Comparative Analysis of Ensemble Methods for Binary Classification

Antoine Sirvent; Edouard Chappon

1 Introduction

Real-world datasets often have imbalanced in class distribution. In such cases, single models can struggle to capture the true patterns underlying the data, either due to high bias or variance errors. Ensemble methods offer a way to partly solve these issues by combining multiple models or base learners. This has been shown to improve stability and predictive accuracy.

In this project, we study ensemble methods applied to binary classification, using a 25 datasets. We focus on how different ensemble strategies perform when the base learners are logistic regression or decision trees. Moreover, to address class imbalance, we investigate whether undersampling, oversampling, and hybrid sampling can boost performance compared to a baseline model that does not employ any rebalancing.

Our aim is to identify under which conditions each of these rebalancing or ensemble approaches yields the most accurate and robust performance. By examining multiple hyperparameter settings and reporting performance metrics such as the F1-score, we seek to find impact of hyperparameters, algorithm and method to handle imbalance within the class distribution.

Outline. This work is organized as follows :
In **Section 2** We explain reduction error properties and we describe the ensemble methods that we use along the study. Then, we describe in **Section 3** the 25 benchmark datasets used in our experiments. Those experiments are detailed in **Section 4** including the cross-validation procedure, the sampling strategies, and the hyperparameter configurations explored. We prensent an analyse of the results in the **Section 5**. We finish this project by the **Section 6** where we summarize the key insights about hyperparameter impact, and we give practical advice about selection of ensemble techniques and balancing approaches.

2 Ensemble method algorithms

Limitations of Single Models : A single model may exhibit high bias or high variance and struggle to capture the complexity and variability of the data. Ensemble methods help to overcome these issues to some extent.

Theoretical Reduction of Error Variance : Consider the models $(h_k)_{k=1}^K$ and the true label $r(x) \in \{-1, 1\}$ in a binary classification context. The aggregated hypothesis is defined as the average of the predictions (the final prediction is obtained by applying the sign function) :

$$H(x) = \frac{1}{K} \sum_{k=1}^{K} h_k(x)$$

If we define the error of each model as $\epsilon_k(x) = \Big|h_k(x) - r(x)\Big|$, then the quadratic error of the ensemble model is given by :

$$E\Big[\Big(H(x) - r(x)\Big)^2\Big] = E\Big[\Big(\frac{1}{K}\sum_{k=1}^{K}(h_k(x) - r(x))\Big)^2\Big] = E\Big[\Big(\frac{1}{K}\sum_{k=1}^{K}\epsilon_k(x)\Big)^2\Big]$$

If the errors $\epsilon_k(x)$ are centered and uncorrelated, then :

$$E\Big[\Big(\frac{1}{K}\sum_{k=1}^{K}\epsilon_k(x)\Big)^2\Big] = \frac{1}{K^2}\sum_{k=1}^{K} E\Big[\epsilon_k(x)^2\Big] = \frac{1}{K}\overline{E\Big[\epsilon^2(x)\Big]},$$

This analysis shows that, in theory, for the aggregated estimator, the error variance is reduced by a factor of $1/K$. However, this optimal reduction is only achieved if the errors are independent, which is not the case in practice.

In this study, we combine models of the same type, except for the stacking method. Thus, the only variability that could be introduced by aggregating our models via their average would be the hyperparameter settings. This approach is not effective, as the errors are then highly correlated.

Combination Strategies : To create our ensemble models, we choose a type of base model. Here, we opt for logistic regression and decision trees. We focus on four methods :

- **Bagging (Bootstrap Aggregating) :**
  This method involves taking a single already strong model and training it on several training sets generated from the original dataset using the bootstrap technique, which consists of sampling with replacement as many data points as in the original set. Thus, each model will contain some duplicate data in its training set (and hence assign them more weight), while other data may be missing.

  Bagging introduces diversity by training the models with different bootstrap datasets. The final prediction is obtained by aggregating the results of all models using majority voting. This approach reduces the error variance compared to a model trained on the entire training set.

  In this project, we use bagging with logistic regression models and decision trees, also known as Random Forest.
- **Boosting Methods :**
  While Adaboost combines strong models that may exhibit high error variance, boosting methods consist of sequentially training weak models h_k in order to correct the training errors —and thus the bias— of the previous models. These models are then combined by weighted majority voting based on their performance :

  $$H = \sum_{k=1}^{K} \alpha_k h_k.$$

- **Adaboost :**

  Adaboost is a boosting method that focuses, for each new model, on the data for which previous models performed poorly. At each iteration k, a new model is trained by assigning a higher weight w_i^k to the examples misclassified by previous models via the weighted error ϵ_k. The current model is then weighted by $\alpha_k = \frac{1}{2} \ln\left(\frac{1-\epsilon_k}{\epsilon_k}\right)$, and the new weights are updated from the previous weights by $w_i^{k+1} = w_i^k \exp\Big(-\alpha_k \, y_i \, h_k(x_i)\Big)$, up to a normalization factor.

  We apply Adaboost with logistic regression models and decision trees.

- **Gradient Boosting :**
  After an initial model h has been fitted on the dataset to minimize the overall loss, Gradient Boosting trains each model h_k to predict the residual r_k of the loss by minimizing the quadratic error $r_k(x)$ for all examples. Here, r_k is the negative gradient of the loss function with respect to the partial model : $H_{k-1} = h + \sum_{l=1}^{k-1} \alpha_l h_l$. Finally, the weight α_k is computed by minimizing the overall loss of the new model H_k.

  This approach allows the algorithm to be adapted to different types of loss functions, both in regression and in classification. We apply Gradient Boosting only with decision trees.

- **Stacking :**
  Stacking is an ensemble learning method that combines several base models (first level) via a meta-model (second level) in order to optimize the final prediction [2]. First, several models h_k (with $k = 1, 2, \ldots, K$) are trained on the initial dataset to generate predictions $h_k(x_i)$ for each example x_i. These predictions then form a new set of explanatory variables. In a second step, a meta-model g is trained on these outputs to learn how to combine the different predictions, so that the final prediction is given by :

$$\hat{y}(x) = g\Big(h_1(x), h_2(x), \ldots, h_K(x)\Big).$$

3 Datasets

In this section, we describe the datasets used in our experiments. We outline key characteristics of each dataset, such as the number of samples, number of features, class balance, and the presence of missing values. All 25 datasets have 2 classes and no missing features.

3.1 Analysis of Data Balance

To determine if a dataset is balanced or imbalanced, we use a threshold of 30% for the minority class. If at least 30% of the samples belong to the minority class, we consider the dataset *balanced* ; otherwise, it is *imbalanced*.
Table 1 summarizes the datasets used in our study. According to the criterion, 15 datasets are balanced and 10 are imbalanced.

Dataset	Samples	Features	Min Ratio	Balanced
abalone8	4177	10	13.6%	False
abalone17	4177	10	1.4%	False
autompg	392	7	37.5%	True
australian	690	14	44.5%	True
balance	625	4	46.1%	True
bupa	345	6	42.0%	True
german	1000	24	30.0%	True
glass	214	9	32.7%	True
hayes	132	4	22.7%	False
heart	270	13	44.4%	True
iono	351	34	35.9%	True
libras	360	90	6.7%	False
newthyroid	215	5	30.2%	True
pageblocks	5473	10	10.2%	False
pima	768	8	34.9%	True
segmentation	2310	19	14.3%	False
sonar	208	60	46.6%	True
spambase	4597	57	39.4%	True
splice	3175	60	48.1%	True
vehicle	846	18	23.5%	False
wdbc	569	30	37.3%	True
wine	178	13	33.1%	True
wine4	1599	11	3.3%	False
yeast3	1484	8	11.0%	False
yeast6	1484	8	2.4%	False

TABLE 1 – Résumé des datasets utilisés dans les expériences.

3.2 Strategies for Handling Imbalance

One of the main objectives of our study is to analyze the impact of various methods for addressing imbalanced data. In many real-world classification tasks, one class significantly outnumbers the other. This imbalance can cause standard learning algorithms to become biased toward the majority class, resulting in poor performance on the minority class.
We consider three strategies to handle class imbalance :

Oversampling : Increase the number of instances in the minority class. A simple approach is to replicate existing minority examples until the classes are balanced. In this work, we use a more sophisticated method called SMOTE (Synthetic Minority Oversampling Technique), which generates new synthetic samples based on feature-space similarities between existing minority instances. This helps improve diversity within the minority class.

Undersampling :
Reduce the number of instances in the majority class by randomly removing samples, to balance the class distribution. While this effectively addresses imbalance, it can also discard potentially useful information from the majority class.

Hybrid Sampling : Combine both oversampling and undersampling. For example, the SMOTEENN method first applies SMOTE to generate synthetic minority examples and then uses Edited Nearest Neighbors (ENN) to remove ambiguous or noisy samples from the dataset. This produces a more balanced and cleaner dataset for model training.

We compare these techniques with using the original imbalanced dataset without any sampling. However, comparing models solely based on accuracy is not fair, as a model could achieve high accuracy on imbalanced data simply by always predicting the majority class. To mitigate this issue, we use the F1-score.

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}},$$

with $\text{precision} = \frac{TP}{TP+FP}$, $\text{recall} = \frac{TP}{TP+FN}$

4 Méthodologie

To compare the 3 sampling methods (undersampling, oversampling and hybridsampling) against not rebalancing the data, and to compare the 4 ensemble methods, we perform ; over a set of hyperparameters ; a 4-fold cross-validation over each imbalanced dataset, preprocessed with each sampling methods, and over the balanced dataset ; with each algorithm.

Cross Validation : We perform for each method M a 4-folds cross validation on a set of hyperparameters Θ on 25 datasets separated in 15 imbalaced (treated with 3 samplings methods and without sampling method), and 10 balanced classes. The cross-validation algorithm is reminded in Algorithm 1.

Algorithm 1 Cross-Validation for Model Evaluation

Require: Data D, number of folds I, model M, parameter space $\{\Theta^l, l \in L\}$
Ensure: Optimal parameters θ^* and mean validation score $\bar{s}$
  Partition D into L folds : $D = \{D_1, D_2, \ldots, D_I\}$
  $scores \leftarrow \{\}$
  for $i \leftarrow 1$ **to** I **do** ▷ Cross-validation loop
    Define $D^{(i)}_{\text{train}} \leftarrow D \setminus D_i$ and $D^{(i)}_{\text{val}} \leftarrow D_i$
    Train model M with parameters $\Theta^l, \forall i \in L$ on $D^{(i)}_{\text{train}}$ and evaluate on $D^{(i)}_{\text{val}}$ to obtain score s_i
    $scores \leftarrow scores \cup \{s_i\}$
  end for
  $\bar{s} \leftarrow \frac{1}{I} \sum_{i=1}^{I} s_i$
  return Optimal parameters θ^* and mean score $\bar{s}$

There are multiple ways to perform stacking, and selecting the hyperparameters for base models and the meta-model can be tricky, often leading to overfitting even with classical cross-validation [1]. To avoid overfitting in stacking, we employed a nested cross-validation scheme to tune the hyperparameters of the base learners. It consist to replace the training

step of the cross-validation by a more complex procedure (Algorithm 2) that includes an inner cross-validation for the base models.

Algorithm 2 Train Base Models with Hyperparameter Tuning for Stacking

Require: Train data D_{train}, validation data D_{val}, base models $h_1, \ldots, h_K$ with parameter spaces $\Theta_1, \ldots, \Theta_K$, meta-model g with hyperparameters Θ^i.
Ensure: Optimized base models and prediction on D_{val}

```
// PHASE 1 : Optimize hyperparameters for each base model
for k ← 1 to K do                                          ▷ For each base model
    Divide D_train into L' inner folds
    for each parameter combination θ ∈ Θ_k do
        for j ← 1 to L' do                                 ▷ Inner cross-validation
            Define training and validation splits from inner folds
            Train h_k with parameters θ and evaluate
        end for
        Compute mean score across inner folds
        Update best parameters θ*_k if score improves
    end for
end for
// PHASE 2 : Generate out-of-fold predictions using optimized models
Divide D_train into L' inner folds
for j ← 1 to L' do
    Define inner training and validation splits
    for k ← 1 to K do
        Train h_k with optimal parameters θ*_k
        Generate and store out-of-fold predictions
    end for
end for
// PHASE 3 : Train meta-model on out-of-fold predictions
Train meta-model g using out-of-fold predictions with hyperparameters Θ^i
// PHASE 4 : Generate final predictions
for k ← 1 to K do
    Train h_k with optimal parameters θ*_k on entire D_train
    Generate predictions for D_val
end for
return Meta-model prediction on validation set
```

Models Evaluated and Hyperparameter Optimization : We evaluated a range of classification algorithms, from baseline models to advanced ensemble techniques, as summarized in Table 4. For the two baseline algorithms (Logistic Regression and Decision Tree), we selected four hyperparameter configurations each to ensure a fair comparison. For ensemble methods, we combined each base learner's hyperparameter options with three choices for the number of estimators (50, 100, 150). This resulted in 12 configurations (3×4) for each of bagging and boosting methods. The stacking approach explored 64 configurations, representing the combination of hyperparameters from two base models and the meta-learner ($4 \times 4 \times 4$).
metrics saved during training :

Logistic Regression			**Decision Trees**				**Stacking**
LR	Bagging LR	AdaBoost LR	DT	Rand Forest	AdaB DT	Grad Boost	Stack
Base : $C \in \{0.01, 0.1, 1, 10\}$			Base : $(depth, min\_split) \in \{2, 3\} \otimes \{10, 30\}$				Base : LR $\otimes$ DR
-	$n\_estimators \in \{50, 100, 150\}$		-	$n\_estimators \in \{50, 100, 150\}$			Meta-C $\in \{10^{-i}\}_{i=0}^{4}$
4	12		4	12			64

TABLE 2 – Overview of models and hyperparameter spaces

- **Evaluation results** : F1-score of train and validation for each combination of hyperparameters, with their standard deviations across cross-validation folds.
- **Overfitting/underfitting metrics** : Analysis of model behavior :
  - Gap between training and validation performance for all hyperparameters
  - Detection of underfitting and overfitting [1].
- **Execution time** : Time of compute for all cross-validation.

Implementation and reproductibility Experiments are leaded in python with scikit library. We train all models with a seed to ensure reproductibility.

5 Analysis of Results

In this section, we present a detailed analysis of the experimental results, focusing on the effectiveness of the data rebalancing methods (Section 5.2), the performance of ensemble techniques (Section 5.3), and the impact of hyperparameters on overfitting (Section 5.4).

5.1 First Analyses : Balanced Data

Table 3 presents the mean test F1-scores (over 4 folds) of the best hyperparameter configuration for each of the 8 algorithms across the 15 balanced datasets. We also report the standard deviation of the F1-score across the folds.

We observe that the ensemble methods achieve the best test performance for every balanced dataset. Among the ensemble techniques, those based on logistic regression tend to have the lowest scores, while the single Decision Tree is the weakest of all eight algorithms but seems to benefit the most from ensemble methods. The algorithm that achieved the most top scores is Gradient Boosting, with AdaBoost (using trees) coming in second.

There is a large amount of data from our experiments (in total, we trained 7260 models : 132 hyperparameters tested over 15 balanced datasets and 10 imbalanced with 4 sampling methods.). We do not include the equivalent tables for every sampling method here. Instead, in the following sections, we aggregate results and extract statistics to study

1.

- **Underfitting :** When both training and validation scores are low ($F_1\_train < 0.6$ and $F_1\_val < 0.6$).
- **Overfitting :** When the gap between training and validation scores is high (gap > 0.15).

Dataset	Logistic Regression	Bagging LR	AdaBoost LR	Decision Tree	Random Forest	AdaBoost Tree	Gradient Boosting	Stacking
australian	87.6 ± 0.9	**88.0 ± 0.7**	87.5 ± 0.8	85.2 ± 1.4	87.7 ± 1.0	86.0 ± 1.9	86.7 ± 1.3	87.5 ± 0.3
autompg	88.7 ± 2.7	88.9 ± 2.9	83.2 ± 2.8	86.8 ± 4.3	87.3 ± 4.3	91.1 ± 2.6	**91.8 ± 2.1**	88.6 ± 3.0
balance	95.1 ± 1.4	95.1 ± 1.4	93.6 ± 2.9	74.9 ± 3.3	88.8 ± 5.0	**96.4 ± 1.7**	94.0 ± 1.7	95.1 ± 1.4
bupa	65.8 ± 5.9	66.1 ± 6.0	67.0 ± 8.4	60.7 ± 3.6	71.6 ± 3.7	70.2 ± 4.9	**74.0 ± 4.4**	65.9 ± 3.9
german	70.0 ± 1.5	70.1 ± 1.7	70.1 ± 1.4	64.5 ± 2.3	70.1 ± 2.5	67.4 ± 2.4	**70.8 ± 3.5**	69.5 ± 1.0
glass	70.4 ± 5.9	71.9 ± 5.6	67.2 ± 6.2	78.4 ± 8.4	81.8 ± 10.5	**89.9 ± 6.5**	86.5 ± 6.1	77.9 ± 8.3
heart	83.0 ± 6.4	83.3 ± 7.0	82.9 ± 6.9	77.5 ± 5.4	83.6 ± 6.3	79.3 ± 6.6	81.3 ± 5.5	**83.7 ± 6.7**
iono	86.7 ± 5.4	86.1 ± 5.5	86.0 ± 3.9	88.8 ± 1.3	91.1 ± 2.9	92.5 ± 2.7	**93.3 ± 1.3**	90.4 ± 4.8
newthyroid	88.9 ± 5.0	88.5 ± 4.9	82.8 ± 4.7	88.9 ± 7.4	93.8 ± 3.6	**94.7 ± 3.7**	93.0 ± 2.2	89.5 ± 6.7
pima	73.6 ± 3.4	73.8 ± 2.6	73.1 ± 2.6	70.3 ± 4.4	**75.0 ± 2.2**	73.4 ± 2.8	73.6 ± 1.1	74.2 ± 3.3
sonar	79.0 ± 3.9	79.4 ± 3.4	78.1 ± 5.0	72.8 ± 5.8	79.6 ± 3.3	**87.8 ± 5.5**	84.9 ± 5.4	78.9 ± 6.5
spambase	92.7 ± 0.6	92.7 ± 0.6	91.0 ± 0.8	87.1 ± 0.9	91.8 ± 0.2	92.1 ± 0.6	**94.9 ± 0.5**	93.0 ± 0.1
splice	84.8 ± 0.9	84.8 ± 0.9	84.6 ± 1.0	89.9 ± 0.7	94.3 ± 0.8	97.1 ± 0.2	**97.2 ± 0.6**	91.5 ± 1.4
wdbc	97.9 ± 1.1	97.9 ± 0.7	96.6 ± 2.0	93.0 ± 2.1	94.9 ± 1.4	97.7 ± 1.1	96.4 ± 1.5	**98.5 ± 1.2**
wine	99.4 ± 1.2	99.4 ± 1.2	98.7 ± 2.5	93.7 ± 0.3	99.3 ± 1.3	97.5 ± 1.3	97.4 ± 2.5	**100.0 ± 0.0**

TABLE 3 – F1-Score performance comparison (mean ± standard deviation) from 4-fold cross-validation on 15 balanced datasets. Each value represents the average F1-Score across the four folds, along with the standard deviation indicating the stability of model performance.

the impact of sampling methods, algorithm choices, and hyperparameters on performance.

After this section, each table groups two characteristics—balance method, algorithm, and dataset. If nothing is specified and two characteristics are chosen for the columns and rows, it means that we have averaged the performance of the model selected by cross-validation over the remaining characteristic.

5.2 Effectiveness of Rebalancing Methods

5.2.1 Analysis by Imbalance Level

We grouped the imbalanced datasets into three categories based on the mi-nority class ratio :

- Moderate imbalance (10-30%) : abalone8, pageblocks, segmentation, vehicle, yeast3
- Severe imbalance (3-10%) : libras, wine4
- Extreme imbalance (<3%) : abalone17, yeast6

In Table 4, we show, for each category, the average F1-score achieved by each rebalancing method, the average computation time for one cross-validation run, and the average number of samples after applying the balancing method.
We observe that for the cases of no rebalancing and undersampling, the more imbalanced the dataset, the worse the F1-score. This trend is less clear for oversampling and hybrid sampling, which yield relatively high and stable F1-scores across different imbalance levels. All balancing methods lead to better F1-scores than doing nothing on the imbalanced data. Among them, the hybrid sampling method performs best, followed by oversampling (which in turn outperforms undersampling).
As dataset imbalance becomes more severe, the performance gap between the no-balanced approach and undersampling grows larger, as does the gap between undersampling and

Imbalance Level	imbalanced	undersampling	oversampling	hybridsampling
Moderate Imbalance (10-30%)	0.7949	0.8714	0.8977	**0.9313**
Severe Imbalance (3-10%)	0.6885	0.8131	0.9036	**0.9177**
Extreme Imbalance (<3%)	0.5752	0.7725	0.8970	**0.9248**
Average execution time (s)	20.83	11.21	43.20	19.53
Average sample count	2434	442	4426	3971

TABLE 4 – Average F1-score by imbalance level and sampling method

the oversampling/hybrid methods. This indicates that no balanced approach performances degrades markedly as the imbalance ratio worsens.
In terms of computation, undersampling has the shortest runtime, followed by using the imbalanced data as-is, while oversampling is by far the most time-consuming. This is consistent with the number of samples each method produces (Table 4, bottom rows). None the less, even though the hybrid datasets have about 60% more samples than the imbalanced ones, training on the imbalanced data was about 5% slower. This can be explained by the fact that decision trees and logistic regression do not have a fixed number of iterations. Better class balance and a less noisy dataset (as achieved by SMOTEENN) can reduce intra-class variability and improve the data distribution, allowing these algorithms to converge faster despite the larger dataset.

5.2.2 Aggregate Performance by Algorithm

We calculated the average performance of each rebalancing method for each algorithm individually in table 5.

Hybridsampling is the best sampling method for each algorithm.
We also observe that a single Decision Tree (DT) yields the worst F1-score among all algorithms for every rebalancing method (with one small exception : AdaBoost LR is slightly worse than DT under hybrid and oversampling). For all sampling methods, both bagging and AdaBoost using DT as the base learner achieve better results than their counterparts using LR. This can be explained by the properties of the learners : shallow decision trees are weak learners with high bias, so boosting methods are very effective at improving them by reducing bias. Moreover, DTs are very sensitive to data (small changes in the dataset can lead to large changes in predictions), meaning they have high variance ; thus, they benefit greatly from bagging. In contrast, logistic regression has low variance and relatively low bias, so it gains less from these ensemble techniques.

5.3 Comparison of Ensemble Methods

Tables 6 and 7 present the comparative analysis of the test results gave by the model with the best hyperparameters based on the cross validation score for ensemble methods against their base algorithms (Logistic Regression and Decision Trees).

For each sampling method and balanced data, we display the F1-score of the base algorithm and the relative improvement (or deterioration) in percentage points for each ensemble technique. The numbers in parentheses indicate the percentage of datasets where the ensemble method outperforms the base algorithm. Values in bold represent

Algorithm	imbalanced	undersampling	oversampling	hybridsampling
Logistic Regression	0.7277	0.8435	0.8818	**0.9183**
Bagging LR	0.7361	0.8505	0.8809	**0.9192**
AdaBoost LR	0.7353	0.8258	0.8614	**0.8948**
Decision Tree	0.7009	0.8232	0.8725	**0.9071**
Random Forest	0.7434	0.8537	0.8898	**0.9226**
AdaBoost Tree	0.7846	0.8562	0.9671	**0.9748**
Gradient Boosting	0.7921	0.8581	0.9632	**0.9782**
Stacking	0.7412	0.8657	0.9018	**0.9370**

TABLE 5 – Average F1-score by algorithm and sampling method

the best-performing method for each sampling scenario, while green values indicate the best non-Stacking algorithm.

$$\text{Relative gain} = F1_{\text{ensemble}} - F1_{\text{base}} \tag{1}$$

Sampling	Base LR	Bagging	AdaBoost	Stacking
imbalanced	0.7277	+0.0084 (90%)	+0.0076 (60%)	**+0.0135 (60%)**
undersampling	0.8435	+0.0070 (70%)	-0.0177 (30%)	**+0.0222 (80%)**
oversampling	0.8818	-0.0009 (80%)	-0.0204 (20%)	**+0.0200 (100%)**
hybridsampling	0.9183	+0.0009 (60%)	-0.0235 (0%)	**+0.0187 (100%)**
balanced	0.8424	+0.0017 (60%)	-0.0141 (13%)	**+0.0138 (67%)**

TABLE 6 – Average F1-score and relative improvement for Logistic Regression based methods

Sampling	Base DT	RF	AdaBoost	GB	Stacking
imbalanced	0.7009	+0.0425 (70%)	+0.0837 (80%)	**+0.0912 (80%)**	+0.0403 (70%)
undersampling	0.8232	+0.0305 (90%)	+0.0330 (80%)	+0.0349 (90%)	**+0.0425 (90%)**
oversampling	0.8725	+0.0173 (80%)	**+0.0946 (100%)**	+0.0907 (100%)	+0.0293 (90%)
hybridsampling	0.9071	+0.0155 (70%)	+0.0677 (90%)	**+0.0711 (90%)**	+0.0299 (90%)
balanced	0.8084	+0.0521 (100%)	+0.0671 (100%)	**+0.0686 (100%)**	+0.0478 (93%)

TABLE 7 – Average F1-score and relative improvement for Decision Tree based methods

The results reveal differences between the two base lines. Ensemble methods based on Logistic Regression show modest gains, with some configurations. Ovesampling even degrade performances. With oversampling, the baseline outperforms all its ensemble variants.

Among LR ensembles, Bagging consistently outperforms AdaBoost, indicating that Logistic Regression in these datasets suffers more from variance than bias. Conversely, for Decision Trees is improved by all its ensemble variants. For DT, boosting methods provide substantial improvements by addressing their inherent bias.

This results show how strong learners, like logistic regression, are more affected by the variance and the weak learners, like decision trees with little depth, are more afected by

the bias. Stacking consistently emerges as one of the top performers across most scenarios by leveraging the diversity of multiple base learners mistakes like it is explained in Section 2.

5.4 Hyperparameter Impact Analysis on Model Performance

5.4.1 Analysis of Optimal Hyperparameters

Tables 8 and 9 present for each algorithm across the different sampling methods and balanced dataset the most frequently set of hyperparameters that obtained the best validation score. For each combination of sampling method and algorithm, we display the most common hyperparameter configuration that was selected during cross-validation. Between parentheses is wroten the percentage of datasets where this configuration was chosen as optimal.

Sampling	Logistic Regression	Bagging LR	AdaBoost LR
imbalanced	C=10.0 (60%)	C=10.0, n_estim=100 (30%)	C=10.0, n_estim=50 (40%)
undersampling	C=10.0 (40%)	C=10.0, n_estim=50 (40%)	C=10.0, n_estim=50 (100%)
oversampling	C=10.0 (60%)	C=10.0, n_estim=150 (20%)	C=10.0, n_estim=50 (80%)
hybridsampling	C=10.0 (40%)	C=10.0, n_estim=50 (30%)	C=10.0, n_estim=50 (90%)
balanced	C=10.0 (27%)	C=10.0, n_estim=50 (33%)	C=10.0, n_estim=50 (100%)

TABLE 8 – Most common hyperparameters for Logistic Regression based methods, where C denotes the regularization parameter and n_estim denotes the number of estimators.

Sampling	Decision Tree	Random Forest	AdaBoost Tree	Gradient Boosting
imbalanced	md=3, m_s=10 (90%)	md=3, m_s=10, n_estim=50 (30%)	md=3, m_s=10, n_estim=50 (30%)	md=3, m_s=10, n_estim=150 (30%)
undersampling	md=3, m_s=10 (50%)	md=3, m_s=10, n_estim=50 (30%)	md=3, m_s=30, n_estim=50 (20%)	md=3, m_s=10, n_estim=150 (30%)
oversampling	md=3, m_s=10 (80%)	md=3, m_s=10, n_estim=50 (40%)	md=3, m_s=10, n_estim=150 (40%)	md=3, m_s=30, n_estim=150 (40%)
hybridsampling	md=3, m_s=10 (100%)	md=3, m_s=10, n_estim=100 (30%)	md=3, m_s=10, n_estim=50 (20%)	md=3, m_s=30, n_estim=150 (60%)
balanced	md=3, m_s=10 (80%)	md=3, m_s=10, n_estim=50 (33%)	md=3, m_s=10, n_estim=150 (20%)	md=2, m_s=10, n_estim=150 (20%)

TABLE 9 – Most common hyperparameters for Decision Tree based methods, (md : max depth, m_s : min split, n_estim : the number of estimators.

For the Logistic Regression-based methods (Table 8), we observe that the highest regularization strenght of C=10 obtains best results. This is certainly due to underfitting like discussed in Section 5.4.3, which occur when the regularization is too big (low C hyperparameter). 50 estimators, is the best value of this hyperparameter for adaboost, while bagging shows more variability in the optimal number of estimators depending on the sampling technique.

The Decision Tree-based algorithms (Table 9) show a strong consistency in the selection of tree parameters, with max_depth=3 and min_split=10 being optimal across most algorithms and sampling techniques. This is the most complex setting which belongs to our hyperparameter grid. For gradient boosting methods, larger ensemble sizes (n_estimators=150) are preferred, while Random Forest often performs best with fewer trees (n_estimators=50). While adaboost shows more variabilities.

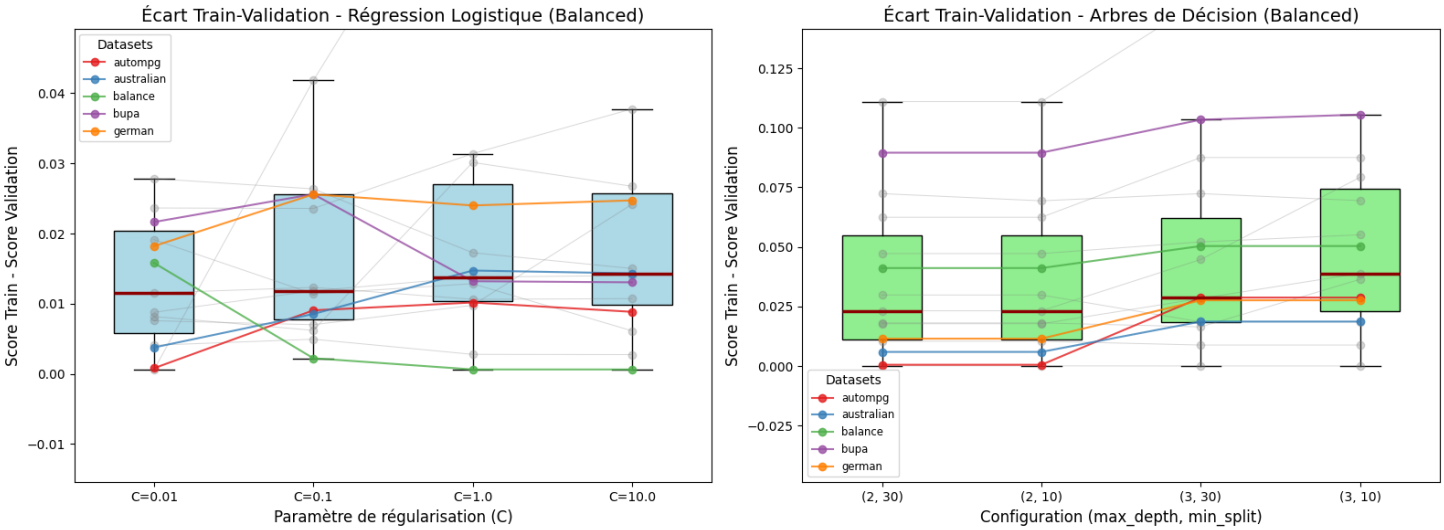

FIGURE 1 – Distribution of train/validation gap according to the hyperparameters

5.4.2 Train/validation gap comparison

For each hyperparameter configuration (regularization parameter C for Logistic Regression models and the combination of max_depth and min_samples_split for Decision Trees), we calculated the gap between training and validation F1-scores across all balanced datasets.

The figures display two main components :

- Boxplots representing the distribution of train-validation gaps across all balanced datasets for each hyperparameter configuration.
- Connected lines showing how individual datasets respond to changes in hyperparameter values.

We highlight five representative datasets (colored lines) to illustrate dataset-specific patterns, while the remaining datasets are shown in light gray for context. The red median line in each boxplot represents the central tendency of gaps for each configuration.

The hyperparameters of LR have been ordered from the highest regularization (smallest C) to the smallest regularization (highest C).
High regularization tend to reduce the overfiting and the gap between train and validation scores. Low regularizaton tend to make the model overfit, then the gap increases.

We ordered the DT hyperparameters from the most restrictive (max_depth=2 and min_split=30) to the less restrictive (max_depth=2 and min_split=30).
Restriction over the set of hypothesis is a prior knowledge we insert in the model and it act on the gap like regularization.

Hyperparameter	Decision Tree	AdaBoost	Random Forest	GradBoost
max_depth=2	0.0%	10.0%	0.0%	16.7%
max_depth=3	6.7%	17.8%	4.5%	30.0%
min_samples_split=10	3.4%	12.2%	3.4%	24.4%
min_samples_split=30	3.4%	15.5%	1.1%	22.2%
n_estimators=50	–	16.7%	1.7%	13.3%
n_estimators=100	–	13.3%	1.7%	26.6%
n_estimators=150	–	11.6%	3.4%	30.0%

TABLE 10 – Percentage of datasets exhibiting overfitting by hyperparameter setting for DT.

5.4.3 Impact of hyperparameters on overfitting and underfitting for DT models

To quantify how specific hyperparameter choices affect model behavior, we analyzed the proportion of models exhibiting overfitting or underfitting across all balanced datasets. The tables 10 and 11 present the percentage of models showing each behavior for various hyperparameter configurations across the DT models. Figure 2 shows the evolution of overfitting and underfitting for LR models according to the value of the hyperparaameter C.

We recall our definitions of problematic model behaviors [2] :

- **Underfitting :** When the training F1-score is below a threshold value ($F_1\_train < 0.7$), indicating the model fails to capture the patterns in the data.
- **Overfitting :** When the gap between training and validation scores exceeds a threshold (gap > 0.15), indicating the model has memorized training data rather than learning generalizable patterns.

In the tables that follow, each cell represents the average percentage of datasets where models with that specific hyperparameter setting (row) and model type (column) exhibited overfitting or underfitting.

Tree depth : Increasing the maximum tree depth from 2 to 3 increases overfitting across all models. This confirms that deeper trees capture more complex patterns. Conversely, shallower trees (max_depth=2) show higher underfitting rates, as they cannot model sufficiently complex relationships.

Model complexity hierarchy : The tables reveal a hierarchy in terms of overfitting tendency : Gradient Boosting > AdaBoost > Random Forest > Decision Tree. This means that, boosting methods are powerfull, complex methods, which excels at capturing patterns in training data but can requires hyperparameter regularization to ensure generalization. Conversly, Random Forest demonstrates resistance to overfitting due to its bagging approach which control variance.

2. Those definitions is based on hard threshold values that we fixed arbitrary. What we want to compare is not the absolute values of underfitting and overfitting but the relative values between algorithm and hyperparameters.

Hyperparameter	Decision Tree	AdaBoost	Random Forest	GradBoost
max_depth=2	13.3%	6.7%	4.5%	0.0%
max_depth=3	6.7%	0.0%	0.0%	0.0%
min_samples_split=10	10.0%	3.4%	2.2%	0.0%
min_samples_split=30	10.0%	3.4%	2.2%	0.0%
n_estimators=50	–	3.4%	3.4%	0.0%
n_estimators=100	–	3.4%	3.4%	0.0%
n_estimators=150	–	3.4%	0.0%	0.0%

TABLE 11 – Percentage of datasets exhibiting underfitting by hyperparameter setting for DT.

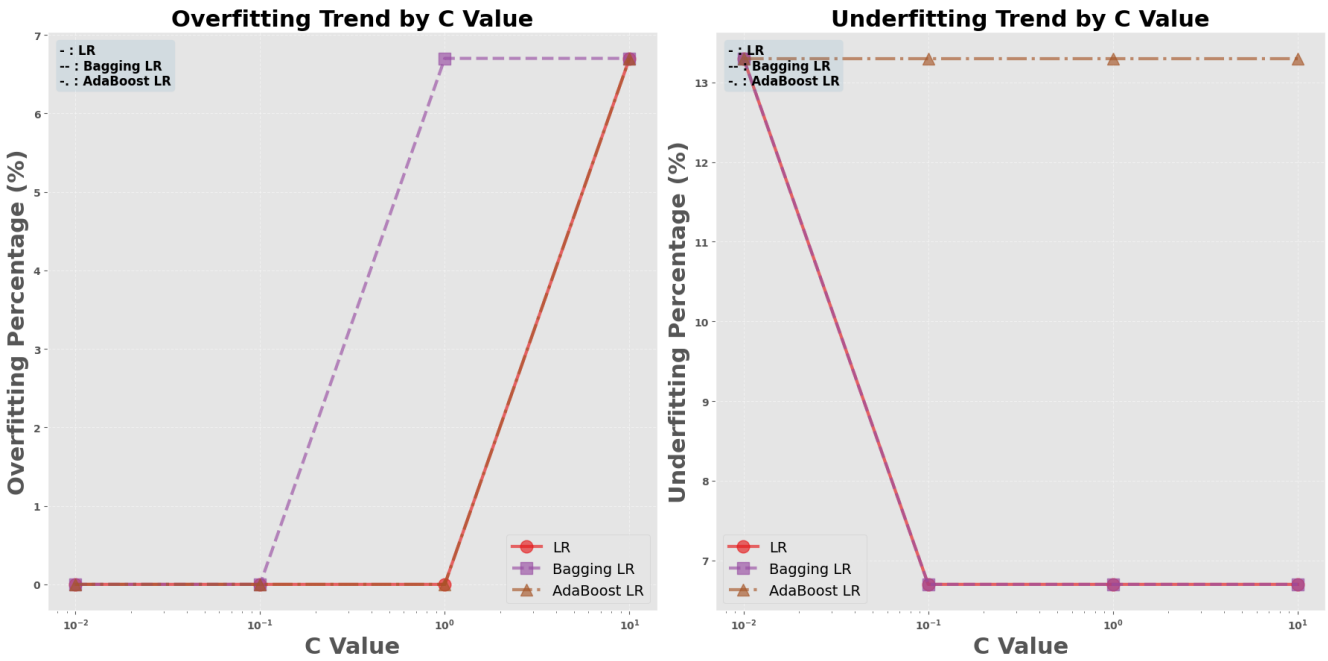

FIGURE 2 – Overfitting and underfitting according to C value

N_estimators : Increasing the number of estimators leads in AdaBoost, overfitting decreases, while in Gradient Boosting and Random Forest, the opposite occurs. This paradoxical behavior of AdaBoost may be attributed to its weighting error mechanism that can have a self-regularizing effect with more estimators while it augments complexity for other methods.

Min_samples_split impact : Increasing min_samples_split from 10 to 30 shows little impact on Decision Trees and slightly reduces overfitting in Random Forest. This parameter is less influential than tree depth for controlling model bias-variance trade-off.

Impact of C hyperparameter on LR based models : The Figure 2 shows that for high regularization (low C), LR based models have tendency to underfit, which leads to a high bias error and for low regularization (high C), it overfit which lead to high variance

error.

6 Conclusion

In conclusion, our comparative analysis of ensemble methods for binary classification has shown that ensemble techniques can substantially improve predictive performance, especially when appropriate data balancing methods are applied. We summarize our key findings and practical recommendations below :

Choice of algorithm : The choice of base model highly influence ensemble performance. We found that every DT ensemble methods, consistently outperformed their base algorithm across most of datasets. While LR ensemble methods lead to low improvment or even degradation of the score. Gradient Boosting and AdaBoost with trees emerged as the top performers in most scenarios, while stacking consistently delivered strong results by effectively leveraging the diversity of multiple base learners. Then We recommand to use DT as base of ensemble method.

Choice of rebalance method : For addressing class imbalance, all rebalancing methods improved performance compared to using imbalanced data directly. Hybrid sampling (combining SMOTE and ENN) delivered the best results across all algorithms and imbalance levels, followed by oversampling, which consistently outperformed undersampling. The performance gap between different sampling methods become bigger as dataset imbalance increases. While undersampling had the lowest execution time, hybrid sampling demonstrated a similar computational efficiency as imbalanced setting due to a better quality of the dataset. We recommand this method to deal with imbalanced dataset.

Choice of hyperparameters : Our hyperparameter analysis revealed distinct patterns across different algorithm families. For Logistic Regression based methods, higher regularization strength (C=10.0) yielded optimal results, suggesting these models tended to underfitting on our datasets. For tree-based methods, max_depth=3 and min_samples_split=10 (the most complex settings in our grid) generally performed best. We observed the overfitting tendency of each method : Gradient Boosting > AdaBoost > Random Forest > Decision Tree, confirming that more powerful ensemble methods requires careful selection of hyperparameters that control the model complexity and create a balance between bias-variance errors. We recommand to estimate the complexity of the problem and more the data manifold is simple more we can use restricted hyperparameters to regularize the model.

Références

[1] Douglas M. Hawkins. "The Problem of Overfitting". In : *Journal of Chemical Information and Computer Sciences* 44.1 (2004). PMID : 14741005, p. 1-12. doi : 10.1021/ci0342472. eprint : https://doi.org/10.1021/ci0342472. url : https://doi.org/10.1021/ci0342472.

[2] David H. Wolpert. “Stacked generalization”. In : *Neural Networks* 5.2 (1992), p. 241-259. ISSN : 0893-6080. DOI : https://doi.org/10.1016/S0893-6080(05)80023-1. URL : https://www.sciencedirect.com/science/article/pii/S0893608005800231.