, we chose the pair (100,100) to maximize the recall. We reproduced the same validation for the dataset with target encoding and we obtained the following pair (250,100) for the recall.

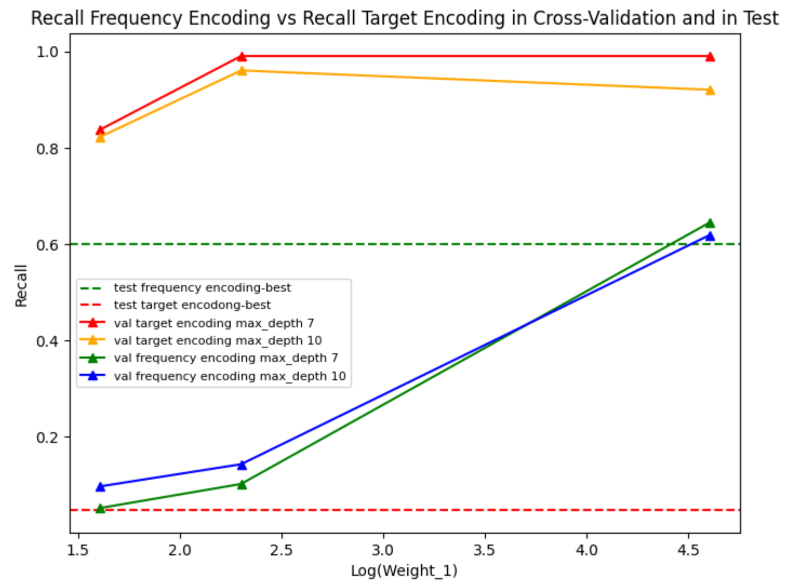


Figure 6: Recall in DecisionTree

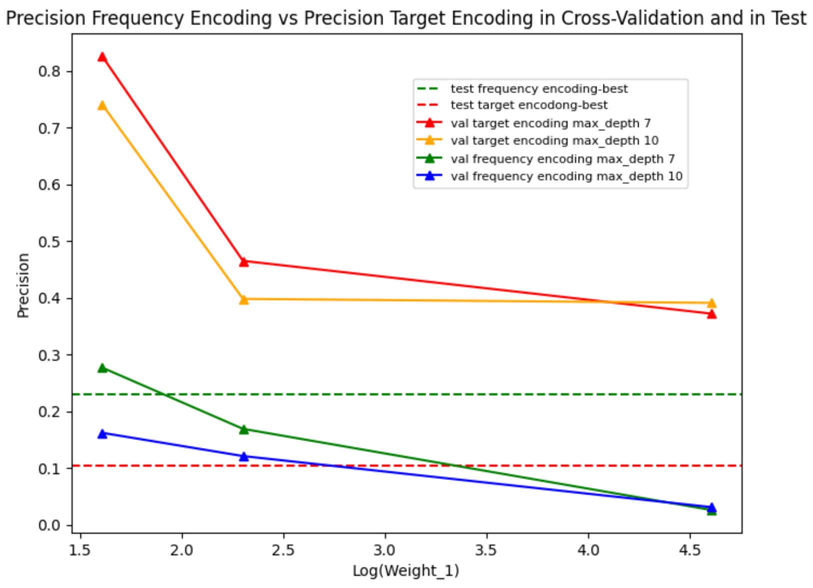


Figure 7: Precision in DecisionTree

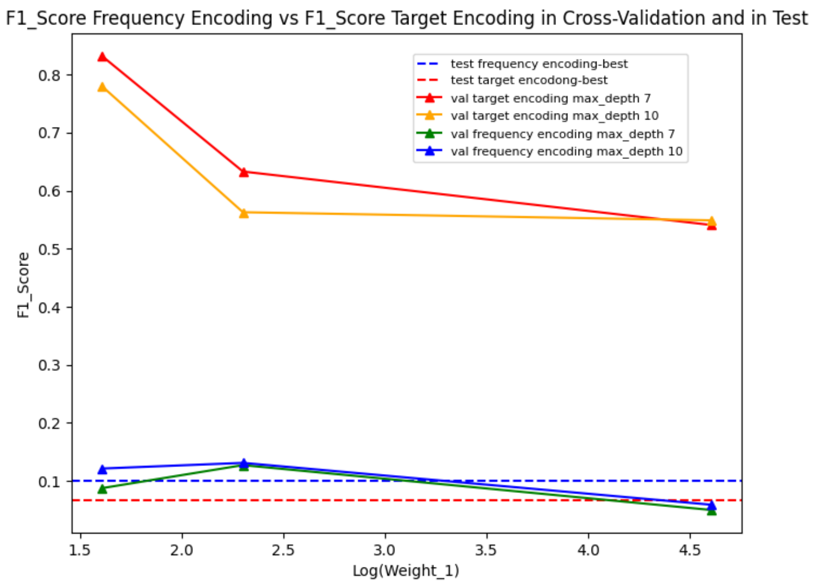


Figure 8: F1 in DecisionTree

7.2.4 Baseline

In table 3 we report the results of our tests with the choice of the best hyperparameters for the Random Forest and we made a comparison between recall, precision and F1Score of our model and the baseline of the same metrics. We computed the baseline of our metrics assuming we computed them with a random classifier that takes as input a dataset with the same proportion of zeros and ones as the original dataset. Assuming that this random classifier with probability 1/2 assigns a positive label to a transaction and with probability 1/2 assigns a negative label to it, we computed the following values in this way:

- Proportion of Ones = 0.0007
- Proportion of Zeros = 0.9993
- TP = Probability Of One $\times ProportionOfOnes = \frac{1}{2} \times 0.0007$
  $FP = ProbabilityOfZero \times ProportionOfZeros = \frac{1}{2} \times 0.9993$
  $FN = ProbabilityOfOne \times ProportionOfOnes = \frac{1}{2} \times 0.0007$

$$Recall = \frac{TP}{TP + FN} = \frac{1}{2} \tag{6}$$

$$Precision = \frac{TP}{TP + FP} = 0.0007 \tag{7}$$

$$F1Score = \frac{2 \times Recall \times Precision}{Recall + Precision} = 0.0014 \tag{8}$$

7.2.5 Test for the Random Forest on the Full Dataset

Subsequently, we trained our Random Tree Classifier with all the datapoints (five milion) and with the best hyperparameters. Then, we obtained the following results:

Encoding	NumTrees	Weight1	Recall	Precision	F1
Frequency	100	100	**0.989**	0.008	0.015
Target	250	100	0.58	0.004	0.008
Baseline	/	/	0.5	0.0007	0.0014

Table 3: Best Test Results for Frequency and Target encoding. This Table shows how Frequency encoding is better than Target encoding

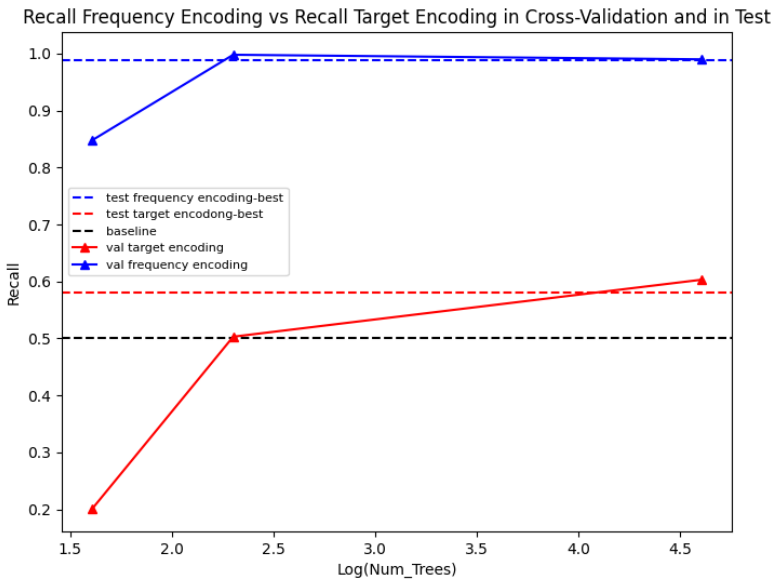


Figure 9: Recall in Random Forest

In 9 , 10 we sum up all the results showing the comparison between Frequency Encoding and Target Encoding for each metric.

8 Conlcusions

The problem of identifying illicit transactions is not simple, in fact they are rare events compared to normal lawful transactions and for this reason it is difficult for a machine learning algorithm to grasp the underlying patterns in the data to be able to predict the legality or otherwise of a transaction. Given these premises, our work does not aim to say whether a transaction is illicit or not with one hundred percent correctness but rather to minimize the cost to a bank in handling this type of problem. From what we had already expressed in 7.1.1, in table 4 we show the results for various combinations of hyperparameters for the random forest with frequency encoding and for each of these, the estimated total cost to the banks, including possible costs for damage caused by a false negative and those caused by a false positive. **The results show that, using our model, it is convenient for a bank to maximize the recall metric and thus minimize the number of false negatives.** According to table 5, our forest reduces the bank cost to 1/41 with respect to not identifying illicit transactions and to 1/50 with respect to the random classifier.

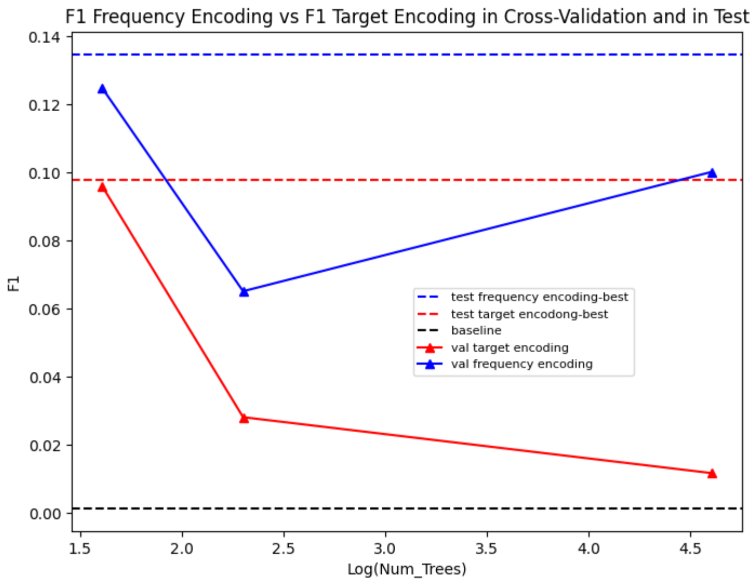


Figure 10: F1 in Random Forest

DataEachTree	NumTrees	Weight1	Recall	Precision	F1	TP	FP	FN	TN	Total Cost
100000	20	5	0.209	0.089	0.124	134	1364	506	397996	507364
100000	20	100	0.848	0.0141	0.027	543	37968	97	391392	134968
100000	100	5	0.471	0.035	0.065	302	8173	338	391187	346173
100000	100	100	0.998	0.0077	0.015	639	82349	1	317011	**83349**
100000	250	5	0.28	0.006	0.1	180	2669	460	396691	462669
100000	250	100	0.995	0.0058	0.011	634	108114	6	291246	114114
250000	100	100	0.989	0.008	0.015	633	78492	7	320868	**85492**

Table 4: Metrics and Cost for the Random Forest for different Hyperparameters. The First Six Rows represents the Total Cost for each Combination of Hyperparameters. The Last Row represents the Total Cost on Random Forest tests

System Adopted	Recall	Precision	Total Cost
Uncontroled	0	1	3500000
Random Classifier	0.5	0.0007	4249125
Our Model	0.989	0.008	**85492**

Table 5: Performance Comparision between Our Random Forest with Frequency Encoding and Baselines