Étude Comparative Des Méthodes D'Ensemble Et Des Techniques De Rééquilibrage Pour La Classification De Données Déséquilibrées

---

1. Introduction

Dans le domaine de l'apprentissage automatique, la classification de données déséquilibrées reste un défi majeur. Ce déséquilibre, caractérisé par une représentation inégale des classes dans un ensemble de données, peut significativement affecter les performances des algorithmes de classification standard. Notre étude se concentre sur l'évaluation et la comparaison de diverses méthodes d'ensemble et non-ensemble, ainsi que sur l'application de techniques de rééquilibrage, pour aborder ce problème.

Nous avons analysé 28 jeux de données présentant divers degrés de déséquilibre, allant de datasets relativement équilibrés à des cas extrêmes où une classe représente moins de 0.1% des échantillons. Les principales difficultés rencontrées incluent :

- La variabilité des datasets bruts: Richesse en nombre d'observations (de 132 à 45 211 instances), en nombre de caractéristiques (de 5 à 91 variables) et en travail exploratoire nécéssaire à leur preprocessing.
- La multiplicité d'approches possibles de l'exercice: chaque étape offrait une multitude de méthodes possibles pour la famille d'algorithme et les ajustements potentiels.
- Choix d'implémentation: La librairie scikit-learn a été retenue pour l'essentiel de l'implémentation , mais ça n'a pas été un choix évident. En effet sklearn n'adapte pas nativement les calcul sur le GPU. Il existe certes une possiblité de requisitionner plusieurs coeurs du CPU avec le paramètre "n_jobs", mais ça reste insuffissant pour des tâches gourmandes en ressources et ça peut rapidement bloquer l'expérimentation. Des tests avec d'autres librairies n'ont pas été concluants car les méthodes manquaient de compatibilité et reduisaient la lisibilité du code.

Notre objectif est d'évaluer l'efficacité de différentes approches pour améliorer la classification sur ces datasets déséquilibrés, en mettant l'accent sur les méthodes d'ensemble et les techniques de rééquilibrage.

2. Méthodologie

2.1 Notations

Avant de présenter les méthodes d'ensemble, définissons les notations utilisées :

- D : Ensemble de données complet (dataset)
- D_i : Sous-ensemble (échantillon) du dataset D
- X : Matrice des caractéristiques, où $X_{i,j}$ représente la j-ème caractéristique du i-ème échantillon
- y : Vecteur des étiquettes de classe, où $y_i \in \{0, 1\}$
- $h(X)$: Fonction de prédiction d'un classifieur
- w_i : Poids attribué à l'échantillon i
- N : Nombre total d'échantillons
- N_+ : Nombre d'échantillons de la classe minoritaire (positive)
- N_- : Nombre d'échantillons de la classe majoritaire (négative)

2.2 Mesures de performance

Notre étude se concentre principalement sur l'optimisation du rappel (recall) et de l'aire sous la courbe précision-rappel (AUC-PR), particulièrement pertinentes pour les problèmes de classification déséquilibrée. Ces métriques sont choisies pour leur capacité à évaluer efficacement les performances sur la classe minoritaire, souvent la plus importante dans les problèmes déséquilibrés.

• **Matrice de confusion** : Avant de définir nos métriques, rappelons les éléments de la matrice de confusion :

- TP (True Positives) : Vrais Positifs
- TN (True Negatives) : Vrais Négatifs
- FP (False Positives) : Faux Positifs
- FN (False Negatives) : Faux Négatifs

• **Rappel (Recall)** :

$$\text{Rappel} = \frac{TP}{TP + FN}$$

Le rappel mesure la proportion d'instances positives correctement identifiées. Dans un contexte déséquilibré, un rappel élevé indique une bonne détection de la classe minoritaire.

• **Précision** :

$$\text{Précision} = \frac{TP}{TP + FP}$$

La précision mesure la proportion de prédictions positives qui sont correctes. Elle est importante pour évaluer la fiabilité des prédictions positives.

• **Ratio Précision/Rappel** : Le compromis entre précision et rappel est crucial dans les ensembles déséquilibrés :

- Un rappel élevé avec une faible précision indique de nombreux faux positifs.
- Une précision élevée avec un faible rappel suggère de nombreux faux négatifs.
- L'objectif est souvent de maximiser le rappel tout en maintenant une précision acceptable.

• **Courbe Précision-Rappel** :

1. Calculer précision et rappel pour différents seuils de classification.
2. Tracer la précision en fonction du rappel.
3. La courbe illustre le compromis entre précision et rappel à mesure que le seuil de classification varie.

• **AUC-PR** (Aire sous la courbe Précision-Rappel) :

$$AUC - PR = \int_0^1 p(r)dr$$

où p est la précision et r est le rappel.

Calcul pratique pour un ensemble fini de points $(r_1, p_1), \ldots, (r_n, p_n)$ ordonnés par rappel croissant :

$$AUC - PR \approx \sum_{i=1}^{n-1} (r_{i+1} - r_i) \frac{p_i + p_{i+1}}{2}$$

Étapes de calcul :

1. Trier les prédictions par score décroissant.
2. Calculer précision et rappel pour chaque seuil.
3. Calculer l'aire sous la courbe en utilisant la méthode des trapèzes.

• **Importance pour les ensembles déséquilibrés** :

- L'AUC-PR est préférée à l'AUC-ROC car elle est plus sensible aux performances sur la classe minoritaire.
- Elle n'est pas affectée par le grand nombre de vrais négatifs typiques des problèmes déséquilibrés.
- Elle permet une évaluation plus nuancée des performances, particulièrement lorsque la détection précise de la classe minoritaire est cruciale.

L'utilisation combinée du rappel et de l'AUC-PR nous permet d'optimiser nos modèles pour une détection efficace de la classe minoritaire tout en maintenant un équilibre avec la précision globale, ce qui est essentiel dans de nombreuses applications réelles impliquant des données déséquilibrées.

2.3 Algorithmes et techniques étudiés

2.3.1 Méthodes de base

Dans notre étude, nous avons examiné plusieurs méthodes de classification fondamentales, chacune apportant ses propres forces à la tâche de classification des données déséquilibrées :

• **Régression logistique**

- Modèle linéaire simple mais puissant
- Efficace pour comprendre l'impact de chaque caractéristique
- Fournit des probabilités facilement interprétables

• **SVM linéaire**

- Cherche l'hyperplan optimal séparant les classes
- Performant dans les espaces à haute dimension
- Robuste face au surapprentissage dans de nombreux cas

• **Arbre de décision**

- Modèle intuitif basé sur une série de décisions
- Capable de capturer des relations non linéaires
- Facile à interpréter et à visualiser

• **k-plus proches voisins (k-NN)**

- Méthode non paramétrique basée sur la proximité
- Simple à mettre en œuvre et à comprendre
- Efficace pour les frontières de décision complexes

• **Perceptron multicouche (MLP)**

- Réseau de neurones capable d'apprendre des patterns complexes
- Peut modéliser des relations hautement non linéaires
- Adaptable à une grande variété de problèmes

Ces méthodes de base constituent le fondement de notre analyse, nous permettant d'établir des points de comparaison solides pour évaluer l'efficacité des techniques plus avancées sur nos ensembles de données déséquilibrées.

2.3.2 Méthodes d'ensemble

Les méthodes d'ensemble, combinant plusieurs modèles pour améliorer les performances globales, sont théoriquement efficaces pour traiter les problèmes de classification déséquilibrée. Voici les techniques que nous avons vues en cours et que nous avons explorées, chacune avec un pseudocode simplifié de son fonctionnement :

• **Bagging**

- Crée plusieurs sous-ensembles de données par échantillonnage avec remplacement
- Réduit la variance et aide à prévenir le surapprentissage
- Particulièrement utile pour stabiliser des modèles instables comme les arbres de décision

$$\begin{aligned}&\text{Pour } i = 1 \text{ à } K :\\&\quad D_i = \text{ÉchantillonAvecRemplacement}(D)\\&\quad M_i = \text{EntraînerModèle}(D_i)\\&\text{Prédiction}(x) = \frac{1}{K}\sum_{i=1}^{K} M_i(x) \text{ ou } \text{VoteMajoritaire}(\{M_i(x)\}_{i=1}^{K})\end{aligned}$$

• **Random Forest**

- Ensemble d'arbres de décision avec sélection aléatoire de caractéristiques
- Offre un bon équilibre entre biais et variance
- Robuste face au bruit et aux valeurs aberrantes

$$\begin{aligned}&\text{Pour } i = 1 \text{ à } K :\\&\quad D_i = \text{ÉchantillonAvecRemplacement}(D)\\&\quad F_i = \text{SélectionAléatoireCaractéristiques}(F, m)\\&\quad T_i = \text{EntraînerArbreDécision}(D_i, F_i)\\&\text{Prédiction}(x) = \text{VoteMajoritaire}(\{T_i(x)\}_{i=1}^{K})\end{aligned}$$

• **Stacking**

- Combine les prédictions de plusieurs modèles via un méta-apprenant
- Permet d'exploiter les forces de différents types de modèles
- Peut capturer des relations complexes en combinants les apprenants de base

$$\begin{aligned}&M_1, M_2, \ldots, M_K = \text{EntraînerModèlesBase}(D_{\text{train}})\\&\text{Pour } x_i \in D_{\text{validation}} :\\&\quad z_i = [M_1(x_i), M_2(x_i), \ldots, M_K(x_i)]\\&M_F = \text{EntraînerMétaModèle}(\{(z_i, y_i)\}_{i=1}^{N})\\&\text{Prédiction}(x) = M_F([M_1(x), M_2(x), \ldots, M_K(x)])\end{aligned}$$

• **AdaBoost**

- Focalise l'apprentissage sur les exemples difficiles à classer
- Attribue des poids plus élevés aux classifieurs performants
- Efficace pour améliorer les performances des classifieurs faibles

AdaBoost (Adaptive Boosting) est un algorithme d'ensemble qui illustre bien le principe d'apprentissage sensible au coût. Il ajuste itérativement les poids des exemples d'entraînement en fonction de leur difficulté de classification. Voici l'algorithme détaillé :

$$
\begin{aligned}
&w_i^{(1)} = \frac{1}{n}, \quad i = 1, \ldots, n \\
&\text{Pour } t = 1 \text{ à } T : \\
&\quad h_t = \text{EntraînerClassifieurFaible}(D, w^{(t)}) \\
&\quad \epsilon_t = \sum_{i=1}^{n} w_i^{(t)} 1[h_t(x_i) \neq y_i] \\
&\quad \alpha_t = \frac{1}{2} \ln\left(\frac{1 - \epsilon_t}{\epsilon_t}\right) \\
&\quad w_i^{(t+1)} = w_i^{(t)} \exp(-\alpha_t y_i h_t(x_i)) \\
&\quad w^{(t+1)} = \frac{w^{(t+1)}}{\sum_{i=1}^{n} w_i^{(t+1)}} \\
&\text{Prédiction}(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)
\end{aligned}
$$

Avec :

- T : Nombre total d'itérations (nombre de classifieurs faibles)
- $w_i^{(t)}$: Poids de l'échantillon i à l'itération t
- h_t : Classifieur faible à l'itération t
- ϵ_t : Erreur pondérée du classifieur faible h_t
- α_t : Coefficient d'importance du classifieur faible h_t
- x_i : Vecteur de caractéristiques de l'échantillon i
- y_i : Étiquette réelle de l'échantillon i (généralement $\{-1, +1\}$ pour AdaBoost)
- $1[.]$: Fonction indicatrice (1 si la condition est vraie, 0 sinon)

• **Gradient Boosting**

- Construit des modèles séquentiellement pour corriger les erreurs des précédents
- Réputé efficace dans la communauté
- applicable différents types de données et de problèmes

L'algorithme du Gradient Boosting peut être décrit mathématiquement comme suit :

$$
\begin{aligned}
&F_0(X) = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, \gamma) \\
&\text{Pour } m = 1 \text{ à } M : \\
&\quad r_{im} = -\left[\frac{\partial L(y_i, F(X_i))}{\partial F(X_i)}\right]_{F=F_{m-1}} \\
&\quad h_m = \arg\min_{h} \sum_{i=1}^{N} (r_{im} - h(X_i))^2 \\
&\quad \gamma_m = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, F_{m-1}(X_i) + \gamma h_m(X_i)) \\
&\quad F_m(X) = F_{m-1}(X) + \eta \gamma_m h_m(X) \\
&\text{Prédiction}(X) = F_M(X)
\end{aligned}
$$

Avec :

- M : Le nombre total d'itérations (arbres)
- $F_m(X)$: Le modèle à l'itération m
- $L(y_i, F(X_i))$: La fonction de perte qui mesure l'écart entre la prédiction $F(X_i)$ et la vraie valeur y_i
- r_{im} : Le résidu (gradient négatif de la fonction de perte) pour l'échantillon i à l'itération m
- $h_m(X)$: Le modèle faible (généralement un arbre de décision) à l'itération m
- γ_m : Le coefficient de pondération optimal pour le modèle faible h_m
- η : Le taux d'apprentissage (learning rate)

• **XGBoost**

- Implémentation optimisée du gradient boosting
- Réputé offrir une excellente performance et vitesse selon la littérature
- Intègre des techniques de régularisation pour prévenir le surapprentissage

L'algorithme XGBoost peut être décrit mathématiquement comme suit :

$$
\begin{aligned}
&F_0(X) = 0 \\
&\text{Pour } m = 1 \text{ à } M : \\
&\quad g_i = \left[\frac{\partial L(y_i, F(X_i))}{\partial F(X_i)} \right]_{F=F_{m-1}} \\
&\quad h_i = \left[\frac{\partial^2 L(y_i, F(X_i))}{\partial F(X_i)^2} \right]_{F=F_{m-1}} \\
&\quad T_m = \arg\max_T \left[\frac{1}{2} \sum_{j=1}^{T} \frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda} - \gamma T \right] \\
&\quad F_m(X) = F_{m-1}(X) + \eta T_m(X) \\
&\text{Prédiction}(X) = F_M(X)
\end{aligned}
$$

Avec :

- M : Le nombre total d'itérations (arbres)
- $F_m(X)$: Le modèle à l'itération m
- $L(y_i, F(X_i))$: La fonction de perte qui mesure l'écart entre la prédiction $F(X_i)$ et la vraie valeur y_i
- g_i : Le gradient de premier ordre de la fonction de perte pour l'échantillon i
- h_i : Le gradient de second ordre (Hessien) de la fonction de perte pour l'échantillon i
- T_m : L'arbre de décision construit à l'itération m
- I_j : L'ensemble des indices des échantillons dans la feuille j de l'arbre
- λ : Le paramètre de régularisation L2 sur les poids des feuilles
- γ : Le paramètre de régularisation sur la complexité de l'arbre (nombre de feuilles)
- η : Le taux d'apprentissage (learning rate)

Nous allons tester ces différentes méthodes d'ensemble pour nous permettre d'observer expérimentalement leurs qualités certaines sur le plan mathématique et théorique. Elles sont censées nous offrir une meilleure généralisation et une robustesse face aux déséquilibres de classes. Leur capacité à combiner différentes perspectives d'apprentissage les rend candidates intérressantes à notre problématique de classification de données déséquilibrées.

2.3.3 Techniques de rééquilibrage

Dans cette quête d'extraire le maximum d'information de données déséquilibrées, nous avons exploré deux approches principales pour gérer le déséquilibre des classes : l'apprentissage sensible au coût et le rééchantillonnage avec SMOTE.

A. Apprentissage sensible au coût
L'apprentissage sensible au coût est une approche qui modifie le processus d'apprentissage pour prendre en compte le déséquilibre des classes sans altérer la distribution des données d'entrée.

Principe : Cette méthode attribue des poids aux classes inversement proportionnels à leur fréquence dans les données d'entraînement. Ainsi, les erreurs sur les classes minoritaires sont plus pénalisées que celles sur les classes majoritaires.

Exemple simple de Formule de calcul des poids :

$$w_j = \frac{n}{kn_j}$$

où :

- w_j est le poids attribué à la classe j
- n est le nombre total d'échantillons
- k est le nombre total de classes
- n_j est le nombre d'échantillons de la classe j

Fonctionnement :

1. Les poids sont calculés pour chaque classe selon la formule ci-dessus.
2. Ces poids sont intégrés dans la fonction de perte de l'algorithme d'apprentissage.
3. Lors de l'entraînement, les erreurs sur les classes minoritaires ont un impact plus important sur la fonction de perte, poussant le modèle à y accorder plus d'attention.

Avantages :

- Ne modifie pas la distribution des données d'entrée
- Peut être appliqué à de nombreux algorithmes de classification
- Particulièrement utile lorsque le coût d'une mauvaise classification varie selon les classes

Implémentation : De nombreux algorithmes de scikit-learn intègrent cette approche via des paramètres spécifiques. Voici un exemple avec RandomForestClassifier, ou l'on calcule nous même les poids à attribuer aux classes avant de les donner en paramètre au classifieur :

```
from sklearn.ensemble import RandomForestClassifier

# Calcul manuel des poids
class_weights = compute_class_weight('balanced', classes=np.unique(y),
y=y)
class_weight_dict = dict(zip(np.unique(y), class_weights))

# Création et entraînement du modèle avec les poids calculés
rf_cost_sensitive =
RandomForestClassifier(class_weight=class_weight_dict, random_state=42)
rf_cost_sensitive.fit(X, y)
```

B. SMOTE (Synthetic Minority Over-sampling Technique)

SMOTE est une technique de sur-échantillonnage qui génère des exemples synthétiques de la classe minoritaire pour équilibrer le jeu de données.

Principe : SMOTE crée de nouveaux exemples de la classe minoritaire en interpolant entre des exemples existants et leurs plus proches voisins.

Algorithme :

1. Pour chaque exemple de la classe minoritaire : a. Trouver ses k plus proches voisins (typiquement k=5) b. Sélectionner aléatoirement l'un de ces voisins c. Créer un nouvel exemple synthétique par interpolation linéaire

Formule de génération :

$$x_{\text{new}} = x_i + \lambda(x_{\text{neighbor}} - x_i)$$

où :

- x_i est l'exemple de la classe minoritaire
- x_{neighbor} est l'un de ses k plus proches voisins
- $\lambda \in [0, 1]$ est un nombre aléatoire

Avantages :

- Augmente la représentation de la classe minoritaire sans simple duplication
- Peut améliorer la généralisation du modèle
- Utile lorsque la collecte de nouvelles données réelles est difficile ou coûteuse

Implémentation : Nous avons utilisé l'implémentation de SMOTE fournie par la bibliothèque imbalanced-learn, en l'intégrant bien sûr à un pipeline :

```
# Exemple simplifié d'utilisation de la technique smote en python
from imblearn.over_sampling import SMOTE
```

```
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train,
y_train)
```

Ces deux techniques offrent des approches complémentaires pour traiter le déséquilibre des classes. L'apprentissage sensible au coût modifie le processus d'apprentissage lui-même, tandis que SMOTE modifie la distribution des données d'entrée. Le choix entre ces méthodes, ou leur combinaison, dépend des caractéristiques spécifiques du jeu de données et des exigences du problème de classification.

2.4 Approche proposée

Notre approche consiste à appliquer systématiquement ces méthodes sur l'ensemble des 28 datasets, en utilisant trois configurations principales:

- Méthodes de base et d'ensemble sans rééquilibrage
- Méthodes avec application de SMOTE
- Méthodes avec pondération des classes

Nous choisirons par la suite deux méthodes qui monterent de bonnes capacités et on essayera de pousser leur ajustement pour avoir le meilleur résultat possible.

Pour chaque configuration, nous utilisons la validation croisée stratifiée pour optimiser les hyper-paramètres clés de chaque algorithme. Voici un pseudo-code simplifié de notre approche :

```
Pour chaque dataset D dans l'ensemble des datasets:
    Diviser D en ensembles d'entraînement (80%) et de test (20%)
    Pour chaque méthode M (basique ou ensembliste):
        Pour chaque configuration C (sans rééquilibrage, SMOTE,
pondération):
            Appliquer C à l'ensemble d'entraînement
            Optimiser les hyper-paramètres de M par validation
croisée
            Entraîner M sur l'ensemble d'entraînement complet
            Évaluer M sur l'ensemble de test
            Enregistrer les performances (rappel, AUC-PR, etc.)
```

Cette approche nous permet d'évaluer systématiquement l'efficacité de chaque méthode et technique de rééquilibrage sur une variété de datasets présentant différents degrés de déséquilibre.

Points clefs de l'implémentation :

- **Validation croisée** : Nous utilisons la validation croisée à 5 plis, stratifiée, pour garder les mêmes proportions de classe dans chaque fold.
- **Pipeline scaler-classifier** : permettant la normalisation de chaque fold séparément lors de la cross validation, et une normalisation de l'ensemble lors de la phase test finale.

- **Métriques d'évaluation** : Outre le rappel et l'AUC-PR, nous calculons également toutes les métriques dérivées de la matrice de confusion , ainsi qu'une estimation du coût computationnel en temps de calcul et en mémoire.
- **Comparativité** : Pour assurer la robustesse des comparaisons, un nombre d'itéartions de 100 est fixé pour les algorithmes itératifs. Pour les autres, nous avons essayé de garder une équivalence de complexité du modèle par rapport à d'autres hyper-paramètres, comme la profondeur maximale d'un arbre, ou le nombre d'estimateurs dans un bagging.

3. Protocole expérimental

3.1 Données

Notre étude s'appuie sur un ensemble diversifié de 28 jeux de données, provenant de différents domaines et présentant divers niveaux de déséquilibre. Ces datasets sont issus de vrais données de sources reconnues dans le domaine de l'apprentissage automatique. Certains datasets sont des dérivés synthétiques des originaux, à des fins académiques. Voici un aperçu de quelques-uns de ces datasets après preprocessing, illustrant la variété des problèmes abordés :

Nom du dataset	Taille d'échantillon	Nombre de caractéristiques	Ratio de classe minoritaire	Domaine
Hayes	69	3	43.48%	Sociologie (statut marital)
Sonar	208	60	46.63%	Protection civile (détection de mines)
Iono	350	33	35.71%	Physique (qualité de retour radar)
Spambase	4206	57	39.90%	NLP (détection de spams)
Yeast3	1453	8	11.15%	Biochimie (localisation cellulaire de protéines)
Wine	178	13	33.10%	Agronomie (terroir du vin)
Libras	330	85	7.27%	Hand Talk (reconnaissance du geste)

3.2 Prétraitement des données

La qualité et la pertinence des données sont essentielles pour la validité de notre étude. Nous avons donc implémenté un protocole de prétraitement rigoureux et adaptable, tenant compte des particularités de chaque jeu de données. Les étapes principales de notre méthodologie sont les suivantes :

- **Détection et suppression des doublons** : Nous avons effectué une analyse systématique pour identifier et éliminer les observations dupliquées dans chaque jeu de données.

- **Élimination des variables non informatives** : Les colonnes présentant une variance nulle ou quasi-nulle ont été identifiées et supprimées. Ces variables, n'apportant aucune information discriminante, risquaient de diminuer l'efficacité de nos modèles de classification. Mathématiquement, soit $X = [X_1, X_2, \ldots, X_p]$ notre dataset de features avec p variables et N observations. Pour chaque variable X_j où $j \in \{1, \ldots, p\}$, nous avons calculé la variance :

  $\mathrm{Var}(X_j) = \frac{1}{N-1} \sum_{i=1}^{N} (x_{ij} - \bar{x_j})^2$

  où x_{ij} est la i-ème observation de la j-ème variable, et $\bar{x_j}$ est la moyenne de X_j. Les variables avec $\mathrm{Var}(X_j) \approx 0$ ont été éliminées.

- **Réduction de la multicolinéarité** : Nous avons détecté les colonnes hautement corrélées et conservé un seul représentant de chaque groupe. Cette approche vise à réduire la complexité du modèle tout en préservant l'information pertinente. Mathématiquement, pour chaque paire de variables (X_j, X_k) où $j, k \in \{1, \ldots, p\}$ et $j \neq k$, nous avons calculé le coefficient de corrélation de Pearson :

  $r_{jk} = \frac{\sum_{i=1}^{N} (x_{ij} - \bar{x_j})(x_{ik} - \bar{x_k})}{\sqrt{\sum_{i=1}^{N} (x_{ij} - \bar{x_j})^2 \sum_{i=1}^{N} (x_{ik} - \bar{x_k})^2}}$

  où x_{ij} et x_{ik} sont respectivement les i-èmes observations des variables X_j et X_k, et $\bar{x_j}$ et $\bar{x_k}$ sont leurs moyennes respectives. Lorsque $|r_{jk}| > 0.999$, nous avons conservé seulement l'une des deux variables.

- **Normalisation des données** : Afin d'assurer la comparabilité des variables, nous avons appliqué une normalisation via RobustScaler. Cette méthode a été choisie pour sa robustesse face aux valeurs extrêmes fréquentes dans notre cas. Pour chaque feature X_j, la transformation est définie comme suit :

  $X_{j,scaled} = \frac{X_j - Q_1(X_j)}{Q_3(X_j) - Q_1(X_j)}$

  où $Q_1(X_j)$ et $Q_3(X_j)$ sont respectivement le premier et le troisième quartile de X_j.

- **Traitement des outliers** : Nous avons opté pour la conservation des valeurs aberrantes. Ce choix est motivé par leur fréquence élevée qui aurait demandé un temps disproportionné à les étudier au cas par cas. Notons que dans un contexte de déséquilibre des classes, ces outliers peuvent représenter un potentiel d'information pertinente dans la classification.

3.3 Configuration d'ensemble

Pour garantir la rigueur et la reproductibilité de notre étude, nous avons établi un protocole expérimental aussi robuste que possible. Les composantes clés de notre configuration sont les suivantes :

- **Partitionnement des données** :

  Nous avons alloué 80% des données à l'ensemble d'entraînement et 20% à l'ensemble de test. Cette répartition a été effectuée de manière stratifiée pour préserver la distribution des classes dans chaque sous-ensemble.

- **Validation croisée** :

  Une validation croisée à 5 plis, également stratifiée, a été implémentée pour ajuster entre 1 et plusieurs hyperparamètres pertinents pour l'algorithme appliqué.

- **Métriques d'évaluation** :

  Un ensemble complet de métriques a été utilisé pour évaluer les performances des modèles :

  - Rappel : Mesure de la capacité à identifier correctement la classe minoritaire.
  - Précision : Évaluation de la fiabilité des prédictions positives.
  - F1-score : Moyenne harmonique du rappel et de la précision.
  - AUC-PR : Évaluation du compromis précision-rappel, particulièrement pertinente pour nos jeux de données fortement déséquilibrés.
  - Autres mesures issues de la matrice de confusion comme l'accuracy et l'AUC-ROC.
  - Mesure du temps d'execution de la phase "fit", c'est à dire l'entrainement pour chaque dataset.

  Une mesure indicative de l'utilisation de la ram a été également estimée. Il s'agissait de comparer la taille de mémoire utilisée avant, et juste à la fin d'un entrainement.

- **Techniques de rééquilibrage** :

  Après un premier lancer simple des différents algorithmes (basiques et ensemblistes) sur les datasets nettoyés et sans équilibrage des classes, deux autres lancers succéderont :

  - Le deuxième appliquera les mêmes algorithmes sur des datsets équilibrés par oversampling SMOTE.
  - Le troisième appliquera les mêmes algorithmes sur des datasets non équilibrés, mais en pondérant le poids des erreurs en fonction de la prépondérance de la classe. Ceci se fait via des paramètres comme "class_weight='balanced'"

Un dernier Lancer choisira deux méthodes parmis celles testées, pour maximiser leurs performances en combinant diverses approches.

3.4 Configuration des hyperparamètres

Les plages d'hyperparamètres suivantes ont été servies à la validation croisée pour qu'elle ajuste la valeur adéquate en fonction de la métrique du rappel:

- **Lancer sans rééquilibrage** :

  a) Modèles de base :

  - Régression logistique et SVM linéaire : C ∈ {0.001, 0.01, 0.1, 1, 10}
  - Arbre de décision : max_depth ∈ {3, 5, 10, 20, None}
  - k-NN : n_neighbors ∈ {3, 5, 7, 9, 11}
  - MLP : alpha ∈ {10, 1, 0.1, 0.01, 0.001}

  b) Modèles ensemblistes :

  - Bagging, Random Forest, Gradient Boosting : n_estimators ∈ {10, 20, 50, 100, 200}
  - AdaBoost : n_estimators ∈ {10, 20, 50, 100, 200}, learning_rate ∈ {0.01, 0.1, 0.5, 1.0}
  - Stacking : C du méta-classificateur ∈ {0.001, 0.01, 0.1, 1, 10}
- **Lancer avec SMOTE** :

  Les mêmes plages d'hyper-paramètres que pour le lancer sans rééquilibrage ont été utilisées, mais cette fois en appliquant SMOTE aux données d'entraînement.

- **Lancer avec pondération des classes** :

  a) Modèles de base :

  - Mêmes plages que précédemment, mais avec class_weight='balanced'

  b) Modèles ensemblistes :

  - Mêmes plages que précédemment, avec class_weight='balanced' pour les estimateurs compatibles
- **Lancer avec modèles avancés** :

  a) Random Forest avec pondération et multi-fine-tuné :

  - n_estimators ∈ {200, 300, 500}
  - max_depth ∈ {20, None}
  - min_samples_split ∈ {2, 5}
  - min_samples_leaf ∈ {1, 2}
  - max_features ∈ {'sqrt', 'log2'}
  - criterion ∈ {'gini', 'entropy'}

  b) XGBoost avec pondération et multi-fine-tuné:

  - n_estimators ∈ {100, 200, 500}
  - learning_rate ∈ {0.01, 0.1, 0.2}
  - max_depth ∈ {3, 5, 10}

- gamma ∈ {0, 0.1, 0.3}
- sub_sample ∈ {0.8, 1.0}

Cette approche systématique permet de partir d'une base de comparaison commune avant d'appliquer des techniques plus avancées, tout en respectant l'ordre de comparabilité dû aux hyperparamètres.

4. Résultats

4.1 Vue d'ensemble des performances

Notre étude comparative des différentes méthodes de classification sur 28 jeux de données déséquilibrés a révélé des tendances significatives en termes de performance. L'analyse s'est concentrée sur des métrique pertinentes pour des données déséquilibrées : Le F1_score, le rappel et l'AUC-PR (Area Under the Precision-Recall Curve). Voici un tableau récapitulatif de la performance moyenne des différents algorithmes sur tous les datasets, ce tableau est trié par ordre décroissant de performance sur F1_score ensuite Recall ensuite AUC_PR:

Méthode	Accuracy	Precision	Recall	F1-score	AUC-ROC	AUC-PR	Training Time
Bagging (SMOTE)	0.902461	0.712400	0.730658	0.714346	0.898613	0.733351	11.827343
Stacking (SMOTE)	0.897605	0.699270	0.741267	0.713056	0.909534	0.741371	6.819120
Random Forest (SMOTE)	0.901836	0.720510	0.715625	0.711027	0.915047	0.760684	1.226988
Gradient Boosting (SMOTE)	0.870541	0.669978	0.834988	0.703735	0.902723	0.736917	6.655989
RF (weighted, fine-tuned)	0.869344	0.635234	0.861845	0.698980	0.914696	0.770042	62.967726
Bagging	0.908204	0.742837	0.667037	0.697453	0.889295	0.751663	4.821189
XGBoost (weighted, fine-tuned)	0.865929	0.646359	0.828822	0.695847	0.906995	0.743723	22.844824
Random Forest	0.914418	0.763278	0.652635	0.692095	0.895666	0.761383	0.950948
Gradient Boosting	0.907725	0.774261	0.664689	0.689820	0.908736	0.760447	3.518946

Méthode	Accuracy	Precision	Recall	F1-score	AUC-ROC	AUC-PR	Training Time
Random Forest (Weighted)	0.862453	0.627007	0.856235	0.687864	0.904394	0.763200	1.144079
Bagging (Weighted)	0.906089	0.749385	0.649437	0.680992	0.883085	0.742406	5.071231
Stacking (Weighted)	0.851803	0.617313	0.867978	0.677818	0.916612	0.741051	8.500697
Stacking (multi)	0.909194	0.711080	0.645041	0.666985	0.921778	0.759478	3.196537
AdaBoost	0.899623	0.711821	0.657802	0.663114	0.870385	0.693551	5.215613
AdaBoost (Weighted)	0.878453	0.682912	0.645128	0.660586	0.782565	0.591088	3.545089
k-NN (SMOTE)	0.838237	0.598545	0.797662	0.654415	0.882934	0.679142	0.601345
Decision Tree (SMOTE)	0.822136	0.602896	0.812062	0.652068	0.836497	0.605372	0.567774
Decision Tree (Weighted)	0.815587	0.591188	0.816153	0.647877	0.832827	0.613575	0.384423
MLP (SMOTE)	0.833257	0.597231	0.822331	0.646959	0.894184	0.691412	0.519395
Decision Tree	0.876803	0.672292	0.649198	0.646540	0.811192	0.605907	0.393592
k-NN	0.874893	0.703643	0.616083	0.645483	0.843386	0.672738	0.456229
Logistic Regression (SMOTE)	0.821698	0.574135	0.835937	0.637018	0.880968	0.674968	0.476985
SVM (SMOTE)	0.826253	0.580255	0.824599	0.635498	0.885078	0.678601	0.645203
AdaBoost (SMOTE)	0.771508	0.576654	0.883617	0.634611	0.861879	0.639701	9.487567
SVM (Weighted)	0.789247	0.522687	0.860006	0.604647	0.886868	0.681713	0.447927
MLP	0.886271	0.710151	0.565811	0.600892	0.890884	0.704159	0.453180
Logistic Regression	0.878089	0.691439	0.574947	0.596974	0.864485	0.688063	0.666701
Stacking (SVM)	0.877660	0.684113	0.561107	0.593042	0.892727	0.697352	3.033375
Logistic Regression (Weighted)	0.758756	0.502223	0.866096	0.587524	0.878094	0.675005	0.400183

Méthode	Accuracy	Precision	Recall	F1-score	AUC-ROC	AUC-PR	Training Time
SVM	0.876923	0.678859	0.567243	0.586711	0.855987	0.674779	0.427128

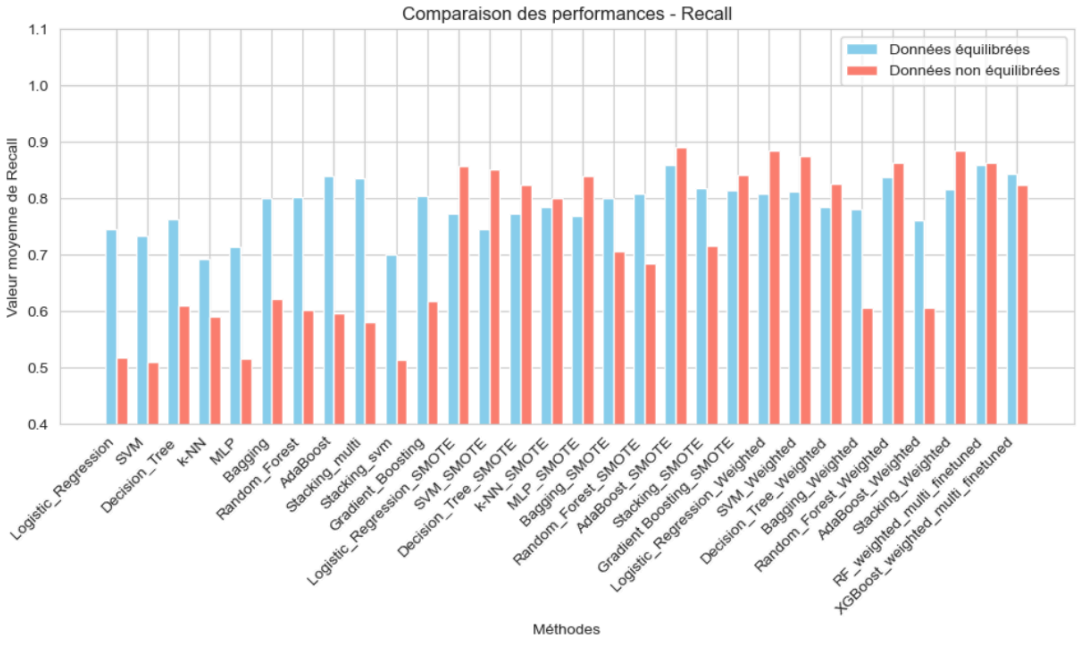
Sans surprise, les méthodes ensemblistes ont en moyenne de meilleurs résultats que les méthodes non ensemblistes:

- Bien que très bien classés en Accuracy, SVM et Logistic_Regression sont les moins bien classés en F1_score et en recall.
- Bagging, stacking, Random_Forest et Boosting sont en haut du classement sur le f1_score et le recall.

Essayons d'affiner notre appréciation en visualisant les barres de performance de chaque méthode par groupe de datasets.

Le groupe 1 est constitué de 7 datasets, ayant une classe majoritaire ne dépassant pas les 60%. Il est censé représenter le groupe de données équilibrées. Le groupe 2 est son complémentaire sur l'ensemble des datasets, c'est à dire les 21 datasets ayant une classe majoritaire dépassant les 60%.

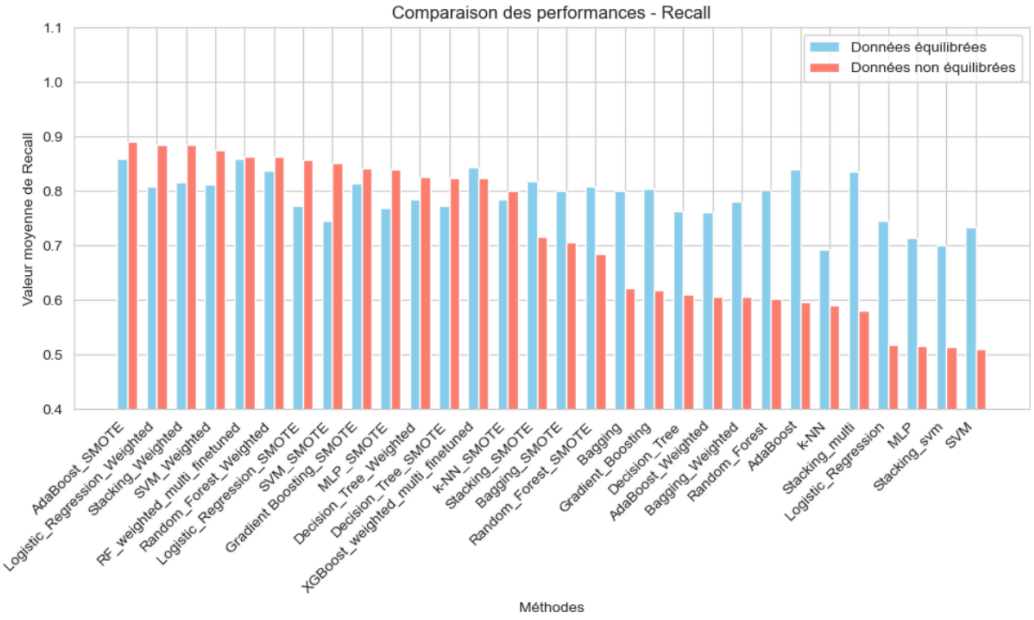
Voici les resultats globaux de la moyenne du Recall par méthode et par groupe:



Les méthodes d'ensembles montrent clairement de meilleurs performances en particuliers sur les datasets déséquilibrés.

Notons que le stacking avec plusieurs SVMs linéaires n'a pas pu bénéficier d'un résultat significativement meilleur qu'un SVM simple sur les données déséquilibrées, il a même montré des résultats moins bons sur les données équilibrées.

Pour mieux le voir, voici le même graphique mais trié par performance de Recall sur les données déséquilibrées :



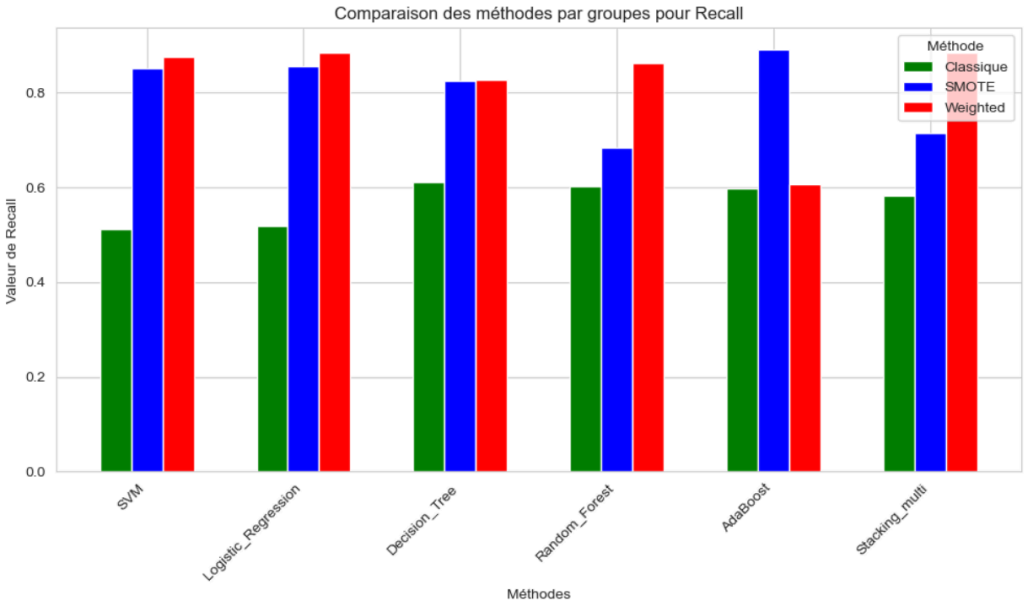
4.2 Impact des techniques de rééquilibrage

4.2.1 Amélioration globale des performances

Nous prenons ici trois algorithmes ensemblistes et trois algorithmes non ensemblistes. Nous étudions l'effet du rééquilibrage par oversampling smote, et par pondération des poids de l'erreur sur la prédiction finale dans les datasets déséquilibrés.

Le graphique représente pour chaque algorithme trois résultats:

- En vert la performance recall de la version classique de la méthode.
- En rouge la performance avec pondération des poids de l'erreur.
- En bleu la version sur-échantillonnée.



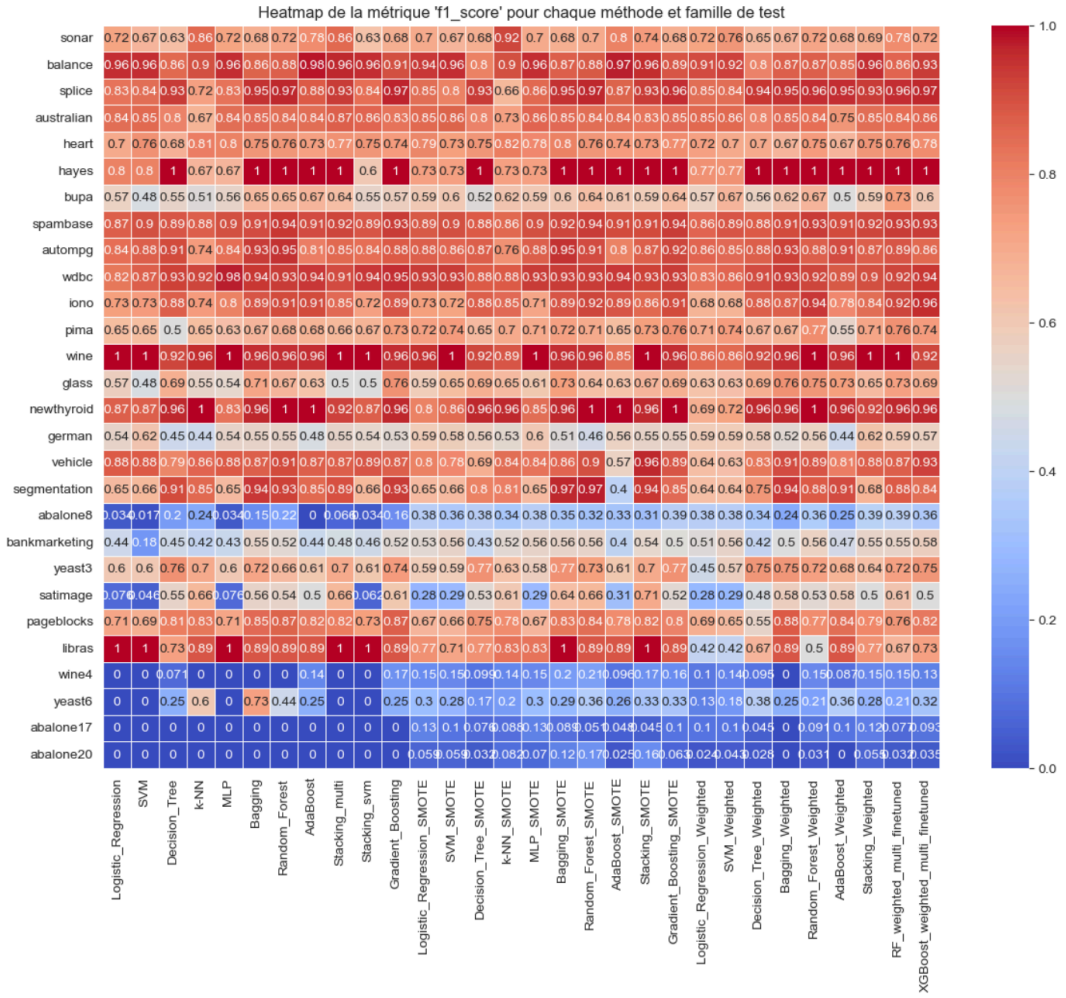
Dans tous les algorithmes, les technique de rééquilibrage ont permis une amélioration significative des résultats:

- Pour les algorithmes non ensemblistes, Les deux techniques semblent avoir un effet du même ordre sur le résultat en particulier pour l'arbre de décision qui semble avoir bénéficié de façon équivalente de l'une ou l'autre.
- Adaboost a beaucoup plus bénéficié de SMOTE, ceci s'explique peut être par le fait que la pondération des poids de l'erreur était intégrée à ses estimateurs, des arbre de décision de faible profondeur.
- Random_Forest et le stacking_multi (familles variées de base-learners) ont plus tiré profît de la pondération du poids de l'erreur.

Dans l'ensemble, sur des datasets de classification binaire dont la classe majoritaire dépasse les 60%, en moyenne tous les algorithmes améliorent leurs résultats avec le rééquilibrage. Une meilleure efficacité semble être du côté de la sensibilité au coût des erreurs.

4.2.2 Cas des datasets fortement déséquilibrés

Il existe un seuil de déséquilibre au delà duquel nos algorithmes se sont confronté à l'impossibilité de bien prédire le bon label, visualisons la Heatmap du score F1 des toutes les méthodes pour tous les datasets:



Rappelons également les proportions de la classe positive dans chaque dataset:

Nom	sonar	balance	splice	australian	heart	hayes	bupa	spambase
Ratio	0.466346	0.460800	0.449348	0.444928	0.444444	0.434783	0.416422	0.398954

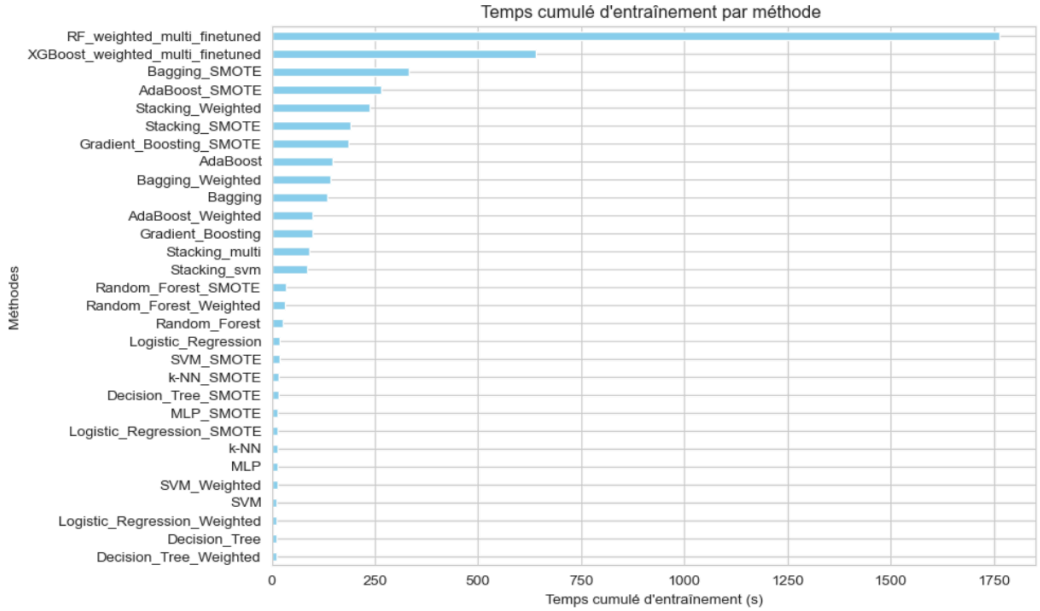
Nom	iono	pima	wine	glass	newthyroid	german	vehicle	segment
Ratio	0.357143	0.348958	0.331461	0.323944	0.302326	0.300000	0.235225	0.142857

Nom	yeast3	satimage	pageblocks	libras	wine4	yeast6	abalone17	abalo
Ratio	0.111493	0.097280	0.095132	0.072727	0.038999	0.024088	0.013886	0.0062

Pour des datasets comme wine4 (ratio de classe 1 de seulement 0.038), presque toutes les méthodes échouent, sauf pour quelques-unes comme XGBoost_weighted_multi_finetuned et RF_weighted_multi_finetuned, qui obtiennent des scores F1 légèrement supérieurs, mais tout de même très faibles. Il en est de même ou pire pour les datasets plus déséquilibrés, et ce malgré les techniques de rééquilibrage. Cela montre que ces datasets sont particulièrement difficiles à traiter, même avec des techniques avancées.

4.3 Coût computationnel

Pour l'aspect coût computationnel, le temps cumulé d'entrainement sur tous les datasets est une information éloquante :



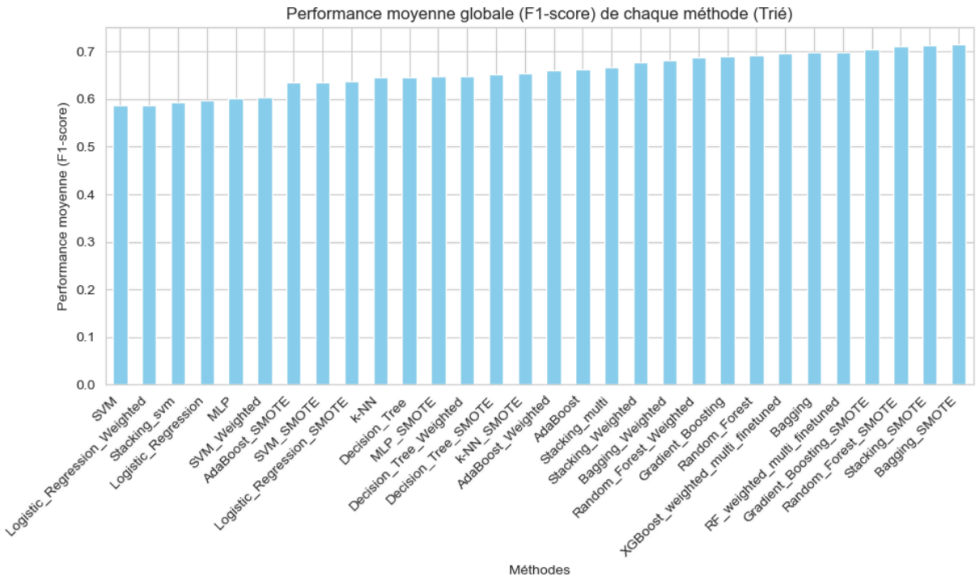
Les algorithmes d'ensemble (Bagging, AdaBoost, Gradient Boosting, Stacking) occupent clairement les premières places en termes de temps d'entraînement. Ces méthodes nécessitent plus de temps, notamment les versions avec fine-tuning, car elles reposent sur la combinaison de multiples modèles, qui sont souvent plus complexes à entraîner.

En particulier, les méthodes comme Random Forest et XGBoost, lorsqu'elles sont ajustées et pondérées, nécessitent des calculs intensifs en raison du nombre d'estimateurs et des processus itératifs de boosting.

Méthodes non-ensemblistes : À l'opposé, des algorithmes plus simples comme Logistic Regression, SVM, k-NN, et Decision Tree prennent beaucoup moins de temps pour l'entraînement. Ces méthodes ne reposent pas sur l'agrégation de plusieurs modèles, ce qui réduit leur coût computationnel. Cela les rend plus rapides à entraîner, mais potentiellement moins performantes dans des situations complexes ou avec des données déséquilibrées.

4.4 Modèles avancés

Ce graphe fait la synthèse de la performance en F1_score moyen de toutes les méthodes :



La méthode XGBoost_Weighted_multi_finetuned et RF_Weighted_multi_finetuned, bien que bien classées ne prennent pas la tête du classement. Sur le critère du F1_score, trois algorithmes ayant bénéficié de l'équilibrage des données par smote qui sont en tête. Il s'agit de trois méthodes ensemblistes :

- Bagging avec smote
- Stacking avec smote
- Random Forest avec smote

Ce résultat pose la question de la pertinence du finetuning poussé rapporté à son coût calculatoire. En effet ces deux méthodes ont consommé plus de la moitié du temps total d'exécution des cross_validations. La random forest, en particulier sur le finetuning des critères de split (gini, entropy) a été très gourmande en ressources. Ce sont certainement des considérations à prendre en compte pour une application non académique pouvant être limitée par le temps ou les ressources physiques.

5. Conclusion et perspectives

Notre étude comparative des méthodes d'ensemble et des techniques de rééquilibrage sur 28 jeux de données déséquilibrés a mis en évidence plusieurs points :

Performance des méthodes d'ensemble : Les algorithmes comme Bagging, Random Forest et les méthodes de Boosting ont généralement obtenu de meilleurs résultats en termes de F1-score, rappel et AUC-PR, notamment sur les datasets fortement déséquilibrés.

Efficacité des techniques de rééquilibrage : SMOTE et la pondération des classes ont amélioré les performances de la plupart des classifieurs. La pondération des classes s'est révélée particulièrement efficace pour certains algorithmes d'ensemble.

Coût computationnel : Les méthodes d'ensemble, bien que plus performantes, ont montré un temps de calcul significativement plus élevé, surtout dans leurs versions ajustées.

Limites sur les données très déséquilibrées : Pour les jeux de données avec un ratio de classe minoritaire inférieur à 5%, même les méthodes avancées ont montré des performances limitées.

Plusieurs pistes pourraient être explorées pour approfondir cette étude :

Techniques de sous-échantillonnage : partir plutôt sur le sous-échantillonnage de la classe majoritaire.

Approches combinées : La combinaison de techniques de sur-échantillonnage et de sous-échantillonnage.

Algorithmes spécifiques : L'intégration d'algorithmes conçus pour les problèmes déséquilibrés, comme ADASYN ou BorderlineSMOTE.

Méthodes basées sur les coûts : Une exploration plus poussée des méthodes basées sur les coûts.

Ces pistes pourraient permettre d'améliorer les performances sur les datasets très déséquilibrés et d'approfondir notre compréhension des possibilités d'ajustement entre les caractéristiques des données, les algorithmes de classification et les techniques de rééquilibrage.