Projet Supervised Machine Learning

Master 1 MIASHS (2022-2023)

Julien Bastian & Ugo Zennaro
Anthropologie, Sciences Sociales et Politiques (ASSP)
Université de Lyon, Université Lumière Lyon 2

Enseignant : Guillaume Metzler

Résumé

L'objectif de ce travail est d'explorer des méthodes d'apprentissage supervisé. Pour se faire nous explorons des méthodes non paramétriques (k plus proches voisins), linéaires paramétriques (séparateurs à vastes marges et régression logistiques) et des méthodes non linéaires (par bagging et par boosting). Pour prolonger l'études de ces méthodes, nous verrons des façons de les renforcer, par combinaisons de celles-ci et par des méthodes d'échantillonages des données d'entrainement qui permettent de contourner et/ou de façonner la morphologie des différents jeux de données étudiés. Ceci avec une emphase sur l'apprentissage dans le cadre de jeux de données déséquilibrés.

1 Introduction

L'objectif de ce projet est d'étudier les résultats de plusieurs méthodes de machine learning supervisé et de proposer des variantes de celles-ci qui visent à améliorer leurs performances. La tâche de machine learning que nous considerons ici est la classification binaire qui est un cas spécifique de machine learning supervisé. On étudiera pour cela trois types d'approches, des non paramétriques, des paramétriques linéaires et des non linéaires

Pour l'évaluation de la qualité des méthodes et de leurs variantes nous avons choisi de mettre l'emphase sur la classification dans le cas de données déséquilibrées [Elkan, 2001]. On parle de données déséquilibrées lorsque une classe est plus représentée qu'une autre (généralement la classe positive). Les datasets que nous considérons ont été sélectionnés pour représenter une gradation dans le déséquilibre. Allant du déséquilibre extrême avec Abalone20 jusqu'au quasi équilibre avec Heart. La table 1 donne les informations principales sur ceux-ci, on voit un déséquilibre vraiment important pour les deux premiers dataset et plus faible ensuite.

Dataset	Individus	Variables	IR
Abalone20	4177	10	159.7
Yeast6	1484	8	41.4
German	1000	24	2.3
Newthyroid	215	5	2.3
Wine	178	13	2
Iono	351	34	1.9
Heart	270	13	1.2

TABLE 1 – Morphologie des datasets, la colonne IR représente le "imbalance ration" c'est à dire l'inverse de la proportion d'individus positifs dans la population.

On présentera les méthodes ainsi que les variantes que nous proposons dans la partie 2. Puis on discutera leurs résultats dans la partie 3.

2 Méthodologie

Notations et critères de performances

On se placera dans le cadre d'une classification binaire, où les données sont issues d'une distribution inconnue $\mathcal{D} = \mathcal{X} \times \mathcal{Y}$ où $\mathcal{X}$ représente la distribution des features et $\mathcal{Y}$ celle des labels. On considèrera un échantillon $S = \{\mathbf{x}_i, y_i\}_{i=1}^n$ de taille m où $\mathbf{x}_i \in \mathbb{R}^d$ est le vecteur de dimension d des features de l'individu i et $y_i \in \{0, 1\}$ son étiquette ou *label*. On appelera label positif (et par abus de language données positive) le label qui correspond à la classe cible, ici quand $y = 1$, inversement pour un label positif qui est le cas où $y = 0$. On pose $S = S_+ \cup S_-$ avec n_+ données $\in S_+$, n_- données $\in S_-$ et $n = n_+ + n_-$

On note, les nombres en lettres minuscules : x, les vecteurs en gras : $\mathbf{x}$, avec $\mathbf{x} = \{\mathbf{x}_i, ..., \mathbf{x}_n\}$, et les matrices en majuscules : $\mathbf{X}$.

Pour mesurer les résultat de la classification on utilise les informations contenue dans la matrice de confusion, définie comme :

		Label	
		Positive	Negative
Prédiction	Positive	TP	FP
	Negative	FN	TN

Avec TP (resp. TN) le nombre de vrais positifs prédits (resp. négatifs) et FP (resp. FN) le nombre de faux positifs prédits (resp. négatifs).

Plusieurs critéres de performances existent pour la classification, particuliérement l'*accuracy* qui considére le taux de prédiction correctes, la *precision* qui est la confiance dans la prédiction d'une étiquette positive et le *recall* représentant le taux de vrai positif. On les définit comme :

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

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

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

L'$accuracy$ n'est pas adapté pour des données déséquilibrées car, le nombre de positifs étant potentiellement trés faible, elle peut être trés élévée sans permettre de conclure sur des bonnes performances sur la classe positive et donc sur la qualité de la classification en générale. La $precision$ et le $recall$ sont en revanche beaucoup plus adaptées à notre situation car ils se concentrent sur les labels positifs. Dans ce travail nous utiliserons comme critére le F_1 qui synthétise la $precision$ et le $recall$. On le définit de la manière suivante :

$$F_1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Approches non paramétriques

Dans cette partie on se concentrera sur l'algorithme non paramétrique des k plus proches voisins (k-NN). Pour tenter de l'améliorer nous emploierons plusieurs méthodes de rééchantillonnage, à savoir : les *Condensed Nearest Neighbor*,le sous-échantillonnage aléatoire et le sur-échantillonnage *Synthetic Minority Over-sampling Technique* (SMOTE) [Chawla et al., 2002]. Puis, on considerera l'algorithme des γ k-nn [Viola et al., 2021] qui modifie la distance aux données en fonctions de leur classes.

k-NN

L'algorithme des k-NN repose sur la notion de distance entre individus, pour deux vecteurs $\mathbf{x_i}$ et $\mathbf{x_j}$ on définit leur distance comme $d(\mathbf{x_i}, \mathbf{x_j}) = \|\mathbf{x_i} - \mathbf{x_j}\|$. L'idée est donc de prédire la classe d'un individu nouveau comme

$$\hat{c} = \underset{c \in \mathcal{Y}}{\operatorname{argmax}} \frac{k_c}{k},$$

où k_c est le nombre d'individu appartenant à la classe c parmis les k plus proches voisins de la données. La prédiction du label d'une nouvelle données depend donc uniquement de sa position dans l'espace des features ce qui est une méthode à la fois simple et puissante. Cependant, le coût en calcul de cet

algorithme est assez élévé et croissant avec le nombre d'individu étant donné que la prédiction requiert de calculer la distance à tous les individus du jeu de données d'entrainement.

De plus dans le cas de données déséquilibrées la prédiction est biaisée par le ratio de données négatives. En effet, celles-ci étant beaucoup plus representées dans l'espace la probabilité de les trouver parmis les voisins d'une données et d'autant plus grande. Ce qui rend plus difficile la définition de régions de l'espace correspondant à des données positives.

Rééchantillonnage

Pour équilibrer la distibution des données il est possible d'employer des méthodes de rééchantillonnage. Nous en présentons trois ici. Tout d'abord les CNN dont l'objectif est de selectionner un sous ensemble S' de l'échantillon S de sorte à ce que ce nouvel échantillon permette de bien classifier toutes les données du jeu d'entrainement. L'algorithme est le suivant :

Algorithm 1: *Condensed Nearest Neighbor*

Input : Un échantillon d'apprentissage S
Output : Un sous ensemble S' de S
$S' = \emptyset$
Supprimer une données de S aléatoirement et l'ajouter à S'
for Tout les $\mathbf{x_i} \in S$ **do**
  if $\mathbf{x_i}$ est mal classé en utilisant 1-NN dans S' **then**
    L'ajouter à S' et le supprimer de S
  end if
end for

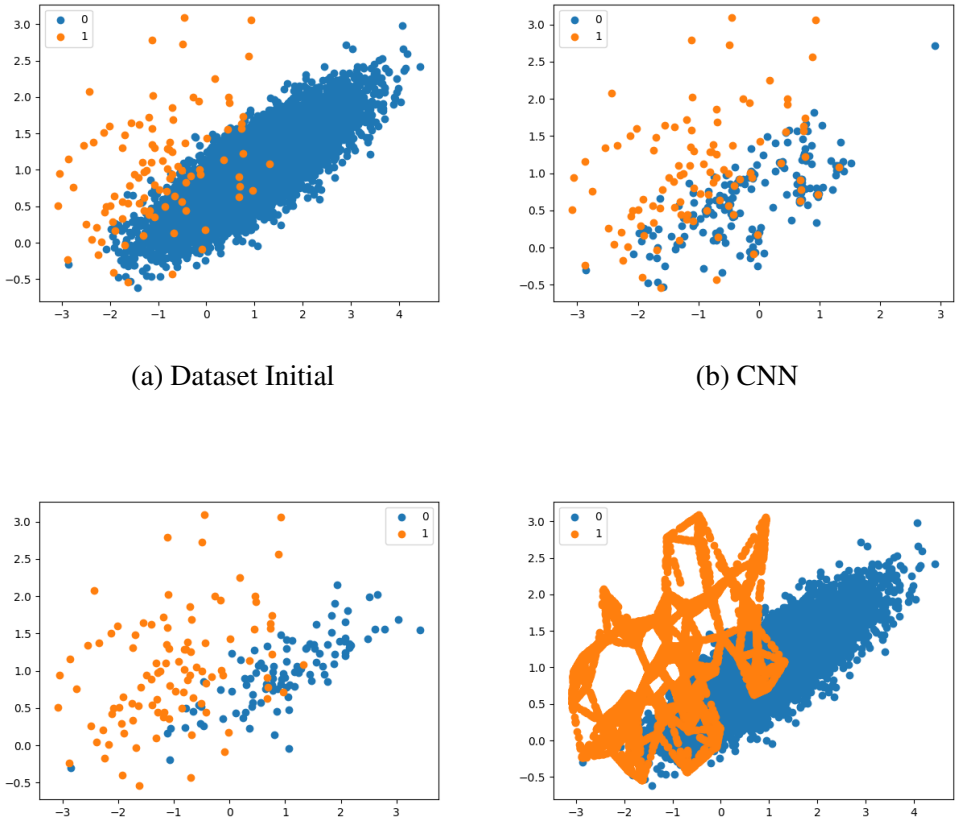
Il est important de noter que l'usage des CNN n'est pas dédié aux distributions déséquilibrées, l'algorithme permet initialement d'obtenir un jeu de données de plus taille afin de réduire le coût de calcul des k-NN. Cependant il s'adapte bien au contexte que nous étudions, surtout lorsque l'ont limite le sous-échantillonnage aux données sur-représentées ce que nous faisons ici.

Ensuite, le sous-échantillonnage aléatoire. Ici la méthode est extrêmement simple, il s'agit simplement de supprimer des données de la classe majoritaire jusqu'à ce que $n_+ = n_-$. L'inconvénient de cette méthode est qu'elle suppose d'amputer une partie potentiellement importante contenue dans le dataset ce qui peut impacter la capacité de généralisation de la méthode apprise et conduire à de l'overfitting.

Enfin, le sur-échantillonnage par SMOTE [Chawla et al., 2002] qui vise à créer de nouvelles données de la classe minoritaires dite synthétiques jusqu'à ce que $n_+ = n_-$. La procédure pour synthétiser ces données est la suivante :

— Selectionner un individu appartennant à la classe minoritaire.
— Calculer ces k plus proches voisins de la même classe.
— En selectionner un alétoirement.
— Placer un nouvel individu sur le segment qui les relie.

Cette technique à l'avantage d'enrichir aléatoirement le dataset avec des données semblables à celles de classe minoritaire présentent initialement. La figure 1 illustre les méthodes présentées sur un jeu de données artificiel déséquilibré (1% de positifs), on voit clairement la particularité de SMOTE dans sa façon d'enrichir l'espace, mais également comment les CNN et le sous-échantillonnage aléatoire engendre des espaces tout à fait différents.



(a) Dataset Initial

(b) CNN

(c) Sous-échantillonnage aléatoire

(d) SMOTE

FIGURE 1 – Illustration des méthodes d'échantillonnage présentées

γk-NN

Les γk-NN [Viola et al., 2021] sont une adaptation des k-NN pour des situations de données déséquilibrées. Il s'agit de modifier le calcul de la distance entre les individus en fonction qu'ils appartiennent à la classe majoritaire ou non. En particulier, pour une donnée $\mathbf{x_i}$ la distance devient :

$$d(\mathbf{x_i}, \mathbf{x_j}) = \begin{cases} d(\mathbf{x_i}, \mathbf{x_j}), & \text{if } \mathbf{x_j} \in S_-. \\ \gamma d(\mathbf{x_i}, \mathbf{x_j}), & \text{if } \mathbf{x_j} \in S_+, \text{ avec } \gamma \in [0; 1]. \end{cases}$$

Ainsi, lorsqu'une donnée appartient à la classe minoritaire la distance des autres avec celle-ci est réduite par un facteur $0 \leq \gamma \leq 1$ ce qui va augmenter le poid relatif de cette classe dans la prédiction.

Approches paramétriques linéaires

Ici, nous étudierons les performances de plusieurs méthodes qui construisent des classifieurs linéaires. D'abord les *Support Vector Machine* (SVM) avec noyau linéaire et noyau polynomial de degré 2, puis la régression logistique. Pour améliorer les résultats de ces méthodes on leur ajoutera une pondération par le coût des erreurs de classification qui les rendra sensibles à la distribution des données lors de l'apprentissage.

SVM

La classification par SVM consiste en la construction d'un hyperplan qui maximise l'écart entre les classes (la marge). L'équation de l'hyperplan est donnée par les paramétres $\mathbf{w}$ et b. Dans une situation idéale on aurait :

$$\mathbf{w^T x_i} - b \geq 1, \text{if } y_i = 1,$$
$$\mathbf{w^T x_i} - b \leq -1, \text{if } y_i = -1,$$

ce qui n'arrive que lorsque les classes sont complétement séparables (hard-margin). En général ce n'est pas possible. On utilise alors la $hingeloss$ pour l'apprentissage :

$$max(0, 1 - y_i(\mathbf{w^T x_i} - b))$$

On peut alors écrire le problème d'optimisation comme :

$$\underset{\mathbf{w},b,\zeta}{\text{minimize}} \|\mathbf{w}\|_2^2 + C\sum_{i=1}^{N} \zeta_i$$

$$\text{s.t. } y_i(\mathbf{w^T x_i} - b) \geq 1 - \zeta_i,\ \zeta_i \geq 0.$$

On minimise donc l'erreur du modèle ainsi que ses paramètres.

De plus, il est possible de transformer les données grâce à un noyau afin d'introduire une non linéarité dans les données, le séparateur reste lui linéaire. Ce noyaux est une fonction $K(\cdot, \cdot)$ qui prend en entrée deux vecteurs et renvoie un nombre réel, il permet de faire une projection des données dans un nouvel espace $\phi(\mathbf{x})$ sans nécessiter la connaissance de leur expression dans cet espace. Ce noyau est en effet le produit scalaire de ces projections et simplifie donc grandement les calculs, on a alors

$$K(\phi(\mathbf{x}), \phi(\mathbf{x'})) = \langle \phi(\mathbf{x}), \phi(\mathbf{x'}) \rangle.$$

Ici, on emploiera deux noyaux, le noyau linéaire $K(\phi(\mathbf{x}), \phi(\mathbf{x'})) = \langle \mathbf{x}, \mathbf{x'} \rangle$ et le noyau polynomial de degré 2 $K(\phi(\mathbf{x}), \phi(\mathbf{x'})) = (\langle \mathbf{x}, \mathbf{x'} \rangle + c)^2$

Régression Logistique

Dans le cadre de la régression logistique l'objectif est "fiter" une fonction logistique à la probabilité d'appartenance à chacune des classes en fonctions. La fonction logistique est définie comme :

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

qui renvoie des valeurs dans l'intervalle $[0; 1]$. Dans le cas du machine learning on a :

$$\sigma(h(x)) = \frac{1}{1 + \exp(-h(x))},$$

la fonction logistique paramétrée par une fonction $h(x)$ qui est un modèle linéaire sur les données. Son apprentissage se fait par minmisation de moins la log vraisemblance :

$$\min_h -\frac{1}{n}\sum_{i=1}^{n} [y_i \ln(\sigma(h(x))) + (1 - y_i) \ln(1 - \sigma(h(x)))]$$

Pondération par le coût

*

Ces méthodes, bien que tout à fait performantes, peuvent souffrir de données déséquilibrées. Pour palier à ce problème nous les transformerons en leur version pondérées par le coût, c'est à dire en une version dans laquelle toutes les erreurs de prédiction n'ont pas la même importance. Pour cela on définit la matrice de coût C dont la case (i, j) représente le coût de prédire i lorsque le vrai label est j.

		Label	
		Positive	Negative
Prédiction	Positive	$C(1,1)$	$C(1,0)$
	Negative	$C(0,1)$	$C(0,0)$

Elle peut prendre plusieurs forme en fonction de la tâche réalisée, mais dans le cas des données déséquilibrées on a :

		Label	
		Positive	Negative
Prédiction	Positive	0	1
	Negative	n_+/n_-	0

Ainsi, le coût associé à une mauvaise classification d'une donnée minoritaire est équivalent à l'inverse de la proportion de sa classe dans la population. Avec cette nouvelle information on peut mettre à jour les fonctions à optimiser dans les méthodes présentées plus haut. On prend $w_- = C(1,0)$ et $w_+ = C(0,1)$ et à chaque donnée x_i on associe son poids $w_i \in w_-, w_+$ et l'optimisation de la régression logistique devient :

$$\min_h -\frac{1}{n}\sum_{i=1}^{n} w_i[y_i \ln(\sigma(h(x))) + (1-y_i)\ln(1-\sigma(h(x)))].$$

Celle des SVM devient :

$$\underset{\mathbf{w},b,\zeta}{\text{minimize}}\ \|\mathbf{w}\|_2^2 + C\sum_{i=1}^{N} w_i\zeta_i$$
$$\text{s.t. } y_i(\mathbf{w^T x_i} - b) \geq 1 - \zeta_i,\ \zeta_i \geq 0.$$

Approches non linéaires

Pour cette partie on portera l'emphase sur des techniques de clasification non linéaires. En premier lieu le $Bagging$, avec principalement les ***Random Forest*** pour lesquelles nous proposerons deux variantes : les ***Random Forest*** pondérées par le coût et le bagging de SVM à noyau polynomial de degré 2. Puis, on considérera le $Boosting$ dans sa version l' ***Adaptative Boosting*** ($AdaBoost$). À nouveau on proposera deux variantes : $AdaCost$ [Fan et al., 1999] qui est une version de $AdaBoost$ sensible au coût et du $Boosting$ par $AdaBoost$ sur des modéles de régression logistiques également sensibles au coût.

Bagging

Le $Bagging$, pour $bootstrap\ aggregating$ est une méthode qui consiste en la combinaison de plusieurs modèles h_t afin de réduire leur variance est donc d'améliorer leurs performances. Chacun d'eux est entrainé sur un échantillon S' de S obtenu par $bootstrap$, le modèle final H est la moyenne des

différents h_t qu'on aura optimisés. L'algorithme qui permet de réaliser cette procédure est le suivant :

Algorithm 2: *Bagging*

Input : Un échantillon d'apprentissage S et un nombre de modèles T
Output : Un modèle H
for $t = 1$ to T **do**
  Tirer un échantillon bootstrap S' depuis S
  Entrainer un modèle h_t sur S'
end for
$H = \frac{1}{T}\sum_{t=1}^{T} h_t$

Les h_t peuvent prendre différentes formes, généralement ce sont des arbres de décision, auquel cas on obtient un modèle appelé $RandomForest$. C'est celui qui nous servira de point de comparaison pour évaluer la qualité des variantes que nous proposons. La première est une version du $RandomForest$ sensible au coût en qui utilise la pondération présentée dans la partie précédente au sein des arbres de décision. La deuxième est un $Bagging$ dans lequel les h_t sont des SVM à noyaux polynomial de degré 2 sensibles au coût.

Boosting

A l'image du $Bagging$, le $Boosting$ est une combinaison de modèle h_t. Cependant, dans ce cas il ne sont pas appris sur des échantillons $bootstrap$ mais séquentiellement de façon à corriger les erreurs des précédents, c'est à dire que h_{t+1} est construit dans l'objectif de corriger les erreurs de h_t. La méthode de boosting de référence est l'algorithme $AdaBoost$ qui est défini comme suit :

Algorithm 3: *AdaBoost*

Input : Un échantillon d'apprentissage S et un nombre de modèles T
Output : Un modèle H
for $i = 1$ to n **do**
  $w_i^{(1)} = 1/n$
end for
for $t = 1$ to T **do**
  Apprendre un modèle h_t sur S avec les poids $\mathbf{w}^{(t)}$
  $\epsilon_t = \sum_{i=1}^{n} w_i^{(1)} \mathbb{1}_{\{h_t(\mathbf{x}_i) y_i < 0\}}$
  $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
  $Z_t = 2\sqrt{\epsilon_t(1-\epsilon_t)}$
  for $i = 1$ to n **do**
    $w_i^{(t+1)} = w_i^{(t)} \frac{\exp(-\alpha_t y_i h_t(\mathbf{x}_i))}{Z_t}$
  end for
end for
$H = \frac{1}{T}\sum_{t=1}^{T} \alpha_t h_t$

La version sensible au coût $AdaCost$ [Fan et al., 1999] est similaire à la différence qu'elle intégre une pondération dans la mise à jour des poids. Cette pondération est représentative du déséquilibre dans les données mais elle n'est pas égale à celle présentée plus haut avec la matrice de coût. L'algorithme est le suivant :

Algorithm 4: *AdaCost*

Input : Un échantillon d'apprentissage S et un nombre de modèles T
Output : Un modèle H
for $i = 1$ to n **do**
  $w_i^{(1)} = 1/n$
end for
for $t = 1$ to T **do**
  Apprendre un modèle h_t sur S avec les poids $\mathbf{w}^{(t)}$
  $\epsilon_t = \sum_{i=1}^{n} w_i^{(1)} \mathbb{1}_{\{h_t(\mathbf{x}_i) y_i < 0\}}$
  $\alpha_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
  $Z_t = 2\sqrt{\epsilon_t(1-\epsilon_t)}$
  for $i = 1$ to n **do**
    $w_i^{(t+1)} = w_i^{(t)} \frac{\exp(-\alpha_t y_i h_t(\mathbf{x}_i)\beta(i))}{Z_t}$
    Où $\beta(i) = \beta(\text{sign}(y_i h_t(x_i), c_i)$ avec c_i la classe des données, est une fonction d'ajustement de coût
  end for
end for
$H = \frac{1}{T} \sum_{t=1}^{T} \alpha_t h_t$

Les h_t appris sont généralement des arbres de décisions mais il est possible d'utiliser d'autres types de classifieurs. Dans notre proposition ils deviennent des régressions logistiques sensibles au coût.

3 Expériences

Protocole expérimental

Pour chacune des méthodes dont nous étudierons les performances si plusieurs hyperparamètres sont donnés ils sont sélectionnés par cross validation en 5-fold. Le jeu de données d'entrainement est une sélection aléatoire de 20% des données.
Pour les knn et ces variantes rééchantillonés le seul hyperparamètre est k dans la range $\{1, 3\}$, pour le γk-nn on ajouter le paramètre γ qui peut prendre ses valeurs dans $\{0.25, 0.5, 0.75, 1\}$. Pour les SVM on cross-valide le paramètre C, qui contrôle la pénalisation des erreurs dans l'intervalle, pour les valeurs $\{0.1, 1, 10\}$. Enfin, pour les méthodes non linéaires on contrôle le nombre maximal de sous modèles qu'elles contiennent T qui peut prendre les valeurs $\{25, 50, 100\}$

Résultats

Approches non paramètriques

On analyse d'abord les différences de performances pour les k-NN et ses versions rééchantillonnées. Celle-ci sont visibles dans le tableau 2 pour lequel trois analyses sont à faire. En premier lieu on remarque que la version sous-échantillonnée offre de moins bonnes performances que la versions de base ce qui correspond avec notre hypothése initiale quant au fait que cette stratégie entraine une trop grande perte d'information et peut conduire à de l'overfitting. Ensuite, on voit que les deux version sur-échantillonnées permettent d'améliorer les performances avec un léger avantage pour l'échantillonnage SMOTE qui offre (de peu) les meilleures performances en moyenne mais également qui

TABLE 2 – F1 moyen en test sur 10 iterations pour les knn et ses versions rééchantillonées

Dataset	k-NN	CNN k-NN	sous-échantillonnage k-NN	SMOTE k-NN
00.62% abalone20	0.0 ± 0.0	0.0 ± 0.0	2.7 ± 0.5	**8.1** ± 5.0
02.36% yeast6	45.7 ± 14.5	**48.2** ± 14.8	19.8 ± 4.2	39.8 ± 8.3
30.00% german	41.7 ± 7.4	48.5 ± 5.8	**50.3** ± 5.8	49.0 ± 5.5
30.23% newthyroid	94.7 ± 2.9	93.4 ± 4.6	92.8 ± 4.3	**95.2** ± 3.3
33.15% wine	**86.3** ± 5.1	85.5 ± 4.5	84.9 ± 7.5	86.0 ± 4.9
35.90% iono	82.5 ± 4.1	**87.5** ± 4.1	84.3 ± 3.5	86.0 ± 4.4
44.44% heart	60.7 ± 5.1	**60.8** ± 5.1	60.7 ± 6.4	60.6 ± 5.1
Moyenne	58.8 ± 5.6	60.6 ± 5.6	56.5 ± 4.6	**60.7** ± 5.2
Temps moyen (en secondes)	0.3 ± 0.0	**17.8** ± 1.4	0.1 ± 0.0	0.6 ± 0.0

TABLE 3 – F1 score moyen en test sur 10 itérations k-NN, SMOTE et γk-NN

Dataset	k-NN	SMOTE k-NN	γk-NN
Abalone20	0.0 ± 0.0	**8.7** ± 5.3	4.7 ± 2.6
Yeast6	45.7 ± 14.5	39.0 ± 8.9	**49.0** ± 13.2
German	41.7 ± 7.4	**48.8** ± 5.1	48.2 ± 2.5
Newthyroid	94.7 ± 2.9	**95.5** ± 3.2	94.1 ± 3.9
Wine	86.3 ± 5.1	85.3 ± 6.2	**86.3** ± 5.1
Iono	82.5 ± 4.1	85.4 ± 2.5	**92.1** ± 3.5
Heart	60.7 ± 5.1	60.7 ± 5.4	**65.3** ± 4.0
Moyenne	58.8 ± 5.6	60.5 ± 5.2	**62.8** ± 5.0
Temps moyen (en secondes)	0.3 ± 0.0	0.6 ± 0.0	**0.8** ± 0.0

performe de loin le mieu sur le dataset le plus déséquilibré Abalone20 avec un score F1 moyen de 8.1, bien que ce résultat reste faible. Enfin, on voit qu'entre l'échantillonnage par CNN et celui par SMOTE le premier est largement plus couteux en temps d'entrainement avec en moyenne 17.8 secondes contre 0.6. Ce résultat est attendu au regard de l'algorithme des CNN qui demande une grande quantité de calcul pour construire l'échantillon de plus petite dimension. Au regard de ces informations on peut conclure que la méthode SMOTE est la plus pertinente pour réaliser le rééchantillonnage des données pour les knn, tant sur le plan des performances que du temps d'entrainement.

On va donc maintenant comparer les performances de la méthode SMOTE avec celle des γk-NN. Le tableau 3 compile ces résultats on voit que la méthode des γk-NN offre les meilleurs résultats avec un F1 score moyen de 62.8 contre 60.5 pour SMOTE et 58.8 pour les k-NN d'origine, avec de meilleures performances sur quasiment tous les datasets. De plus son temps d'apprentissage est négligeable et se trouve dans le même ordre de grandeur que celui des deux autres méthodes. Seul ombre au tableau, elle performe significativement moins bien que SMOTE sur Abalone20 alors que c'est le dataset le plus déséquilibré, la méthode pourrait donc montrer ses limites dans des cas de déséquilibre extrême. Cependant, ce point négatif est en partie compensée par ses résultat sur Yeast6, qui est lui aussi fort déséquilibré, pour lequel elle offre les meilleures performances. On conclut donc que parmis les variantes des k-NN proposées les γk-NN est la meilleure.

TABLE 4 – F1 score moyen en test sur 10 itérations SVM linéaire et variantes

Dataset	SVM linéaire	SMOTE SVM linéaire	SVM linéaire pondéré par le coût
Abalone20	0.0 ± 0.0	**7.0** ± 1.2	6.8 ± 1.4
Yeast6	0.0 ± 0.0	**31.7** ± 5.3	26.4 ± 3.2
German	5.3 ± 7.6	59.1 ± 4.4	**60.0** ± 5.0
Newthyroid	55.7 ± 9.8	**70.5** ± 10.0	70.0 ± 9.1
Wine	**89.6** ± 5.9	84.1 ± 7.6	82.7 ± 8.8
Iono	**88.9** ± 4.0	87.8 ± 3.5	88.6 ± 4.9
Heart	59.9 ± 8.4	**65.7** ± 7.0	64.1 ± 8.4
Moyenne	42.8 ± 5.1	**58.0** ± 5.6	56.9 ± 5.8
Temps moyen (en secondes)	0.2 ± 0.0	**2.6** ± 0.2	0.8 ± 0.1

TABLE 5 – F1 score moyen en test sur 10 itérations SVM polynomial et variantes

Dataset	SVM polynomial	SMOTE SVM polynomial	SVM polynomial pondéré par le coût
Abalone20	0.0 ± 0.0	**8.1** ± 2.5	7.3 ± 2.2
Yeast6	9.8 ± 15.2	**32.4** ± 4.1	27.8 ± 3.1
German	41.5 ± 5.7	62.1 ± 3.7	**62.5** ± 3.3
Newthyroid	70.1 ± 8.4	**78.9** ± 9.7	76.2 ± 8.6
Wine	**89.1** ± 6.6	88.1 ± 7.0	86.4 ± 7.4
Iono	91.9 ± 3.1	**92.6** ± 2.8	91.9 ± 3.0
Heart	63.8 ± 7.9	66.8 ± 7.7	**67.5** ± 9.2
Moyenne	52.3 ± 6.7	**61.3** ± 5.4	59.9 ± 5.3
Temps moyen (en secondes)	0.2 ± 0.0	**2.0** ± 0.4	0.7 ± 0.1

Approches linéaires

Dans cette partie on va se concentrer sur les résultats des SVM, de la régression logistique et de leurs variantes. Ils sont lisibles dans les tableaux 4 5 6 dans lesquels plusieurs choses sont notables. D'abord, les versions SMOTE et pondérées par le coût performent systématiquement mieux que les méthodes originales, c'est particulièrement visible pour la régression logistique 6. A quoi on peut ajouter que c'est dans tous les cas SMOTE qui fonctionne le mieux, le plus grand écart entre SMOTE et pondération par le coût est notable pour le SVM à noyau polynomial 5. En revanche, SMOTE entraine un temps d'apprentissage largement supérieur pour les SVM. En effet, celui-ci est multiplié par un facteur > 2 par rapport à la pondération par le coût. Mais il reste généralement supportable ne dépassant pas les 3 secondes en moyennes.

TABLE 6 – F1 score moyen en test sur 10 itérations régression logistique et variantes

Dataset	Régression logistique	SMOTE Régression logistique	Régression logistique pondérée par le coût
Abalone20	0.0 ± 0.0	**5.9** ± 1.0	5.0 ± 0.7
Yeast6	0.0 ± 0.0	**27.0** ± 4.2	24.6 ± 4.0
German	11.7 ± 4.8	**50.1** ± 3.7	49.9 ± 4.0
Newthyroid	0.0 ± 0.0	74.4 ± 10.3	**76.9** ± 9.9
Wine	0.0 ± 0.0	**81.1** ± 7.6	80.8 ± 7.7
Iono	83.4 ± 4.0	**88.4** ± 2.9	88.2 ± 4.0
Heart	42.0 ± 11.5	64.8 ± 9.1	**65.5** ± 9.6
Moyenne	19.6 ± 2.9	**56.0** ± 5.5	55.8 ± 5.7
Temps moyen (en secondes)	0.0 ± 0.0	**0.0** ± 0.0	0.0 ± 0.0

A l'éclairage de ces informations on choisit de conserver les versions pondérées par le coût pour construire des versions non linéaires. Bien que leurs résultats soient légérement inférieurs ce choix est motivé par deux raisons :

— Le temps d'apprentissage : 2 à 3 secondes est supportable quand on entraine le modèle une seule fois mais dans le cadre des approches non linéaires cet entrainement pourra être réalisé jusqu'à 100 fois en fonction du nombre de sous modèles h_t qui composent H

— La praticité : l'implémentation des méthodes pondérées par le coût est bien plus intuitif pour construire des méthodes non linéaires car aucun prétraitrement n'est requis il suffit de légérement modifier la fonction à optimiser en fonction du cas considéré.

A partir de maintenant lorsque l'on parlera de ces méthodes il sera sous entendu qu'il s'agit de leur version pondérée par le coût.

Approches non linéaires

On se concentre dans un premier temps sur les méthodes de bagging. D'abord on compare les versions bagging des approches linéaires sélectionnées plus haut, puis on conservera la plus performante pour la comparer avec les RandomForest. On a donc dans le tableau 7 les résultats des du bagging des méthodes linéaires, ici le résultat est sans appel : la combinaison de SVM à noyau polynomial est largement plus performante, avec un score F1 moyen de 60.6 contre 57.2 pour le SVM à noyau linéaire et 54.5 pour la régression logistique, mais également des meilleures performances sur absolument tous les datasets. Naturellement c'est cette méthode que l'on conserve pour la comparaison avec les RandomForest. En revanche le temps d'apprentissage explose, atteignant quasiment 1 minutes pour les SVM. Il est d'ailleurs intéressant de noter que la régression logistique est le seul cas où les performances en bagging sont inférieures aux performances d'un seul modèle (54.5 contre 55.8), la cause de ce résultat n'est pas claire pour nous mais il semble être du à une baisse de performance sur Heart (61.9 contre 65.5) qui est le dataset le moins déséquilibré, une explication est donc certainement à chercher de ce côté.

TABLE 7 – F1 score moyen en test sur 10 itérations bagging des méthodes linéaires

Dataset	Bagging SVM polynomial	Bagging SVM linéaire	Bagging Régression Logistique
Abalone20	**8.4** $\pm$ 2.7	7.4 $\pm$ 1.2	5.1 $\pm$ 1.1
Yeast6	**32.3** $\pm$ 4.2	29.0 $\pm$ 4.9	25.8 $\pm$ 3.9
German	**62.7** $\pm$ 3.3	59.6 $\pm$ 4.7	49.8 $\pm$ 3.9
Newthyroid	**76.1** $\pm$ 9.3	68.2 $\pm$ 8.3	71.0 $\pm$ 11.4
Wine	**86.4** $\pm$ 7.4	83.2 $\pm$ 8.3	80.6 $\pm$ 8.6
Iono	**90.2** $\pm$ 3.7	89.5 $\pm$ 3.3	87.3 $\pm$ 4.7
Heart	**68.0** $\pm$ 6.4	63.4 $\pm$ 7.9	61.9 $\pm$ 10.9
Moyenne	**60.6** $\pm$ 5.3	57.2 $\pm$ 5.5	54.5 $\pm$ 6.3
Temps moyen (en secondes)	46.1 $\pm$ 2.4	**56.0** $\pm$ 2.0	10.9 $\pm$ 0.5

Comme on pouvait s'y attendre le bagging de SVM à noyau polynomial ne tient pas la comparaison avec les RandomForest (tableau 8) et se fait largement dépasser sur quasiment tous les datasets à l'exception de Abalone20 pour lequel les RandomForest échouent entièrement avec un score F1 de 0 et de German. Néanmoins il reste moins performant en moyenne et c'est la RandomForest pondérée par le coût qui obtient les meilleurs résultats avec un score F1 moyen de 65.2. Du point de vue du temps d'apprentissage la conclusion est la même, le bagging de SVM prend en moyenne 46.1 secondes à s'entrainer contre à peine 1.7 secondes pour la RandomForest pondérée par le coût. On notera que la RandomForest est la première méthode à dépasser les performances des γk-NN.

TABLE 8 – F1 score moyen en test sur 10 itérations RandomForest avec variante et bagging de SVM polynomial

Dataset	RandomForest	RandomForest pondérée par le coût	Bagging SVM polynomial
Abalone20	0.0 ± 0.0	0.0 ± 0.0	**8.3** $\pm$ 2.7
Yeast6	38.0 ± 21.3	**51.5** $\pm$ 17.8	33.3 ± 6.9
German	46.1 ± 6.1	41.2 ± 5.9	**62.8** $\pm$ 3.4
Newthyroid	94.4 ± 3.4	**95.0** $\pm$ 2.4	78.2 ± 6.3
Wine	93.3 ± 5.6	**94.7** $\pm$ 3.6	86.4 ± 7.6
Iono	92.6 ± 3.4	**93.2** $\pm$ 3.8	91.1 ± 3.9
Heart	**82.1** $\pm$ 4.7	81.0 ± 4.9	68.7 ± 6.5
Moyenne	63.8 ± 6.4	**65.2** $\pm$ 5.5	61.3 ± 5.3
Temps moyen (en secondes)	1.9 ± 0.2	1.7 ± 0.2	**46.1** $\pm$ 2.4

TABLE 9 – F1 score moyen en test sur 10 itérations

Dataset	AdaBoost	AdaCost	AdaBoost Régression Logistique
Abalone20	0.0 ± 0.0	**6.7** $\pm$ 4.0	2.1 ± 0.4
Yeast6	**45.3** $\pm$ 17.5	41.8 ± 12.0	23.8 ± 3.7
German	51.8 ± 6.0	**57.2** $\pm$ 4.3	45.9 ± 3.4
Newthyroid	**94.4** $\pm$ 3.4	92.5 ± 5.0	75.8 ± 9.3
Wine	**93.2** $\pm$ 5.3	91.9 ± 4.2	80.8 ± 7.9
Iono	**87.6** $\pm$ 4.2	87.4 ± 4.0	86.7 ± 5.0
Heart	76.2 ± 5.2	**79.4** $\pm$ 5.7	64.8 ± 10.5
Moyenne	64.1 ± 5.9	**65.3** $\pm$ 5.6	54.3 ± 5.7
Temps moyen (en secondes)	2.2 ± 0.2	4.5 ± 0.5	**5.5** $\pm$ 0.3

Finalement on considére les résultats du boosting avec AdaBoost, AdaCost et AdaBoost sur la régression logistique (le choix de la régression logistique a été fait car l'apprentissage de AdaBoost requiert une méthode qui renvoie une probabilité d'appartenance) 9. Cette fois-ci c'est AdaCost qui offre les meilleures performances avec un score F1 de 65.3 contre 64.1 pour AdaBoost et 54.3 pour AdaBoost sur la régression logistique. En revanche les temps d'apprentissage sont tous les trois assez similaires, quand on les mets en perspectives des différences constatées pour le bagging, allant de 2.2 secondes en moyennes à 5.5. AdaCost se révéle donc être la méthode la plus performante parmis les méthodes non linéaires mais également parmis toutes les autres. Cela tout en conservant un temps d'apprentissage tout à fait supportable de 4.5 secondes en moyenne.

4 Conclusion

Dans cette étude nous avons proposé plusieurs variantes d'algorithmes classiques de machine learning supervisé pour la classification ceci avec une certaine concentration sur les cas où les données sont déséquilibrées. Toutes ces variantes ne sont pas révélées compétitives avec les méthodes initiales mais dans chaque cas l'une au moins d'entre elles à su améliorer significativement les performances. Dans le cas des k-NN on a une franche amélioration grâce au rééchantillonnage SMOTE ainsi qu'au γk-NN. Puis, pour les méthodes linéaires on a à nouveau un progrés grâce à SMOTE mais également par la prise en compte d'une pondération par le poids. Enfin, pour les méthodes linéaires, bien que

TABLE 10 – F1 score moyen en test sur 10 itérations des meilleures méthodes pour chaque approche (γk-NN, SVM polynomial pondéré, AdaCost)

Dataset	γk-NN	SVM polynomial pondéré	AdaCost
Abalone20	4.7 ± 2.6	**7.3** ± 2.2	6.7 ± 4.0
Yeast6	**49.0** ± 13.2	27.8 ± 3.1	41.8 ± 12.0
German	48.2 ± 2.5	**62.5** ± 3.3	57.2 ± 4.3
Newthyroid	**94.1** ± 3.9	76.2 ± 8.6	92.5 ± 5.0
Wine	86.3 ± 5.1	86.4 ± 7.4	**92.2** ± 4.1
Iono	**92.1** ± 3.5	91.9 ± 3.0	87.4 ± 4.0
Heart	65.3 ± 4.0	67.5 ± 9.2	**79.2** ± 5.9
Moyenne	62.8 ± 5.0	59.9 ± 5.3	**65.3** ± 5.6
Temps moyen	0.8 ± 0.0	0.6 ± 0.0	**4.2** ± 0.3

le replacement des sous-modèles par des SVM ou des régressions logistique ne ce soit pas révélé pertinent, la prise en compte d'une pondération relative au déséquilibre dans les données à eu un effet positif. Le tableau 10 récapitule les performances des meilleures méthodes pour chaque partie. AdaCost obtient les meilleurs résultats mais il est intéressant de noter que les γk-NN obtient de meilleures performance que la meilleure méthode linéaire à savoir le SVM à noyau polynomial pondérée par le coût, mais aussi les meilleures performances en général sur trois datasets (Yeast6, Newthyroid, Iono). Ce qui est encourangeant pour les capacités des méthodes non paramètriques à être compétitives avec des méthodes bien plus sophistiquées.

De nombreuses perspectives sont imaginables à l'aune de ces conclusions. On pourrait considérer des méthodes de rééchantillonnages plus sophistiquées comme un mélange de sous-échantillonnage et de SMOTE (comme proposé par les auteur·es). Pour les k-NN et à la maniére des γk-NN il serait intéressant de réfléchir à d'autres mesures de distances adaptives au déséquilibre dans les données avec par exemple un γ proportionnel à la proportion de données minoritaires. Mais il serait également possible de considérer d'autres façons de considérer la distance entre les observations qui serait plus pertinente. Sur la même idée on pourrait employer des noyaux directement construit pour répondre à des situations de données déséquilibrées, pour cela on se tournera par exemple vers l'article [Maratea and Petrosino, 2011]. Enfin, et bien que cela dépasse le cadre de ce travail, l'usage de méthode d'apprentissage profond pourrait avoir un impact extrêmement positif pour la tâche considérée ici. Il serait possible d'utiliser des Generative Adversarial Networks pour générer de nouvelles données de la classe minoritaire, à l'image de SMOTE mais avec plus de raffinement. Mais également on peut imaginer des réseaux de neurones convolutifs qui seraient adaptés pour de la classification supervisée et dont on peut supposer qu'ils obtiendraient des résultats largement satisfaisants.

Références

[Chawla et al., 2002] Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). Smote : synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16 :321–357.

[Elkan, 2001] Elkan, C. (2001). The foundations of cost-sensitive learning. In *International joint conference on artificial intelligence*, volume 17, pages 973–978. Lawrence Erlbaum Associates Ltd.

[Fan et al., 1999] Fan, W., Stolfo, S. J., Zhang, J., and Chan, P. K. (1999). Adacost : misclassification cost-sensitive boosting. In *Icml*, volume 99, pages 97–105.

[Maratea and Petrosino, 2011] Maratea, A. and Petrosino, A. (2011). Asymmetric kernel scaling for imbalanced data classification. In *International Workshop on Fuzzy Logic and Applications*, pages 196–203. Springer.

[Viola et al., 2021] Viola, R., Emonet, R., Habrard, A., Metzler, G., Riou, S., and Sebban, M. (2021). A nearest neighbor algorithm for imbalanced classification. *International Journal on Artificial Intelligence Tools*, 30(03) :2150013.