

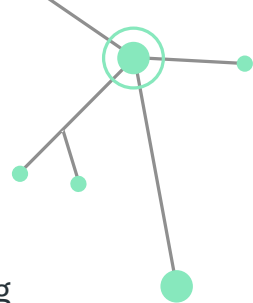
# Random Forest Improvement for binary classification

---

MACHINE LEARNING I - 2023/2024

Work by: Francisco Carqueija (202205113), Marta Longo (202207985), Sara Táboas (202205101)

# Introduction



It was proposed to us to develop a project that would focus on understanding and enhancing a selected supervised machine learning algorithm. The primary objectives of our work rely on:

- **Understanding the Algorithm:** Gaining both theoretical and empirical insights into how the chosen algorithm works.
- **Benchmarking:** Evaluating the performance of the against standard benchmark datasets.
- **Data characteristics:** Investigating how specific data characteristics affect the algorithm's performance.
- **Algorithm improvement:** Proposing and implementing a variation of the algorithm to improve its robustness to selected data issues.

By the end of this project, we aim to demonstrate a thorough analysis and enhancement of the chosen classification algorithm, providing insights into its strenghts and areas for improvement.





# Table of contents

## 01 Selected Algorithm

Classification algorithm overall description

## 02 Data Characteristics and Problem Statement

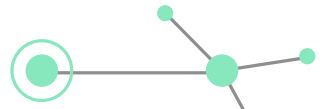
Data characteristic we chose to tackle

## 03 Empirical Study

Experimental setup and analysis of the results

## 04 Conclusions

Our findings after evaluating the behaviour of the proposed variant on the same set of datasets and compare the results with the original version





01

# Selected Algorithm

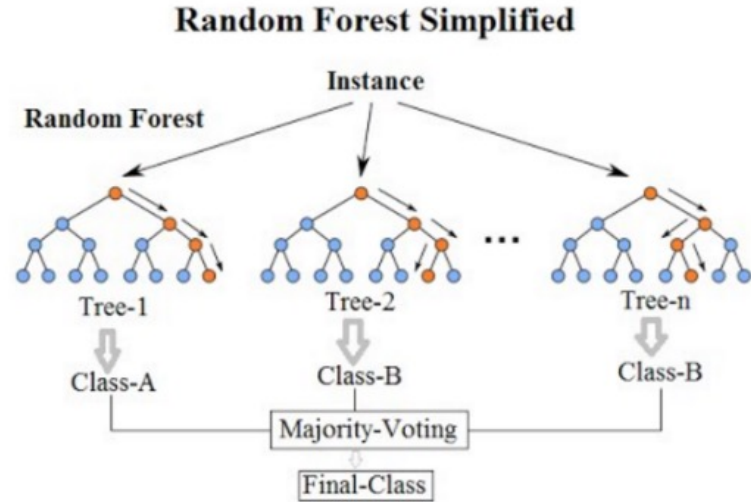
---

Random Forest



# Random Forest

Random Forest is an ensemble learning algorithm primarily used for classification and regression tasks. It constructs multiple decision trees during training and merges their outputs to improve the final prediction's accuracy and control overfitting. Each tree in a random forest is built from a bootstrap sample of the training data, and when splitting nodes, a random subset of features is considered. This randomness helps make the model robust to overfitting and increases its generalizability.



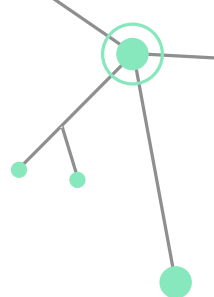
# Characteristics

## Strengths

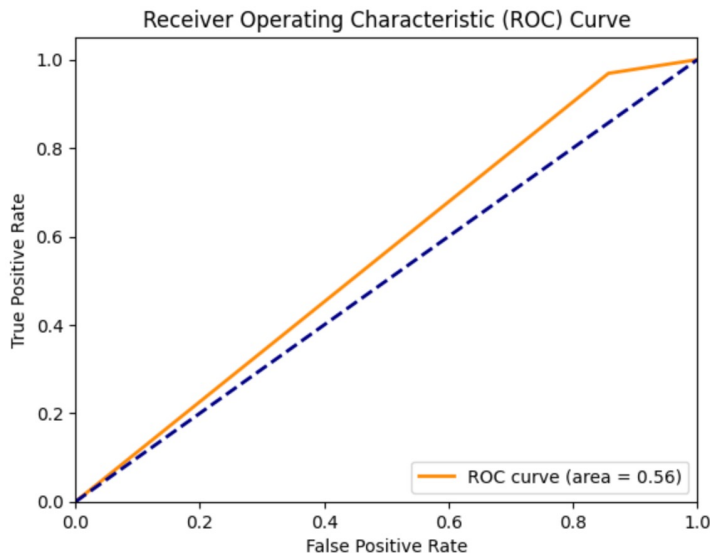
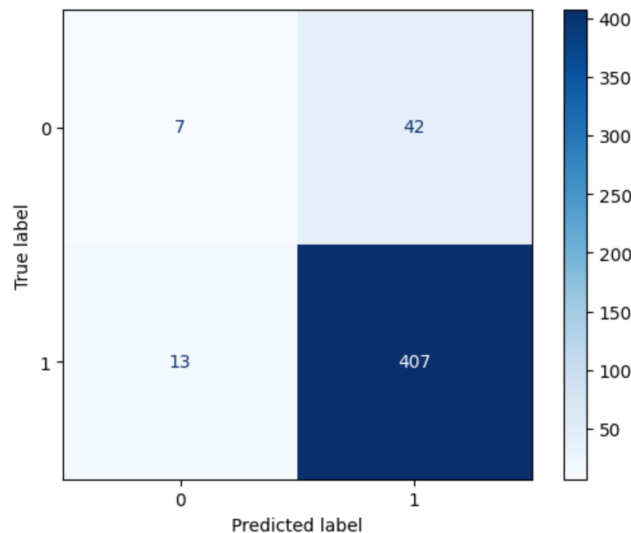
- **Robustness:** Less likely to overfit compared to individual decision trees due to averaging.
- **Versatility:** Can handle both classification and regression problems. Easy to define/tune hyper-parameters.
- **Feature Importance:** Provides insights into feature importance.

## Weaknesses

- **Computational Cost:** More resource-intensive than single decision trees, especially with many trees.
- **Interpretability:** More complex to interpret than a single decision tree.



# Example of Standard Random Forest Results



**Accuracy score = 0.8827292110874201**

**Balanced Accuracy Score = 0.555952380952381**



# 02

## Data Characteristics Problem Statement

---

Class Imbalance in Binary Problems

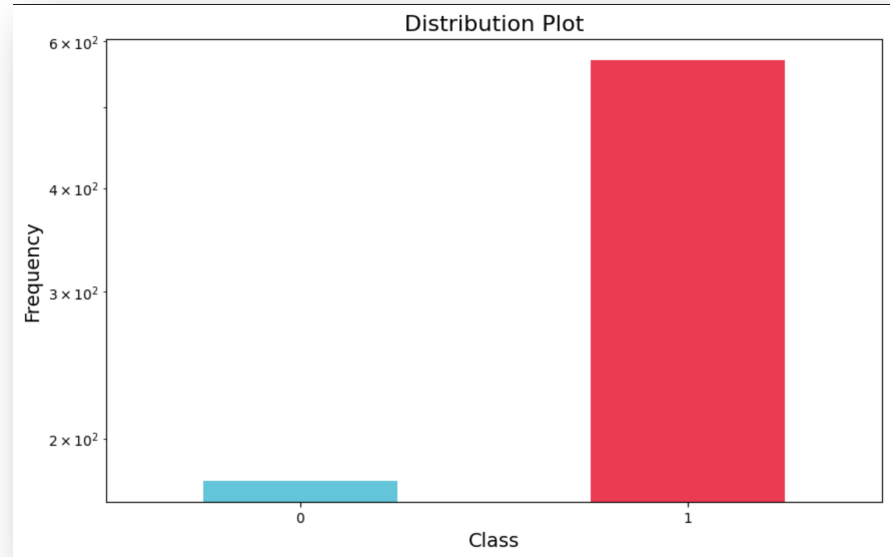


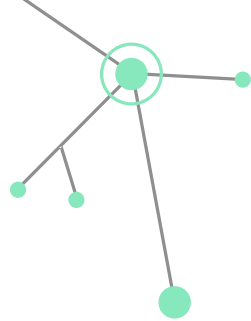


# Class Imbalance

**Class Imbalance** occurs when the distribution of classes is not uniform, i.e., some classes have significantly more instances than others.

Algorithms may become biased towards the majority class, leading to poor predictive performance on the minority class.



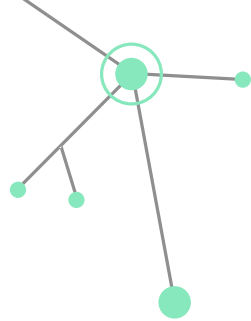


# Behaviour of the Algorithm Concerning Class Imbalance in Binary Problems:

In binary classification problems with class imbalance, Random Forests can exhibit certain behaviours. Primarily, they **may develop a bias towards the majority class**, as the trees in the forest are trained on **samples that predominantly contain more instances of the majority class**.

**This can lead to skewed predictions** where the model tends to favour the majority class over the minority class. As a result, performance metrics like accuracy can be misleadingly high, while more informative metrics such as precision, recall, balanced accuracy and the F1-score for the minority class may be lower.





## Proposed Approach:

Modify the Random Forest algorithm to include **balanced class weights**. This involves adjusting the weights of the classes inversely proportional to their frequencies in the training data. By doing so, the model will pay more attention to the minority class, reducing bias towards the majority class.

## Expected Outcomes:

- Higher recall for minority class
- More balanced performance metrics
- Generalization to new data





**03**

# **Empirical Study**

---

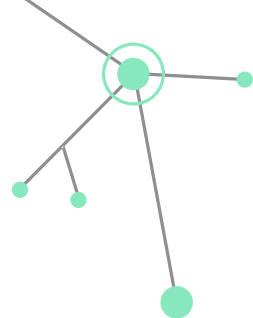


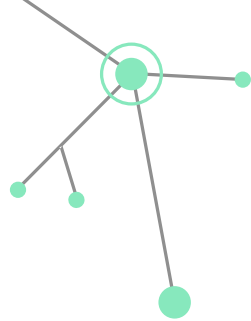
# Datasets Used:

- **Data set 1:** (RangeIndex: 5473 entries, 0 to 5472 || Data columns (total 11 columns) || dtypes: float64(10), int64(1) || memory usage: 470.5 KB)
- **Data set 2:** (RangeIndex: 1563 entries, 0 to 1562 || Data columns (total 38 columns) || dtypes: float64(29), int64(9) || memory usage: 464.1 KB)
- **Data set 3:** (RangeIndex: 540 entries, 0 to 539 || Data columns (total 21 columns) || dtypes: float64(18), int64(3) || memory usage: 88.7 KB)
- **Data set 4:** (RangeIndex: 768 entries, 0 to 767 || Data columns (total 9 columns) || dtypes: float64(8), int64(1) || memory usage: 54.1 KB)
- **Data set 5:** (RangeIndex: 748 entries, 0 to 747 || Data columns (total 5 columns) || dtypes: float64(1), int64(4) || memory usage: 29.3 KB)
- **Data set 6:** (RangeIndex: 2310 entries, 0 to 2309 || Data columns (total 20 columns) || dtypes: float64(16), int64(4) || memory usage: 361.1 KB)
- **Data set 7:** (RangeIndex: 458 entries, 0 to 457 || Data columns (total 40 columns) || dtypes: float64(19), int64(21) || memory usage: 143.3 KB)
- **Data set 8:** (RangeIndex: 748 entries, 0 to 747 || Data columns (total 5 columns) || dtypes: float64(1), int64(4) || memory usage: 29.3 KB)
- **Data set 9:** (RangeIndex: 1109 entries, 0 to 1108 || Data columns (total 22 columns) || dtypes: float64(18), int64(4) || memory usage: 190.7 KB)
- **Data set 10:** (RangeIndex: 32769 entries, 0 to 32768 || Data columns (total 10 columns) || dtypes: int64(10) || memory usage: 2.5 MB)

## Data characteristic of interest: Imbalanced Binary Class

\*all of the datasets used were obtained in the OpenML-CC18 Curated Classification benchmark





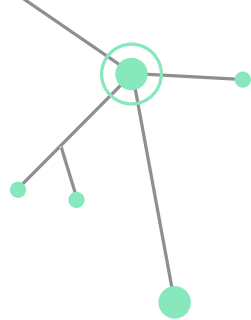
# Hyperparameters of the Algorithm:

The two main hyper-parameters of random forests are the **number of base models to generate** and the **number of attributes to randomly select at each node**.

In this context we selected 200 as the number of estimators. However, if we wanted to obtain more reliable statistics for the attribute importance, we would use 1000 trees for the experiment.

The number of attributes selected at each split-node were the number of the attributes of each respective dataset.





# Improvement of the algorithm

The original algorithm calculates the entropy with the proportion of each class. Then, to ensure that the algorithm would pay more attention to the minority class during training, we adjusted the class weights to be inversely proportional to the class frequencies in the training data.

To do this, we followed these steps:


1. Calculate the weights of each class on the dataset
2. Calculate the weighted entropy by assigning the weights and then multiplying them by each class proportion
3. Apply the weighted entropy as the splitting criterion of each random forest tree.

```
def calculate_class_weights(y):  
    class_count = Counter(y)  
    total_samples = sum(class_count.values())  
    class_weights = {cls: total_samples / class_count[cls] for cls in sorted(class_count)}  
    return class_weights
```



# Results Analysis

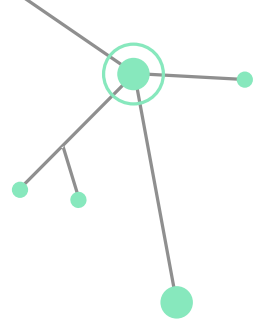
After implementing the weighted version of the algorithm, we observed a decrease in overall accuracy score and an increase in balance accuracy score and ROC Area Under Curve score (when comparing the results of the dataset used in slide 7).



**Accuracy score** – the decrease of this score is expected due to the redistribution of the model's focus. Lower accuracy scores mean the model is no longer predicting almost all the instances as the majority class.

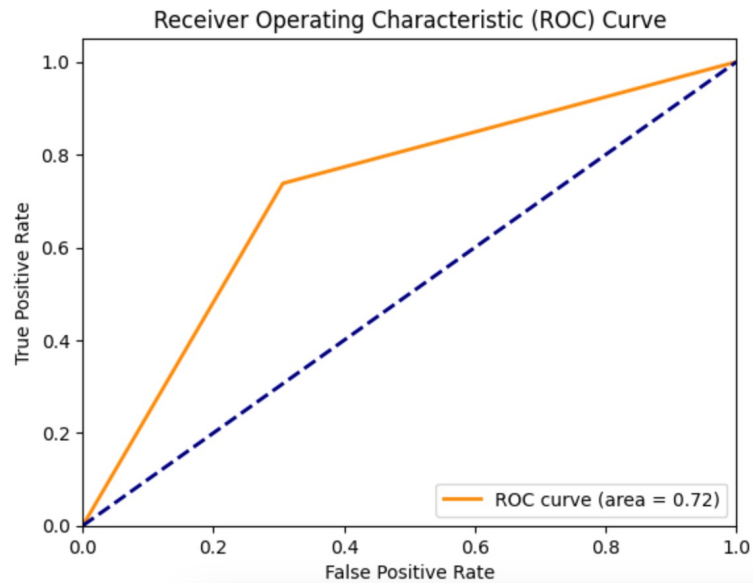
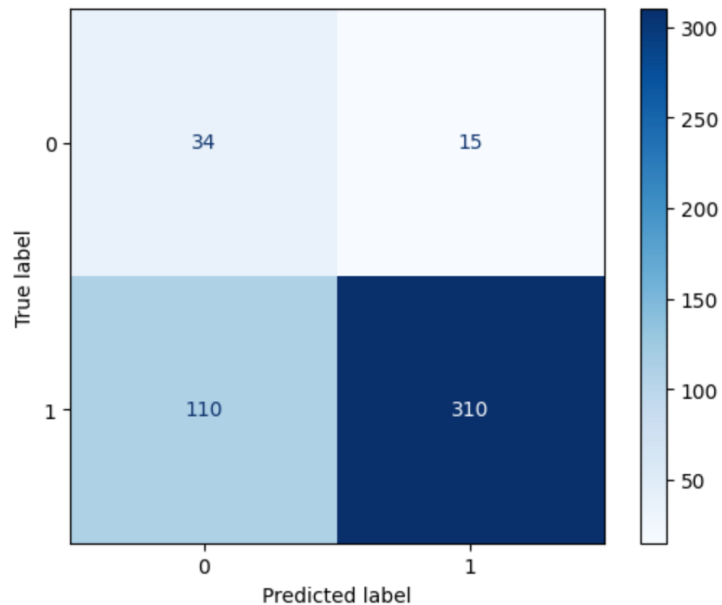
**ROC AUC score** – the increase of this score demonstrates that our improved Random Forest model has a greater ability to correctly differentiate between the minority and majority classes

**Balanced accuracy score** – the increase of this score indicates that our model is now more equitable in its predictions, effectively recognizing and correctly classifying instances from both classes. It demonstrates that the model's performance is no longer biased towards the majority class.





# Results Analysis from the previous example



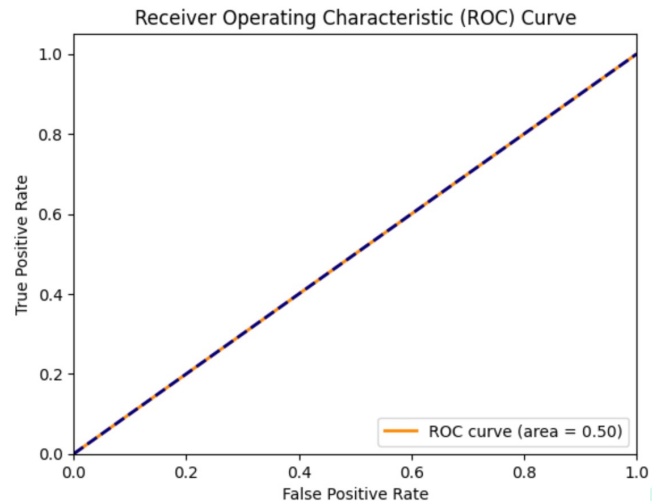
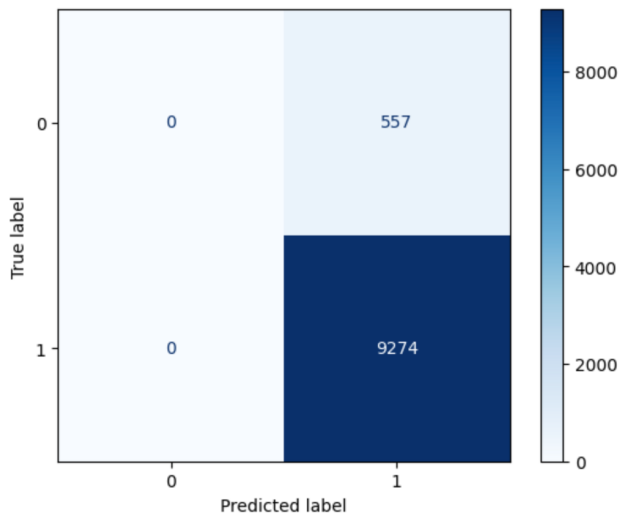
**Accuracy score = 0.7334754797441365**

**Balanced Accuracy Score = 0.7159863945578231**

# Exception:

## Standard Random Forest Version

In one of our datasets, the standard version of the algorithm classifies all instances as belonging to the majority class. This results in a high overall accuracy ( $\sim 0.94$ ) because most instances are correctly classified as the dominant class. However, this approach leads to a very low balanced accuracy (0.50), as the model fails to correctly identify any instances of the minority class

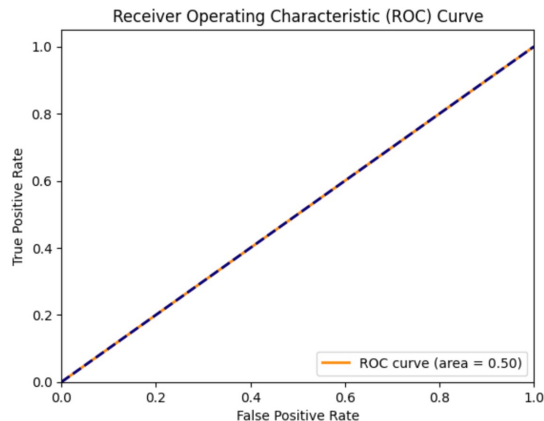
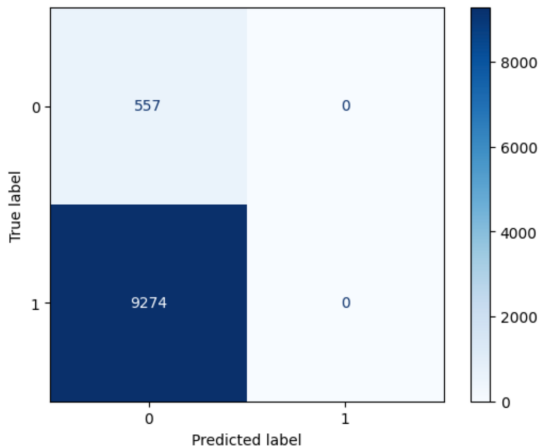


# Exception:

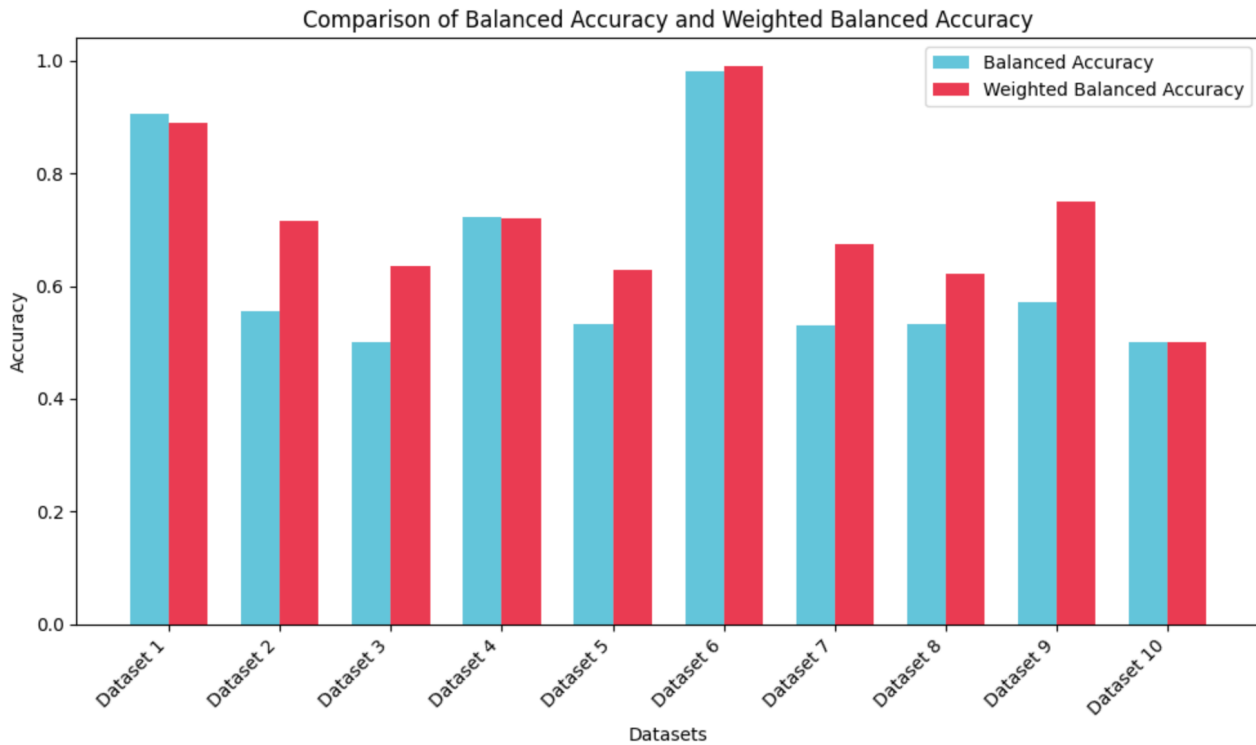
## Weighted Random Forest Version

However, when applying the weighted version of the algorithm, all instances are being classified as the minority class. This occurs because of the overcompensation for the class imbalance.

The increased weight for the minority class makes the algorithm focus excessively on minimizing errors for this class. The classifier might learn that predicting the minority class for most instances minimizes the weighted error more effectively than a balanced prediction.



# Results Analysis





**03**

# **Conclusions**

---



## To sum up,

- Enhancing Random Forest classifiers to address class imbalance improves their performance, providing more realistic and reliable results.
- By incorporating class weights, the algorithm effectively manages the disparity between majority and minority classes, preventing bias towards the majority. This adjustment leads to more balanced predictions and better metrics such as precision, recall, balanced accuracy and F1-score.
- Empirical evidence shows that this approach ensures robust models that deliver accurate and fair predictions, crucial in real-world applications where misclassifying the minority class is costly.

