Ingénierie de la représentation des variables pour la classification binaire à partir des données déséquilibrées

Mémoire

Nora Ajakan

Maîtrise en informatique - avec mémoire
Maître ès sciences (M. Sc.)

Québec, Canada

Ingénierie de la représentation des variables pour la classification binaire à partir des données déséquilibrées

Mémoire

Nora Ajakan

Sous la direction de:

Richard Khoury, directeur de recherche

Résumé

De nombreuses applications de classification binaire, telles que la prédiction de fraude et la prédiction de rétention, impliquent des ensembles de données déséquilibrées. Bien que les méthodes d'ensemble soient les mieux adaptées à ces contraintes, les règles de décision produites sont difficiles à interpréter en tant que groupe en raison de leur nombre et de leurs redondances sous-jacentes. Il est donc intéressant de simplifier les méthodes d'ensemble apprises en un petit ensemble équivalent de conditions sans sacrifier la performance à la simplicité. En interprétant simplement un arbre de décision comme un empilement de fonctions indicatrices binaires et un modèle linéaire, nous proposons une méthode qui apprend le sous-ensemble efficace d'indicateurs qui relie les données à un espace de représentation de faible dimension où elles deviennent linéairement séparables. Ces fonctions binaires permettent à un large éventail d'algorithmes d'apprentissage automatique simples d'être efficaces et sont également plus faciles à analyser, à étudier ou à valider par les experts du domaine que les branches initiales de l'arbre dans l'ensemble appris.

Abstract

Many binary classification applications, such as churn prediction and fraud detection, involve unbalanced large datasets. While ensemble trees are the most suited algorithms given these constraints, the decision rules produced are hard to interpret as a group due to their number and their underlying redundancies. It is then of interest to simplify the learned ensemble trees into a small equivalent set of conditions without trading performance for simplicity. By simply interpreting a decision tree as a stack of binary indicator functions and a linear model, we propose a method that learns the effective subset of indicators that map the data to a low dimension feature space where it becomes linearly separable. These binary functions enable a wide range of simple machine learning algorithms to be efficient and are also easier to analyze, investigate or validate by domain experts than the initial tree branches in the learned ensemble.

Table des matières

Liste des tableaux

Liste des figures

Liste des abréviations

LCASF La Capitale Assurances et Services Financiers

SVM Méthodes à vecteurs de support

XGBoost *eXtreme Gradient Boosting*

Adaboost *Adaptive Boosting*

PCA Analyse en composantes principales (*Principal Components Analysis*)

RCA Analyse des composants pertinentes (*Relevant Components Analysis*)

Remerciements

Je tiens à m'acquitter d'une dette de reconnaissance auprès de toutes les personnes qui ont contribué de près ou de loin à la réalisation de ce mémoire.

Je tiens tout d'abord à exprimer ma reconnaissance envers mon directeur de recherche Richard Khoury pour son accompagnement, sa disponibilité et ses conseils judicieux tout au long de ma maîtrise. Je le remercie également pour sa patience, ses encouragements et la confiance qu'il m'a accordé. Ainsi que pour toutes nos discussions constructives qui m'ont aidé à forger un meilleur esprit d'analyse et ont contribué à alimenter ma réflexion.

Je désire aussi remercier mes collègues au GRAAL pour leur précieuse collaboration. Merci à Khalil pour nos discussions et échanges constructifs qui ont contribué à la réalisation de ce mémoire. Merci à Hana, Prudencio,Mehdi, Mazid, Rogia, Talia et Mehdi pour nos échanges, leur aide inconditionnelle ainsi que l'ambiance de bienveillance qu'ils créent.

Merci aux examinateurs de mon mémoire, Pascal Germain et Christophe Pere pour avoir accepté de juger mon travail. Leurs suggestions ont permis la bonification de ce mémoire.

Je tiens à remercier et à dédier ce travail à mes parents, Aicha et Said. À qui je suis redevable pour ce que je suis aujourd'hui et je ne saurais les remercier assez pour les sacrifices qu'ils ont faits à mon égard.

Je remercie ma famille pour leur soutien.. Merci à Hana et à Khalil pour leur présence et support tout au long des épreuves que j'ai surmontées. Un grand merci à mon neveu Youssef pour son amour inconditionnel et son aura de joie et à mon frère Mohamed pour son humour et bienveillance.

Introduction

Le secteur de l'assurance est extrêmement compétitif. Au Canada, plus de 156 assureurs offrent des protections. Cette panoplie de choix facilite aux assurés la comparaison et la recherche des meilleures offres. L'acquisition de nouveaux clients et la rétention des clients actuels sont donc des défis majeurs pour les compagnies d'assurance. Or selon une étude de l'agence indépendante d'assurance Thomas (2020), le coût de rétention d'un ancien client est de 7 à 8 fois moins cher que l'acquisition d'un nouveau client.

Il est donc primordial pour les compagnies d'assurance de bien comprendre et modéliser les comportements des clients à travers des stratégies proactives permettant de détecter le plus tôt possible les clients à haut risque de résiliation, afin que la compagnie puisse diriger ses efforts vers eux.

Dans le cadre de ce mémoire, les données de la compagnie La Capitale Assurances et Services Financiers (LCASF) sur le secteur de l'assurance vie temporaire sont utilisées. Selon LCASF, le pourcentage de résiliation dans ce secteur est aux alentours de 7% par an. De par la rareté des données d'intérêt, dans notre cas les résiliations comparées aux renouvellements, la rétention de la clientèle est considérée comme un problème de traitement de données déséquilibrées.

L'entreprise a collecté des données formées par des descriptions de clients et leurs résultats de résiliation. Ce résultat est une réponse « oui » ou « non » à la question de savoir si un client avait arrêté son contrat. On se propose alors de créer un détecteur de cas de résiliation. Deux approches s'imposent, à savoir la détection d'anomalies, et la classification binaire. L'investigation des données collectées a démontré que les observations de résiliations se séparent en deux groupes : des anomalies ou des cas extrêmes facilement détectables par un détecteur d'anomalies, et des observations clairsemées dans le domaine des observations de non-résiliation. Dans ce dernier cas, les données d'une résiliation risquent d'être très similaires à celles de données voisines de prolongement. Et donc, elles ne sont pas adaptées aux détecteurs d'anomalies. Nous avons alors opté pour la classification binaire.

Une méthode axée sur la production de modèles simples et performants est proposée dans ce mémoire. Elle consiste à apprendre des combinaisons linéaires favorisant l'apprentissage d'un ensemble d'arbres de décision efficaces et à faibles profondeurs et à en déduire un ensemble

réduit de branches et de transformations linéaires les plus utiles pour la prédiction.

Le présent travail est constitué de quatre chapitres ainsi que de l'introduction générale, l'introduction au domaine d'affaires et la conclusion.

Le **Chapitre 1** est dédié aux prérequis pour la compréhension des chapitres qui s'en suivent. Il porte sur la classification binaire supervisée, les fonctions objectives puis des exemples de classificateurs. Ensuite, la présentation des métriques d'évaluation pertinentes au cas étudié ainsi que les méthodes de peaufinage des paramètres et des attributs.

Le **Chapitre 2** est une présentation des données déséquilibrées. Il discute de leurs impacts sur la tâche de classification, leurs propriétés intrinsèques ainsi qu'une revue des techniques de traitement de classification binaire basée sur des données déséquilibrées.

Le **Chapitre 3** présente notre approche. Ce chapitre porte sur le choix des métriques d'évaluation, de la famille de classificateurs ainsi que les transformations des données. Ensuite, l'approche d'apprentissage adaptatif de projections en combinant PCA pondéré et XGBoost et la méthode proposée pour la simplification du modèle sont présentées.

Le **Chapitre 4** porte sur les résultats des expériences effectuées en comparant les résultats des algorithmes ainsi que l'analyse de leurs comportements. Ceci est suivi par les résultats de la simplification du meilleur modèle, l'analyse de la représentation apprise et l'interprétabilité ou la compréhension des données au sens du modèle.

Domaine d'affaires

Le domaine de l'assurance représente un marché financier très important au Canada. Ceci est reflété notamment par l'augmentation régulière du volume annuel des primes d'assurance achetées, tel qu'illustré à la figure 0.1. D'après l'Association canadienne des compagnies d'assurances de personnes (ACCAP, 2018), en 2017 presque 29 millions de canadiens ont souscrit à une police d'assurance, ce qui a représenté plus de 110 milliards de dollars en primes d'assurance et 92 milliards de dollars en bénéfices payés aux clients.

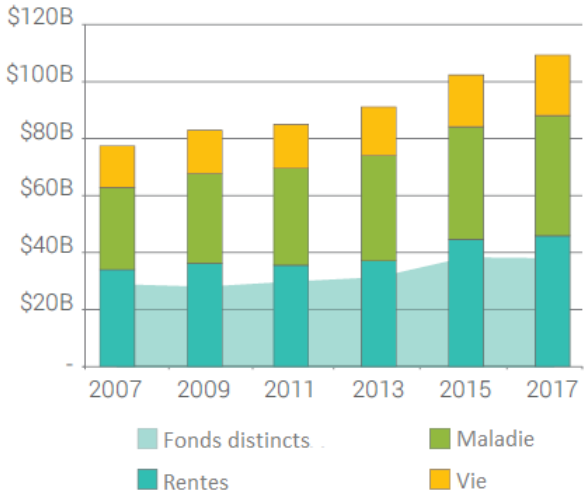


FIGURE 0.1 – L'évolution des primes d'assurance au Canada (extrait de ACCAP (2018))

Le secteur de l'assurance vie en particulier, tant collective qu'individuelle, représentait plus de 20 milliards de dollars en primes en 2017 et 4 730 milliards de dollars en couverture de protection, tel qu'illustré à la figure 0.2. De cette couverture, seulement 39% est associée à un programme d'assurance collective, et 61% est en assurance individuelle. Les clients se procurent une assurance vie individuelle afin de fournir une protection financière à leurs familles en cas de décès. Le retour de cette prime peut être utilisé par exemple pour subvenir aux besoins de la famille, compenser une perte de revenu, ou payer une dette. Il existe deux types d'assurance-vie individuelle :

— Assurance vie permanente : une protection à vie qui offre des avantages aux bénéficiaires

en cas de décès de l'assuré et permet également d'accumuler les primes pour utilisation en cas d'urgence financière.

— Assurance vie temporaire : un produit qui offre une couverture pour une durée prédéterminée (10, 15, 20, 25, 30, ou 35 ans), ce qui permet à l'assuré d'ajuster sa couverture en fonction de l'évolution de ses besoins financiers.

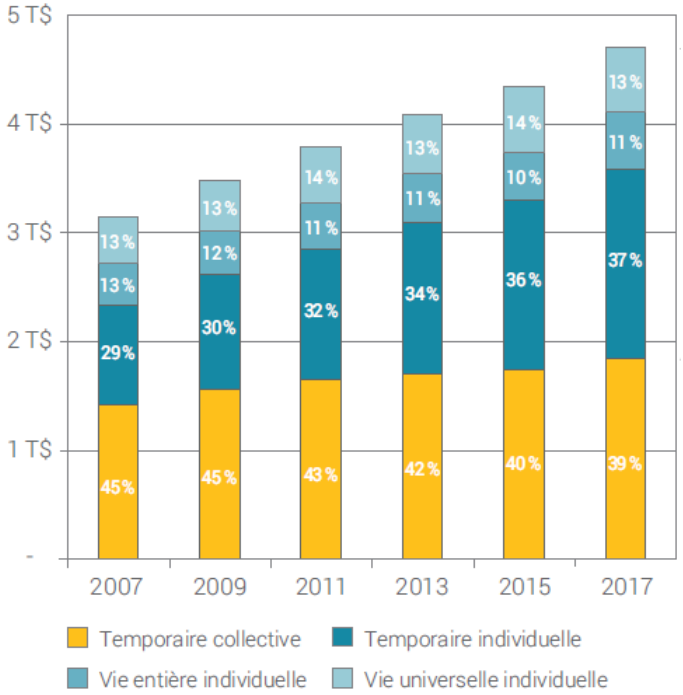


FIGURE 0.2 – L'évolution des couvertures d'assurance vie au Canada, extrait de ACCAP (2018)

Le secteur de l'assurance étant extrêmement compétitif, la rétention de clients est un enjeu majeur pour ce secteur, d'autant plus que selon Thomas (2020) l'industrie de l'assurance a le coût d'acquisition de clients le plus élevé comparé à d'autres secteurs d'activité. Et comme mentionné précédemment, le coût de rétention d'un client est de 7 à 8 fois moins cher que l'acquisition d'un nouveau client. La rétention des clients actuels est donc primordiale à la profitabilité des compagnies d'assurance. Le défi de gestion de la rétention représente l'ensemble des efforts réalisés par une compagnie d'assurance afin de fidéliser ses clients actuels, et son succès se reflète par l'intention de ses clients à renouveler leurs contrats.

Afin d'avoir une gestion de rétention efficiente, il est important de comprendre les facteurs qui poussent le client à résilier ou à ne pas renouveler son contrat d'assurance. La figure 0.3 montre les raisons principales qui influencent la décision du client, selon une étude menée par Assurland (2019). Il existe trois facteurs principaux expliquant la résiliation d'un contrat :

— Le client a trouvé une offre plus avantageuse en termes de prix et/ou de qualité chez un compétiteur.

— Le client a été conquis par une meilleure stratégie de ventes et/ou de commercialisation d'un compétiteur.

— Le client est insatisfait de son expérience avec la compagnie actuelle.

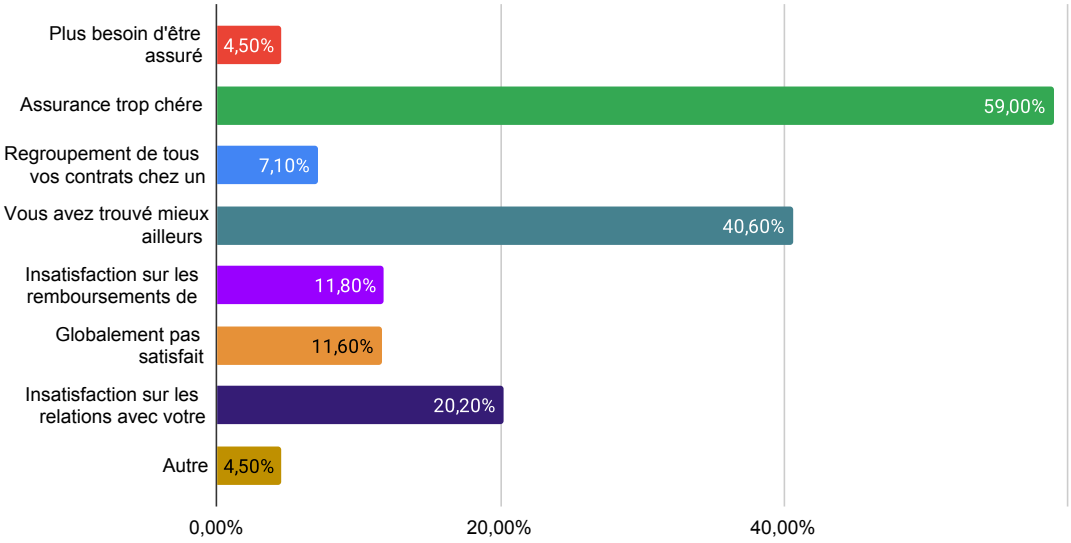


FIGURE 0.3 – Les raisons principales de résiliation des contrats d'assurance extrait de Assurland (2019)

Grâce aux banques de données dont disposent les compagnies d'assurances, portant sur les historiques des interactions, les descriptions de clients, etc., il est possible de développer des méthodes de prédictions préventives afin de cibler les clients ayant un haut risque de résiliation et leur fournir un meilleur service.

Chapitre 1

Pré-requis

Introduction

L'apprentissage automatique est un sous-domaine de l'intelligence artificielle qui consiste à utiliser des mathématiques et statistiques afin de rendre les machines capables d'apprendre à réaliser des tâches en utilisant des données. Contrairement à la programmation classique où les étapes de la réalisation de la tâche sont spécifiées avec des instructions, un algorithme d'apprentissage automatique consomme les données et extrait automatiquement des étapes équivalentes.

1.1 La classification binaire supervisée

Les sous-domaines de l'apprentissage automatique diffèrent selon le type de l'observation et leur structure. En particulier, en apprentissage automatique supervisé, une observation collectée par le passé est formée par la solution de la tâche, et des données utiles en lien avec les solutions. Ces composantes sont appelées une étiquette (ou sortie) et une instance (ou donnée en entrée), respectivement. Lorsque l'étiquette est scalaire, on parle de régression. Lorsqu'elle est discrète (une catégorie), on parle de classification. En particulier, il s'agit de la classification binaire lorsque la sortie appartient à un ensemble à deux catégories.

Dans le cas particulier de rétention de la clientèle, l'étiquette est une réponse "oui" ou "non" à la question : "est-ce que ces données correspondent à un client qui désiste ?". La tâche est donc une classification binaire. Toutefois, elle peut être formulée comme une régression si on souhaite déterminer la probabilité (scalaire dans l'intervalle $[0; 1]$) d'abandon. On peut imaginer une instance comme des attributs décrivant le profil du client, ses contrats, son interaction avec l'agence et son historique d'indemnisations.

Notons $\mathcal{X}$ le domaine des instances et $\mathcal{Y}$ le domaine des étiquettes. Dans le cadre de la classification binaire, il est courant de choisir $\mathcal{X} = \mathbb{R}^m$ pour $m \in \mathbb{N} \setminus \{0\}$ et $\mathcal{Y} = \{\pm 1\}$ ou bien

$\mathcal{Y} = \{0, 1\}$. On suppose qu'il existe une relation $f : \mathcal{X} \longrightarrow \mathcal{Y}$ entre les instances et les étiquettes et on souhaite la modéliser (ou estimer son approximation) par une hypothèse $h : \mathcal{X} \longrightarrow \mathcal{Y}$. La fonction h est communément appelée modèle et elle minimise idéalement la vraie erreur de classification E_{0-1}.

$$E_{0-1}(\mathcal{X}, h) = \mathbb{E}_{\mathcal{X}}[h(\mathbf{x}) = f(\mathbf{x})] \tag{1.1}$$

En pratique, il est impossible d'évaluer $E_{0-1}(\mathcal{X}, h)$ car f n'est pas accessible. Il est en effet impossible de collecter des étiquettes pour toutes les observations possibles. On a cependant un ensemble fini de $n \in \mathbb{N} \setminus \{0\}$ observations indépendantes et identiquement distribuées $S = \{(\mathbf{x}_i, y_i),\ \mathbf{x}_I \in \mathcal{X}, y_i \in \mathcal{Y}, i \in \{1, .., n\}\}$. La tâche d'apprentissage consiste alors à produire un modèle h qui minimise l'erreur $E_{0-1}(S, h)$. Il s'en suit que l'évaluation de cette erreur serait minimale et elle est donc une estimation biaisée de $E_{0-1}(\mathcal{X}, h)$. On considère alors $E_{0-1}(\mathcal{T}, h)$ sur un échantillon de test $\mathcal{T}$ indépendant de S et qui est inutilisé dans la phase d'apprentissage. $E_{0,1}(\mathcal{T}, h)$ est une estimation non biaisée de $E_{0-1}(\mathcal{X}, h)$. En résumé, la production d'un classificateur binaire consiste à utiliser les observations collectées par le passé $\mathcal{S}$ pour apprendre un modèle h qui performe bien, selon les métriques d'évaluation choisies, dans la classification de nouvelles instances non vues durant l'entraînement.

1.2 Fonctions objectifs en classification binaire

L'apprentissage d'un classificateur est effectué par minimisation de l'erreur $E_{0,1}(\mathcal{S}, h)$. La non-différentiabilité de cette fonction limite les choix des algorithmes d'optimisation à ceux basés sur la recherche exhaustive (souvent efficace modulo une structuration de données) et aux méthodes numériques heuristiques pour l'optimisation non linéaire où la dérivée est inconnue (descente du simplex). Toutefois, afin de bénéficier des algorithmes d'optimisation exploitant les dérivées (solutions exactes, descente du gradient, algorithme du simplex), il est courant de remplacer l'erreur de classification par une fonction différentiable par rapport aux paramètres du classificateur et qui est idéalement légèrement supérieure à l'erreur de classification. On appelle alors cette fonction erreur alternative ou bien fonction objective. Cette section liste les fonctions objectives les plus utilisées en classification binaire.

1.2.1 Fonction de perte

Une fonction de perte $\ell : \mathcal{Y} \times \mathcal{Y} \longrightarrow \mathbb{R}_+$ est une fonction qui associe une pénalité réelle positive à l'écart (au sens d'un critère) entre la prédiction $\hat{y}$ fournie par le classificateur et la vraie étiquette y observée durant la phase de l'entraînement. La pénalité la plus importante est associée aux écarts les plus importants alors que les écarts les plus faibles ont souvent une pénalité nulle ou légèrement supérieure à zéro.

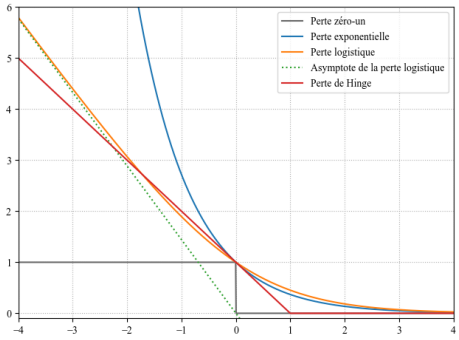


FIGURE 1.1 – Illustration des différentes pertes

L'erreur de classification correspond à la perte zéro-un notée ℓ_{0-1}.

$$\begin{aligned}\forall (y_1, y_2) \in \mathcal{Y} \times \mathcal{Y}, \ell_{0-1}(y_1, y_2) &= \mathbb{1}[y_1 = y_2] \\ &= 1 \text{ si } y_1 \neq y_2 \text{ et } 0 \text{ sinon.}\end{aligned}$$

$E_{0,1}(\mathcal{S}, h)$ s'écrit comme une somme de la perte zéro-un évaluée sur toutes les observations de $\mathcal{S}$.

$$E_{0,1}(\mathcal{S}, h) = \sum_{(\mathbf{x},y)\in\mathcal{S}} \ell_{0-1}\big(y, h(\mathbf{x})\big) = \sum_{(\mathbf{x},y)\in\mathcal{S}} \ell_{0-1}\big(y, \hat{y}\big)$$

Une fonction objective alternative est obtenue en remplaçant la fonction de perte ℓ_{0-1} par une fonction qui la borne supérieurement pour tout $(y_1, y_2) \in \mathcal{Y} \times \mathcal{Y}$. Le paragraphe suivant illustre les fonctions de perte usuelles en classification binaire.

1.2.2 Fonctions de pertes alternatives

Soit $\mathcal{Y} = \{\pm 1\}$ et une fonction $g : \mathcal{X} \longrightarrow \mathbb{R}$. On distingue deux approches principales permettant de dériver h un classificateur binaire à partir de g. La première consiste à prendre le signe de g afin de produire des sorties discrètes dans $\{\pm 1\}$.

$$\forall \mathbf{x} \in \mathcal{X}, h(\mathbf{x}) = signe\big(g(\mathbf{x})\big)$$

La deuxième consiste à transformer les valeurs de g, communément appelées logits, en des scores dans $[0, 1]$ simulant la probabilité, au sens de g, d'appartenance de l'instance $\mathbf{x}$ à la classe 1. La fonction sigmoïde σ est une des transformations générant un tel score.

$$\forall \mathbf{x} \in \mathcal{X}, \ P[y = 1|g, \mathbf{x}] = \sigma\big(g(\mathbf{x})\big) \in [0, 1]$$

Avec

$$\forall x \in \mathbb{R}, \ \sigma(x) = \frac{1}{1 + e^{-x}}$$

La fonction h peut alors être formulée comme un seuil sur les valeurs de la probabilité $P[y(\mathbf{x}) = 1|g, \mathbf{x}]$. Par exemple :

$$\forall \mathbf{x} \in \mathcal{X}, \ h(\mathbf{x}) = 1 \text{ si } P[y = 1|g, \mathbf{x}] > 0.5 \text{ et } -1 \text{ sinon.}$$

Dans les deux cas, la fonction de perte ℓ_{0-1} s'exprime en fonction de g comme suit.

$$\begin{aligned}\forall(\mathbf{x},y)\in\mathcal{S},\ \ell_{0-1}(y,h(\mathbf{x})) &= \mathbb{1}[h(\mathbf{x})=y]\\ &= \mathbb{1}[y\cdot g(\mathbf{x})>0]\end{aligned}$$

On peut donc penser à la fonction de perte alternative comme une fonction associant une pénalité supérieure à 1 lorsque $y.g(\mathbf{x})$ est négatif, et une perte positive dans $[0,1[$ sinon. De plus, il est préférable d'avoir une perte décroissante en fonction de $y.g(\mathbf{x})$.

Perte exponentielle

La perte exponentielle est définie comme suit.

$$\forall(\mathbf{x},y)\in\mathcal{S},\ \ell_e\big(y,g(\mathbf{x})\big)=e^{-y.g(\mathbf{x})}$$

Lorsque $g(\mathbf{x})$ est de même signe que y, la perte exponentielle associe une faible pénalité (exponentielle d'un argument négatif). Toutefois, elle associe une pénalité très importante et qui varie en $e^{|g(\mathbf{x})|}$ le cas échéant.

Perte de Hinge

La perte de Hinge est définie comme suit.

$$\forall(\mathbf{x},y)\in\mathcal{S},\ \ell_H\big(y,g(\mathbf{x})\big)=max(0,1-y\cdot g(\mathbf{x})\big)$$

De manière analogue à la perte exponentielle, la perte de Hinge associe une pénalité nulle lorsque y et $g(\mathbf{x})$ sont de même signe et que $|g(\mathbf{x})|\geq 1$ et associe une pénalité supérieure à 1 si $y\cdot g(\mathbf{x})<0$ qui est croissante en $|g(\mathbf{x})|$.

Perte logistique binaire

La perte logistique binaire est définie comme suit.

$$\forall(\mathbf{x},y)\in\mathcal{S},\ \ell_{log}\big(y,g(\mathbf{x})\big)=log\big(1+e^{-y\cdot g(\mathbf{x})}\big)$$

1.3 Exemples de classificateurs binaires

1.3.1 Seuil de décision

Un seuil de décision (voir par exemple Shalev-Shwartz and Ben-David (2014)) prédit une valeur dans $\{\pm 1\}$ en évaluant si une composante de l'instance dépasse un certain seuil. Ce classificateur est régi par trois paramètres : le seuil $\theta\in\mathbb{R}$, $j\in\{1,...,m\}$ l'indice de la composante à évaluer, et $b\in\{\pm 1\}$ indiquant si la prédiction $+1$ est associée ou non à des instances où la composante j dépasse θ.

$$\forall b\in\{\pm 1\}, j\in\{1,...,m\}, \theta\in\mathbb{R} \text{ et } \mathbf{x}\in\mathbb{R}^m,\ h_{b,j,\theta}(\mathbf{x})=b.signe(\theta-\mathbf{x}[j])$$

Selon son expression, un seuil de décision divise l'espace des instances en deux partitions disjointes, définie par la frontière $\mathbf{x}[j] = \theta$ et associe une étiquette binaire à chacune des partitions. La prédiction consiste alors à associer à une instance, l'étiquette de la partition où elle se trouve.

L'apprentissage de ce classificateur s'effectue efficacement par recherche exhaustive sur des données triées par colonne (Shalev-Shwartz and Ben-David, 2014). C'est-à-dire, pour une colonne j :

$$\mathbf{x}_1[j] \leq \mathbf{x}_2[j] \leq ... \leq \mathbf{x}_n[j]$$

L'algorithme d'apprentissage débute avec un seuil $\theta = -\infty$, classifiant tous les exemples à -1 et évalue la fonction objective. Ensuite, les exemples sont examinés itérativement et à chaque itération, on déplace la frontière au milieu de deux exemples successifs. Il en découle qu'à chaque itération, un seul exemple change de côté par rapport à la frontière. Et donc, un seul terme de la fonction objective sera mis à jour. En pratique, on soustrait l'ancienne contribution de l'exemple à la fonction objective et on lui ajoute la nouvelle évaluation de la fonction de perte après changement d'étiquette.

$$\forall i \in \{1, ..., n\}, E_{i+1} = E(\mathcal{S}, \theta_i = \frac{\mathbf{x}_i[j] + \mathbf{x}_{i+1}[j]}{2}) = E_i - w_i.\ell(y_i, -1) + w_i.\ell(y_i, +1)$$

Où $\mathbf{x}_{n+1}[j] = \mathbf{x}_n[j] + 1$. θ est alors choisie à partir de l'itération pour laquelle la fonction objective est minimale. En pratique, pour bénéficier de l'astuce de tri, on calcule pour toute colonne j, une permutation $\pi : \{1, ..., n\} \longrightarrow \{1, ..., n\}$ permettant d'ordonner les instances d'entraînement selon la colonne j. En d'autres termes :

$$\mathbf{x}_{\pi(1)}[j] \leq \mathbf{x}_{\pi(2)}[j] \leq ... \leq \mathbf{x}_{\pi(n)}[j]$$

Il en découle que seules les permutations des indices sont stockées. Cela permet d'éviter de stocker les colonnes ordonnées et les étiquettes correspondantes à chaque colonne permutée et surtout d'éviter de réordonner un sous échantillon de $\mathcal{S}$. Cela permet entre autres la construction efficace d'arbres de décision.

1.3.2 Arbre de décision

Un arbre de décision (voir par exemple Steinberg (2009)) partitionne l'espace des attributs $\mathcal{X}$ en des partitions disjointes séparées par des frontières de type seuil de décision. Il est en général construit de manière incrémentale à partir d'un seuil de décision initiale. Initialement, l'arbre est un noeud prédisant une valeur constante pour tout $\mathbf{x} \in \mathcal{X}$. La partition initiale est donc $\mathcal{X}$. Ensuite à chaque itération, on examine si on peut atténuer la fonction objective en subdivisant chaque région produite en deux régions disjointes, via un seuil de décision.

Les seuils de décision et les valeurs prédites sont structurés dans un arbre où les seuils occupent les noeuds internes et les valeurs de sorties associées aux régions sont stockées aux feuilles (noeuds terminaux). Les noeuds internes évaluent si une instance vérifie une condition (définie par un seuil de décision). Si tel est le cas, le noeud enfant à droite sera évalué, sinon, le noeud à gauche. On répète le même processus si l'enfant est un noeud interne. S'il est une feuille, la prédiction pour l'instance sera la valeur associée à la feuille.

La fonction de décision d'un arbre de décision à $N \in \mathbb{N} \setminus \{0\}$ feuilles est modélisée par une fonction $q : \mathcal{X} \longrightarrow \{1, ..., N\}$ et un vecteur $\mathbf{w}$ de N composantes, stockant les sorties pour chaque feuille. q associe à chaque instance l'indice de la partition (ou la feuille) qui l'inclut. La prédiction est alors définie par :

$$\forall \mathbf{x} \in \mathcal{X}, g(\mathbf{x}) = \mathbf{w}[q(\mathbf{x})]$$

1.3.3 Séparateur linéaire

En classification binaire, un séparateur linéaire est défini comme une combinaison linéaire des attributs, passée en argument à une fonction (typiquement non linéaire) afin d'avoir des valeurs discrètes dans $\{0, 1\}$ ou $\{\pm 1\}$, ou bien des valeurs scalaires entre $[0, 1]$ dans le cas de la régression logistique (voir par exemple Shalev-Shwartz and Ben-David (2014)). En s'alignant avec les notations de la section 1.3.2, la fonction de décision g du modèle linéaire est définie en fonction d'un vecteur $\mathbf{w} \in \mathbb{R}^m$ et d'un biais $b \in \mathbb{R}$ comme suit :

$$\forall \mathbf{x} \in \mathcal{X}, g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$$

L'apprentissage des paramètres $\mathbf{w}$ et b peut s'effectuer itérativement en utilisant l'algorithme du perceptron, ou par descente du gradient pour la minimisation du log-vraisemblance ou de l'erreur logistique.

Le produit scalaire $\mathbf{w}^T \mathbf{x}$ fait en sorte que le gradient de la fonction objective pour une instance $\mathbf{x}_i$ est colinéaire à $\mathbf{x}_i$. Il s'ensuit que $\mathbf{w}$ peut s'exprimer comme une combinaison linéaire des instances d'entraînement.

$$\exists (\alpha_1, \alpha_2, ..., \alpha_n) \in \mathbb{R}^n, \mathbf{w} = \sum_{i=1}^{n} \alpha_i . \mathbf{x}_i$$

Les méthodes à vecteurs de supports (SVM) identifient les coefficients α_i en minimisant l'erreur de Hinge régularisée L_2 pour favoriser la convergence vers des $\mathbf{w}$ de faible norme.

On observe également que la fonction de décision d'un noeud interne d'un arbre de décision peut s'exprimer comme un séparateur linéaire. En effet, la fonction de décision correspondante

s'écrit comme suit.

$$\begin{aligned}\exists j \in \{1,...,m\},\ \theta \in \mathbb{R},\ \text{tel que } \forall \mathbf{x} \in \mathcal{X},\ g(\mathbf{x}) &= \theta - \mathbf{x}[j] \\ &= (0,...,0,-1,0,...,0).\mathbf{x} + \theta \\ &= \mathbf{w}^T\mathbf{x} + b\end{aligned}$$

Avec $\mathbf{w}$ un vecteur de $\mathbb{R}^m$ de composantes nulles sauf à la position j, où $\mathbf{w}[j] = -1$, et $b = \theta$. Il s'ensuit qu'il est possible d'optimiser le biais d'un séparateur linéaire en apprenant un noeud interne d'un arbre de décision de profondeur unité et en utilisant les projections $\mathbf{w}^T\mathbf{x}$ à la place des instances. De plus, on peut même partitionner la droite $\mathbb{R}$ en des intervalles disjoints permettant de mieux séparer les exemples d'entraînement selon les valeurs de $\mathbf{w}^T\mathbf{x}$ en apprenant un arbre de décision de profondeur supérieure à 1.

1.3.4 Modèle linéaire généralisé

Le modèle linéaire généralisé applique un séparateur linéaire sur les images d'une instance par des transformations de $\mathcal{X}$ vers $\mathbb{R}$ non nécessairement linaires. Notons Φ_1, Φ_2,...,Φ_T ces transformations, pour $T \in \mathbb{N} \setminus \{0\}$. Mathématiquement, la fonction de décision d'un modèle linéaire généralisé s'écrit comme suit.

$$\exists \mathbf{w} = (w_1,...,w_T) \in \mathbb{R}^T \text{ et } b \in \mathbb{R} \text{ tel que } \forall \mathbf{x} \in \mathcal{X},\ g(\mathbf{x}) = b + \sum_{t=1}^{T} w_t.\Phi_t(\mathbf{x})$$

L'arbre de décision comme modèle linéaire généralisé

Soit un arbre de décision à N feuilles, défini par la fonction $q : \mathcal{X} \longrightarrow \{1,...,N\}$ associant à chaque instance l'indice de la partition (ou la feuille) qui la contient, et le vecteur de poids $\mathbf{w} = (w_1,...,w_n)$ où chaque composante w_i correspond au poids associé à la partition d'indice i. On construit également la famille de fonctions indicatrices $(\Phi_i)_{i\in\{1,...,N\}}$ définie par :

$$\begin{aligned}\forall \mathbf{x} \in \mathcal{X},\ \Phi_i(\mathbf{x}) &= \mathbb{1}[q(\mathbf{x}) = i] \\ &= 1 \text{ si } (q(\mathbf{x}) = i) \text{ et } 0 \text{ sinon}\end{aligned}$$

Φ_i est une fonction qui indique l'appartenance de l'instance à la partition d'indice i. On observe alors que la sortie de l'arbre de décision est un modèle linéaire généralisé sans biais :

$$\forall \mathbf{x} \in \mathcal{X},\ g(\mathbf{x}) = \sum_{i=1}^{N} w_i.\Phi_i(\mathbf{x})$$

Cette modélisation est équivalente à éclater l'arbre de décision en N branches. Chaque branche prédit 1 si l'instance satisfait toutes ses conditions et 0 sinon. Les sorties de toutes les branches formeront un vecteur à N composantes et qui sera multiplié par $\mathbf{w}$. La figure 1.2 montre cette

procédure pour un arbre de décision à 3 feuilles et dont les noeuds internes vérifient des conditions notées A et B.

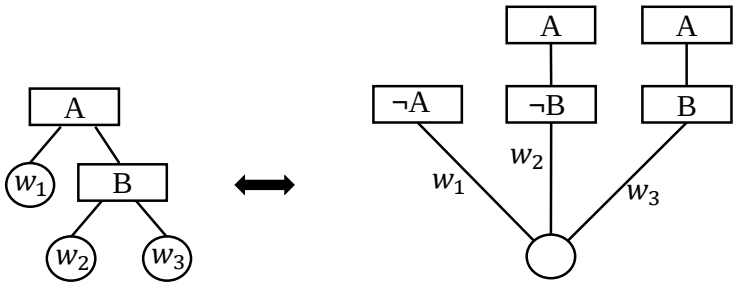


FIGURE 1.2 – Équivalence entre un arbre de décision et un modèle linéaire généralisé

En particulier, pour $N = 2$, l'arbre de décision peut être exprimé en fonction d'une de ses branches.

$$\forall \mathbf{x} \in \mathcal{X},\ g(\mathbf{x}) = w_1 + (w_2 - w_1).\Phi_2(\mathbf{x})$$

La figure 1.3 montre graphiquement cette équivalence.

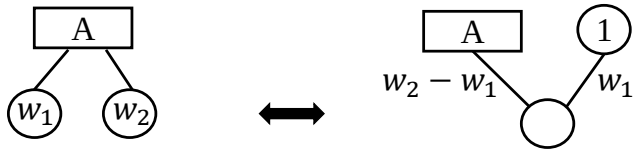


FIGURE 1.3 – Équivalence entre un arbre de décision à 2 branches et un modèle linéaire généralisé basée sur une branche

Un seul arbre de décision donne généralement de faibles performances vu qu'en majorité des cas les données ne sont pas séparables par des seuils de décision. Toutefois, plusieurs arbres peuvent bien performer dans une combinaison intelligente, appelée méthode d'ensemble.

Les méthodes d'ensemble apprennent un classificateur dont la sortie est une combinaison linéaire de sorties de plusieurs prédicteurs. Ces derniers peuvent être de même famille de fonctions ou bien de familles hétérogènes. L'objectif de l'algorithme d'apprentissage est de produire des prédicteurs assez complémentaires afin d'obtenir une combinaison qui performe mieux que tous les prédicteurs individuels qui la forment. On distingue deux stratégies de

construction d'une telle complémentarité, à savoir : le *bagging* (bootstrap aggregation) et le *boosting*.

Le bagging

Le bagging consiste à fournir à chaque prédicteur un sous échantillon de $\mathcal{S}$ obtenu par tirage aléatoire avec remise. Il y a donc une grande chance que certains prédicteurs reçoivent des échantillons différents les uns des autres. Ils produiront alors des modèles variés et qui ont une grande chance d'être complémentaires. La technique du bagging est souvent utilisée pour créer des ensembles d'arbres de décision, appelés forêts aléatoires.

Le Boosting

Le boosting construit itérativement les prédicteurs et leurs poids associés en garantissant que le nouveau prédicteur soit complémentaire à l'ancienne combinaison. D'abord, les évaluations de la fonction de perte associée à chaque exemple sont pondérées uniformément avec des poids uniformes $d_i = 1/n \ \forall i \in \{1, ..., n\}$. Ensuite, l'algorithme d'apprentissage ajuste les pondérations pour valoriser les exemples pour lesquels l'ensemble déjà créé performe mal. Le prochain prédicteur sera alors orienté vers la modélisation de ces exemples. Certains algorithmes modifient en plus les vraies étiquettes avec un terme équivalent à un résidu entre la prédiction de l'ensemble et la vraie étiquette, pour chaque observation.

Dans ce qui suit, la fonction de décision d'un ensemble à T classificateurs binaires $\{f_1, ..., f_T\}$ sera notée comme suit.

$$\forall \mathbf{x} \in \mathcal{X}, g^{(T)}(\mathbf{x}) = \sum_{j=1}^{T} \alpha_j \ . \ f^{(j)}(\mathbf{x})$$

Avec $(\alpha_1, ..., \alpha_T) \in \mathbb{R}^T$. Les classificateurs et l'ensemble sont également appelés votants et vote de majorité, respectivement. Il est possible d'inclure les poids dans la sortie des votants individuels. On aura alors une somme d'algorithmes de régression.

On distingue deux approches principales de Boosting, à savoir Adaboost (Schapire, 2003) et XGBoost (Chen and Guestrin, 2016). L'algorithme Adaboost ajoute à l'ensemble $g^{(t)}$ créé à l'itération t, une fonction f multipliée par son poids α pour minimiser l'erreur exponentielle pondérée suivante.

$$\begin{aligned} E_{exp}(\mathcal{S}, t+1) &= \sum_{i=1}^{n} d_i \ . \ exp[-y_i(g^{(t)}\big(\mathbf{x}_i) + \alpha.f(\mathbf{x}_i)\big)] \\ &= \sum_{i=1}^{n} d_i \ . \ exp[-y_i \ . \ g^{(t)}(\mathbf{x}_i)] \ . \ exp[-y_i \alpha f(\mathbf{x}_i)] \\ &\propto \sum_{i=1}^{n} d_i^{(t)} \ . \ exp[-y_i \alpha f(\mathbf{x}_i)] \end{aligned} \tag{1.2}$$

Avec

$$d_i^{(t+1)} = \frac{d_i \ . \ exp[-y_i.g^{(t)}(\mathbf{x}_i)]}{\sum_{i=1}^{n} d_i \ . \ exp[-y_i g^{(t)}(\mathbf{x}_i)]} \tag{1.3}$$

On peut démontrer que :

$$d_i^{(t+1)} = \frac{d_i^{(t)} \ . \ exp[-y_i.\alpha_t.f^{(t)}(\mathbf{x}_i)]}{Z_t} \tag{1.4}$$

avec $Z_t = \sum_{i=1}^{n} d_i^{(t)} \ . \ exp[-y_i.\alpha_t.f^{(t)}(\mathbf{x}_i)] = E_{exp}(\mathcal{S}, t)$ un terme de normalisation.

L'algorithme Adaboost en découle alors sous forme de trois étapes principales. En effet, à chaque itération t, l'algorithme :

— Appelle l'algorithme d'apprentissage du votant pour produire un classificateur f minimisant l'équation 1.2

— Trouve α la solution de 1.2 pour un f choisi

— Met à jour les poids des observations selon l'équation 1.4

L'algorithme détaillé est fourni en Annexe A.

Selon l'expression 1.3, l'algorithme Adaboost associe aux erreurs de prédiction ($y_i.g^{(t)}(\mathbf{x}_i) < 0$) un poids exponentiel, d'amplitude proportionnelle à $exp(|g^{(t)}(\mathbf{x})|)$. Pour les bonnes prédictions, il associe un faible poids proportionnel à $exp(-|g^{(t)}(\mathbf{x})|)$. En tenant compte du terme de normalisation, les poids associés aux bonnes classifications seront négligeables devant ceux associés aux poids des erreurs de classifications. La pondération par l'algorithme Adaboost est donc assimilable à la création d'un nouvel échantillon d'entraînement en éliminant les exemples bien classifiés.

De plus, l'expression de α et celle des nouveaux poids $d^{(t+1)}$ permettent au classificateur produit à l'itération t de devenir équivalent à un choix aléatoire au sens des pondérations de l'itération suivante. Ceci, avec un bon choix de $f^{(0)}$, rend Adaboost naturellement adapté aux données déséquilibrées. La propriété A.2.1 de l'annexe A détaille cet effet.

L'algorithme XGBoost, quant à lui, minimise toute fonction objective deux fois différentiable par rapport aux prédictions de l'ensemble, et construit un ensemble à partir de la somme de T algorithmes de régression $f^{(1)}, ..., f^{(T)}$.

$$\forall \mathbf{x} \in \mathcal{X}, g^{(T)}(\mathbf{x}) = \sum_{j=1}^{T} f^{(j)}(\mathbf{x}) \tag{1.5}$$

À chaque itération $t+1$, l'algorithme optimise un votant f qui minimise le développement de

Taylor d'ordre 2 de la fonction objective $E(\mathcal{S}, t+1)$ autour des prédictions $g_{i,t} = g^{(t)}(\mathbf{x}_i)$.

$$E(\mathcal{S}, t+1) = \sum_{i=1}^{n} \Big[\ell(y_i, g_{i,t}) - a_{i,t+1}.f(\mathbf{x}_i) + \frac{b_{i,t+1}}{2} f^2(\mathbf{x}_i) \Big]$$
$$\text{avec } a_{i,t+1} = -\frac{\partial \ell(y_i, g_{i,t})}{\partial g_{i,t}} \text{ et } b_{i,t+1} = \frac{\partial^2 \ell(y_i, g_{i,t})}{\partial g_{i,t}^2}$$

En complétant le carré, la fonction objective s'écrit comme suit.

$$E(\mathcal{S}, t+1) = E(\mathcal{S}, t) + \text{cste} + \sum_{i=1}^{n} \frac{b_{i,t+1}}{2} \Big[f(\mathbf{x}_i) - \frac{a_{i,t+1}}{b_{i,t+1}} \Big]^2$$

Ainsi, à chaque itération t, XGBoost calcule des nouveaux poids définis par $d_i^{(t)} = b_{i,t}/2$ et des nouvelles étiquettes $y'_i = a_{i,t}/b_{i,t}$ et appelle l'algorithme d'apprentissage pour produire un algorithme de régression f minimisant l'erreur quadratique moyenne entre les prédictions par f et les pseudo étiquettes $(y'_i)_i$ et de carrés pondérés par $(d_i^{(t)})_i$.

Par sa formulation indépendante du choix de la fonction de perte, XGBoost s'avère plus générique qu'Adaboost et peut être utilisé avec des fonctions objectives personnalisées. Les librairies logicielles l'implémentent avec la fonction de perte logistique. Cependant, elles offrent la possibilité d'injecter une fonction calculant $a_{i,t}$ et $b_{i,t}$. En particulier, lorsqu'on choisit la perte exponentielle, et lorsqu'on choisit f comme une fonction proportionnelle à un classificateur, l'algorithme XGBoost devient une approximation d'Adaboost. Les détails des liens entre Adaboost et XGBoost sont fournis en annexe B.

Dans le cas où f est un arbre de régression, XGBoost peut être reconstruit comme une fonction linéaire des sorties des feuilles des arbres de l'ensemble. De plus, en additionnant les vecteurs indiquant l'importance des attributs de chaque arbre individuel, on obtient un vecteur résultant indiquant leur importance au sens de XGBoost. Comme la construction des arbres de décision est incrémentale, il est possible que certains attributs ne soient jamais utilisés. Il est alors possible de les enlever et de réduire la dimension des instances. Le vecteur d'importance des attributs peut être utilisé dans une méthode supervisée de sélection d'attributs, visant à optimiser davantage la fonction objective. Ceci fera l'objet de la section 1.7.

1.4 Critères de performances en classification binaire

Durant la phase de test, le classificateur produit par un algorithme d'apprentissage peut être évalué d'une part en calculant l'erreur de classification, ou la fonction objective alternative sur l'ensemble de test. De l'autre part, il peut être évalué selon des métriques additionnelles basées sur l'analyse des erreurs de classifications. En particulier, les vraies et les fausses positives.

1.4.1 Matrice de confusion

Les prédictions du classificateur sont soit positives (+1) ou négatives (−1). Une prédiction est dite 'vraie' lorsqu'elle est égale à la vraie étiquette associée à l'instance. Selon son exactitude, une prédiction peut alors appartenir à l'une des catégories suivantes.

— **Vraies positives (VP)** : Ce sont les instances d'étiquettes positives, auxquelles le classificateur a associé des étiquettes positives

— **Vraies négatives (VN)** : Ce sont les instances d'étiquettes négatives, auxquelles le classificateur a associé des étiquettes négatives.

— **Fausses positives (FP)** : Ce sont les instances d'étiquettes négatives, auxquelles le classificateur a prédit des étiquettes positives

— **Fausses négatives(FN)** : Ce sont les instances d'étiquettes positives, pour lesquelles le classificateur a prédit des étiquettes négatives.

Ces quatre mesures donnent une description assez complète du comportement du classificateur et de la répartition de ses erreurs. Elles sont généralement représentées dans un tableau (voir tableau 1.1) appelé **matrice de confusion**.

Tableau 1.1 – Matrice de confusion

	Prédiction positive	**Prédiction Négative**
Vraie étiquette positive	VP	FN
Vraie étiquette négative	FP	VN

Idéalement, on souhaite avoir un classificateur ayant une matrice de confusion diagonale, sans fausses positives ni fausses négatives. Plus la matrice de confusion est proche d'une matrice diagonale, mieux est le classificateur.

1.4.2 Exactitude

Pour un ensemble $\mathcal{S}$ de n exemples, l'exactitude de la classification est le ratio de la somme des vraies prédictions sur le nombre total des prédictions.

$$\text{Exactitude}(\mathcal{S}) = \frac{VP + VN}{VP + FP + VN + FN} = \frac{VP + VN}{n} = 1 - E_{0-1}(\mathcal{S}, h)$$

Cette métrique est pertinente lorsque les exemples sont équilibrés, c'est-à-dire distribuée uniformément par classe. En effet, dans le cas idéal où les deux classes ont le même nombre d'exemples, la contribution de chaque classe dans l'exactitude est exactement 50%. Par exemple, dans le cas particulier du classificateur trivial constant $\mathbf{x} \in \mathcal{X}, h_c(\mathbf{x}) = -1$, l'exactitude sur un ensemble équilibré sera de 50%. Or, dans notre cas de données déséquilibrées, l'évaluation de l'exactitude serait biaisée. À titre d'exemple, en supposant que la classe positive représente seulement 1% des données, le même classificateur constant aurait une exactitude de 99% sans détecter aucune vraie positive.

Certains algorithmes d'apprentissage peuvent produire un tel classificateur trivial. En effet, dans le cas des petits disjoints et du chevauchement (voir les sections 2.2.2 et 2.2.1), l'amélioration des vraies positives risque de s'accompagner par une diminution importante du nombre de vraies négatives. Il s'ensuit que l'amélioration du nombre de vraies positives génère une grande pénalité dans la fonction objective. Pour remédier à ce problème, il est courant d'introduire des poids aux termes de la fonction objective pour augmenter la contribution des erreurs sur la classe +1 dans sa diminution. Ceci est équivalent à dupliquer des exemples de la classe minoritaire où à enlever des exemples redondants de la classe majoritaire.

Le problème constaté relativement à l'exactitude (et l'erreur de classification) est qu'elle peut être élevée avec $VP = 0$ dans le cas des données déséquilibrés. Nous avons donc besoin de métriques par classe, à savoir la précision, le rappel et la mesure F_β

1.4.3 Précision et Rappel

La précision et le rappel se calculent relativement à une seule classe de choix C qui sera référée par la classe positive. Donc, les prédictions seront dites positives si elles sont égales à C et négatives sinon. Ces métriques sont définies comme suit.

$$\text{Précision}(C) = \frac{VP}{VP + FP}$$
$$\text{Rappel}(C) = \frac{VP}{VP + FN}$$

La précision indique le pourcentage des instances d'étiquettes positives C relativement au nombre d'instances ayant la prédiction C. Un classificateur peut être très précis tout en détectant un très faible nombre d'instances de vraies étiquettes C. Le rappel s'impose alors comme métrique complémentaire quantifiant le pourcentage restant d'instances d'étiquette C à inclure dans les prédictions positives.

Selon la valeur d'affaire souhaitée du classificateur, il est courant de préférer une augmentation du rappel contre une faible baisse de la précision et vice versa. Cette préférence est gérée par la métrique F_β.

1.4.4 F_β-score

La mesure F_β-score, une métrique à valeur dans $[0, 1]$, permet de quantifier des comportements de classificateurs réalisant des compromis entre la précision et le rappel.

$$F_\beta\text{-score} = (1 + \beta^2) * \frac{\text{Précision} * \text{Rappel}}{(\beta^2 * \text{Précision}) + \text{Rappel}}$$

Pour $\beta = 1$, la précision et le rappel contribuent de manière égale à l'augmentation de F_β. Des valeurs plus faibles de β correspondent à préférer des classificateurs à haute précision. Inversement, des valeurs plus élevées de β favoriseront les classificateurs à haut rappel. Le cas

limite de $\beta = +\infty$ correspond à préférer des classificateurs à très haut rappel et à très faible précision. Inversement, $\beta = -\infty$ donne une métrique F_β qui associe un score important aux classificateurs à très haute précision et à très faible rappel.

1.5 Sélection de modèles et des hyperparamètres

Nous avons vu que les algorithmes d'apprentissage produisent des classificateurs en minimisant une fonction objective supérieure à l'erreur de classification. Or, nous avons vu que pour les données d'entraînement déséquilibrées, certains classificateurs simplistes décideront de mal performer sur la classe minoritaire. Cela arrive lorsque le classificateur appartient à une famille de fonctions qui n'est pas assez riche pour exprimer les relations entre les instances et les étiquettes. Toutefois, l'utilisation de familles de fonctions complexes et très riches risque de produire un classificateur qui prédit au cas par cas. Il s'agit d'une mémorisation totale de l'ensemble d'entraînement. Il y a donc absence d'apprentissage dans les deux cas. En effet, dans le premier cas, le classificateur confond les exemples difficiles de classe $+1$ à des exemples de classe -1. Dans le deuxième cas, le classificateur mémorise les observations au lieu d'en extraire des corrélations et des règles de décision. L'identification d'un bon choix de complexité s'impose.

Les familles de modèles diffèrent par leur complexité et leur force d'exprimer des relations entre des instances et des étiquettes. On parle alors de sélection de modèles lorsqu'on souhaite choisir une famille de modèles. De plus, au sein d'une même famille, la complexité des classificateurs dépend des choix des hyperparamètres. Ces derniers sont définis comme des paramètres non optimisés par la minimisation de la fonction objective. Ainsi, on parle de sélection d'hyperparamètres ou de paramétrage fin lorsqu'il s'agit de sélectionner des hyperparamètres.

Selon leur type, on distingue trois techniques de sélection de modèles et des hyperparamètres. À savoir :

— La régularisation de la fonction objective

— L'arrêt prématuré

— La recherche exhaustive

1.5.1 La régularisation de la fonction objective

Nous abordons la régularisation de la fonction objective dans le but de calibrer la complexité du modèle afin d'éviter le surapprentissage. Cette méthode consiste à modifier la fonction objective pour pénaliser les modèles les plus complexes et minimiser l'erreur d'apprentissage. Ceci est effectué en ajoutant un terme de régularisation à la fonction objective telle que :

$$\min_h \sum_{i=1}^{n} \ell(h(\mathbf{x}_i), y_i) + \lambda R(h)$$

où λ est le coefficient qui permet de contrôler l'importance relative du terme de la régularisation $R(h)$ et du terme de l'erreur. Un λ élevé risque de produire un classificateur simple qui minimise la régularisation sans forcement minimiser l'erreur de classification. Il faut choisir l'hyperparamètre λ afin d'atteindre un compromis de minimisation des deux termes de la fonction objective.

1.5.2 L'arrêt prématuré

L'arrêt prématuré est utilisé pour déterminer le nombre d'itération pour les modèles appris itérativement. Il consiste à arrêter l'entraînement au moment déclenchant le surapprentissage (1.4)

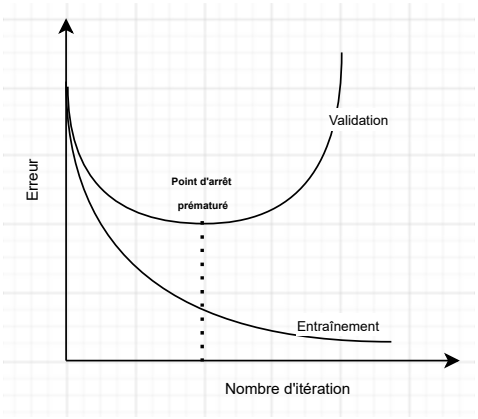


FIGURE 1.4 – Exemple synthétique d'arrêt prématuré

Parmi les conditions d'arrêt possibles on énumère :

— La fonction objective évaluée sur l'ensemble de validation est saturée et devenue asymptote (des faibles fluctuations autour d'une valeur constance). L'arrêt prématuré est effectué lorsque l'allure de l'erreur évolue par faibles fluctuations autour d'une valeur finale constante.

— La fonction objective sur l'ensemble de validation cesse de baisser pendant un certain nombre d'itérations. L'arrêt prématuré est effectué en prenant l'itération associée à la valeur minimale de la fonction objective sur l'ensemble de validation.

1.5.3 La recherche exhaustive de grille

On suppose pour chaque hyperparamètre qu'on a un ensemble de valeurs candidates. La recherche exhaustive consiste à tester toutes les combinaisons possibles des choix des hyperparamètres parmi lesdits ensembles. La meilleure combinaison est celle réalisant la meilleure performance selon une métrique évaluée sur l'ensemble de validation. Toutefois il existe des méthodes moins exhaustives tel que la recherche aléatoire et l'optimisation bayesienne. La

première consiste à exploser un ensemble de combinaison aléatoire de valeurs des hyperparamètres l'objectif étant d'échantillonner efficacement la grille des hyperparamètres dans l'espoir de sorte qu'elle contienne une combinaison proche de l'optimal. la deuxième consiste à améliorer itérativement le choix des futures hyperparamètres en fonction des résultats donnés par les hyperparamètres déjà explorés.

1.6 Ingénierie des attributs

Cela consiste à projeter des données dans un espace où elles deviennent facilement séparables (avec une famille de fonctions non complexes). Cela peut s'effectuer en entraînant un modèle sur la tâche de classification et en prenant l'espace de caractéristiques appris, ou bien en utilisant une tâche prétexte (compression des données, apprendre à trier, apprendre une métrique/distance) et en utilisant les données projetées dans cet espace dans un autre algorithme de classification.

Étant donné que les méthodes de Boosting sont les plus performantes pour la classification des données déséquilibrées, et que ces méthodes reposent sur l'idée d'utiliser un classificateur simple et qui s'apprend rapidement comme votant, nous allons considérer les méthodes simples d'ingénierie des attributs. En particulier, PCA et RCA. Ces méthodes seront examinées dans l'optique de générer des projections linéaires susceptibles de remplacer les directions *one-hot* présentes dans les seuils de décision. À rappeler que l'encodage *one-hot* consiste à transformer une variable catégorique à k niveaux en un vecteur à k composantes où chaque composante d'indice j est la réponse (1 si oui, 0 sinon) à la question si la valeur de la variable est égale au niveau j.

1.6.1 Analyse en composantes principales (PCA)

PCA apprend une famille de vecteurs orthogonaux et normés, appelés directions principales. La transformation par PCA est équivalente à une rotation ou une réflexion, selon le signe du déterminant de la matrice de projection. L'apprentissage des directions principales est formulé comme une minimisation de l'erreur de reconstruction pour des fins de réduction de dimensionnalité. La minimisation de l'erreur de reconstruction signifie que l'instance et sa reconstruction à partir de sa projection doivent être égales ou très similaires. L'égalité est obtenue lorsque l'espace appris est de même dimension que l'espace des instances. Lorsque l'espace appris est à faible dimension, les composantes principales générées capturent le maximum possible de l'information contenue dans les instances d'entraînement. Ceci dit, la première composante principale capture la direction moyenne des données. La seconde capture la direction moyenne des données projetées dans l'espace orthogonal au sous-espace déjà créé (droite pour la première composante). Et ainsi de suite jusqu'à ce que le nombre de composantes coïncide avec la dimension des données.

1.6.2 RCA

RCA supervisée est une modification de PCA qui ne garantit pas l'orthogonalité des composantes. La modification consiste à remplacer la matrice de corrélation par une somme de matrices de corrélation des instances de même classe. Chaque élément de la somme est calculé en utilisant le centre de la classe correspondante. La matrice de projection est calculée en utilisant la factorisation de Cholesky de l'inverse de la matrice de corrélation, contrairement à la méthode PCA où on calcule ses vecteurs propres.

1.7 Sélection supervisée des attributs

On sélectionne les attributs permettant de mieux minimiser la fonction objective. Certaines méthodes de réduction de dimensions se basent sur la minimisation d'une fonction extraite entièrement des attributs, ces méthodes sont qualifiées de non-supervisées. Les méthodes supervisées s'adressent à la réduction de la dimensions des attributs pour améliorer les performances d'un algorithme supervisé.

1.7.1 *Sequential Floating Forward Selection* (SFFS)

L'algorithme SFFS (Somol et al., 2010) permet d'identifier l'ensemble des attributs les plus importants pour la classification. L'algorithme démarre avec un ensemble vide d'attributs sélectionnés. Ensuite, à chaque itération, le meilleur attribut permettant d'améliorer la fonction objective est ajouté à l'ensemble. Si ce dernier contient plus que deux attributs, l'algorithme procède à l'élimination (si possible) d'un attribut sélectionné afin d'améliorer la fonction objective. Finalement l'algorithme s'arrête quand aucune amélioration n'est possible (par ajout et suppression) et l'ensemble final des attributs sélectionnés est retourné.

1.7.2 *Recursive Feature Elimination* (RFE)

Certains algorithmes, par leur conception, permettent d'associer un score de pertinence aux attributs utilisés. À titre d'exemple, les arbres de décision et les ensembles d'arbres de décision associent à chaque attribut sa contribution (en termes de gain) dans la minimisation de la fonction objective. Les modèles linéaires appris sur des données centrées réduites encodent une telle information dans les poids appris. En effet, plus la valeur absolue d'un poids est élevée, plus l'attribut est important.

Il est courant pour ces algorithmes d'utiliser des heuristiques permettant la sélection des meilleurs attributs. On peut penser par exemple à la sélection de l'ensemble minimal d'attribut ayant une importance cumulative supérieure à un certain seuil choisi. Cependant, une telle méthode risque d'impliquer une baisse des performances. En effet, il est possible que l'utilisation d'un attribut peu important (à faible gain) révèle des règles de décision utilisant

d'autres attributs et qui séparent davantage les données. En d'autre termes, il est possible que certains attributs à faible gain soient importants pour la classification et devront être retenus.

La méthode RFE (Guyon et al., 2004) adopte une approche itérative et conservatrice d'élimination des attributs. En effet, à chaque itération, seul l'attribut ayant l'importance la plus faible est éliminé. Ensuite, un nouveau prédicteur est appris avec les attributs restants, et pour lequel, on évalue de nouveau l'importance des attributs utilisés. Ceci est répété jusqu'à ce qu'un seul attribut est restant, indépendemment de la variation des performances. Finalement, l'itération correspondant à la valeur minimale de la fonction objective est retenue. L'ensemble pertinent des attributs est alors obtenu en reproduisant la séquence d'éliminations menant à cette itération.

Conclusion

Ce chapitre a porté sur les prérequis théoriques de la classification binaire à partir des données déséquilibrées. Le chapitre suivant consiste à présenter les propriétés ainsi que la revue des techniques de traitement des données déséquilibrées.

Chapitre 2

Données déséquilibrées

Introduction

La problématique des données déséquilibrées est présente dans plusieurs domaines d'applications de l'apprentissage automatique supervisé, incluant notamment la biologie, médecine, et finance (Yang and Wu, 2006). Cette problématique se caractérise par un déséquilibre des probabilités antérieures des classes constituant la variable de sortie (Chawla et al., 2004). Dans ce cas, la classe dominante sera appelée la classe majoritaire (classe 0) et la classe de l'évènement rare sera appelée la classe minoritaire (classe 1).

2.1 Impacts du déséquilibre sur les classificateurs

Soit deux familles de classificateurs, à savoir les modèles à base d'arbres de décisions construits selon un critère de division et les modèles paramétrés mis à jour par descente de gradient. Dans le premier cas, l'objectif est de créer des règles de décision capables d'augmenter la pureté au sein de chaque division. En présence du déséquilibre, il est fort probable que les divisions seront construites en fonction de la classe majoritaire et au détriment de la classe minoritaire comme on peut le voir sur la figure 2.1.

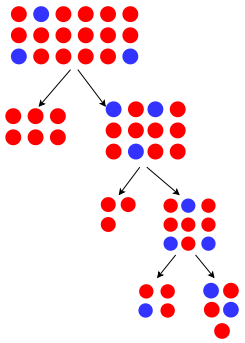


FIGURE 2.1 – Exemple d'arbre de décision à partir de données déséquilibrées

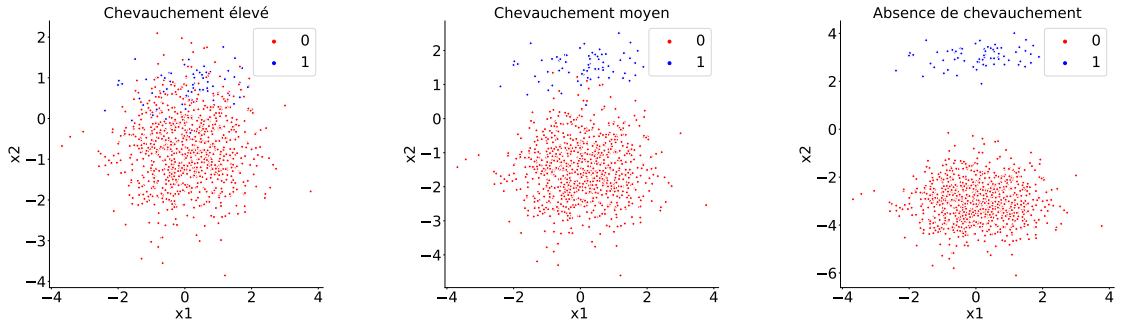
Par analogie, en utilisant un modèle paramétré la descente de gradient est mise à jour pour minimiser la fonction de perte. Cepndant, la mise à jour minimisant l'erreur sur la classe minoritaire est souvent associée à une augmentation importante de l'erreur sur la classe majoritaire. Il s'en suit qu'à la fin de l'apprentissage la contribution de la classe minoritaire dans les exemples est négligeable par rapport à celles des exemples de la classe majoritaire. Ceci revient à dire que les classificateurs sacrifient les performances sur la classe minoritaire pour bien classifier la classe majoritaire. Or même en présence du déséquilibre entre les classes si les données sont linéairement séparables, on peut construire un algorithme performant pour toutes les classes. Ceci revient à dire qu'il est primordial de comprendre les propriétés intrinsèques des données déséquilibrées afin de mieux cerner la nature du problème.

2.2 Propriétés générales des données déséquilibrées

La classification binaire avec l'apprentissage automatique à partir des données déséquilibrées introduit plusieurs défis de par la nature des données (Sun et al., 2009). Cette section porte sur les propriétés de ces données.

2.2.1 Chevauchement entre les classes

On parle d'un problème de chevauchement (Prati et al., 2004) lors de l'absence de séparabilité entre les classes. Ce problème apparaît lorsque dans un espace de représentation des variables il existe le même nombre d'instances par classe. Ceci se traduit par l'égalité des probabilités à priori dans ces espaces ce qui mène à une difficulté de distinction des classes. Afin d'illustrer cette propriété, on a généré un jeu de données [1] synthétique avec 1000 instances et deux classes ainsi qu'un déséquilibre fixé à 7% pour la classe 1 et 93% pour la classe 0 et on a varié le niveau de chevauchement.



(a) (b) (c)

FIGURE 2.2 – Illustration des données avec variation du degré de chevauchement

1. En utilisant Python et la librairie sklearn

La figure 2.2 montre que même avec un niveau de déséquilibre fixe, le degré de chevauchement impacte fortement la capacité de distinguer les deux classes. On remarque que les deux ensembles de données sur la figure 2.2a constituent deux partitions englobantes sans marges de séparation. Cependant, en réduisant le degré de chevauchement, les deux partitions commencent à s'éloigner, exemple 2.2b, jusqu'à l'obtention de deux partitions linéairement séparables, le voir sur la figure 2.2c.

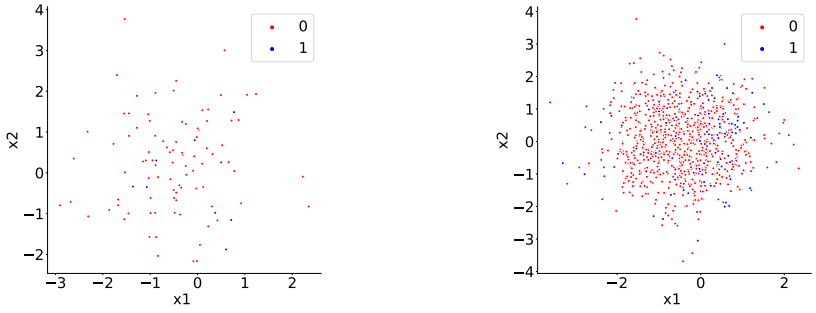
2.2.2 Petits disjoints

La difficulté de la classification à partir des données déséquilibrées peut également être causée par présence des petits disjoints (Jo and Japkowicz, 2004) dans les données. Ceci se produit quand les points de la même classe sont représentés sous forme de petits groupes disjoints, constituant ainsi des sous-concepts marginalisés. Bien que la majorité des données aient implicitement des groupes disjoints de la même classe, l'appareillement avec le problème de déséquilibre augmente la complexité de la classification. Il devient donc difficile de différencier entre le bruit et les sous-concepts.

2.2.3 Manque de données

On s'intéresse dans cette partie au nombre d'exemples de données disponibles pour l'entraînement des modèles. L'idéal est d'avoir un ensemble de données qui couvre tout l'espace de représentation des variables et leurs classes respectives. Le manque de données (Raudys and Jain, 1991) peut être considéré également comme un problème de manque d'information. La figure 2.3 montre le nuage de point à deux classes d'un jeu de données synthétique généré avec les propriétés suivantes :

— Nombre d'instances = 1000, Nombre de variables = 2, Ratio de déséquilibre = 7 : 100 , Degré de séparabilité = 0.5, Bruit = 0.15



(a) 10% d'instances d'entraînement

(b) 100% d'instances d'entraînement

FIGURE 2.3 – Exemple de manque de données d'entraînement

On représente 10% des points de données (2.3a) suivi de l'ensemble des données (2.3b) afin d'illustrer la caractéristique de manque de densité. On constate sur la figure 2.3a un manque de densité rendant ainsi difficile la création d'un modèle de généralisation permettant de créer des frontières de l'espace de représentation à partir des données. En outre, le déséquilibre des données accentue la difficulté de détecter la classe minoritaire, menant au surapprentissage du modèle dans cet espace. La classe minoritaire peut également être considérée comme un bruit dans ce cas de figure.

2.2.4 Bruit des étiquettes

Cette propriété désigne l'attribution erronée des étiquettes aux exemples, en changeant l'étiquette d'une instance de la classe minoritaire en classe majoritaire ou inversement. Ceci peut être causé par l'ambiguïté des instances aux frontières des classes ou par des erreurs de collection des données ce qui peut impacter aléatoirement la distribution des classes dans l'espace de représentation.

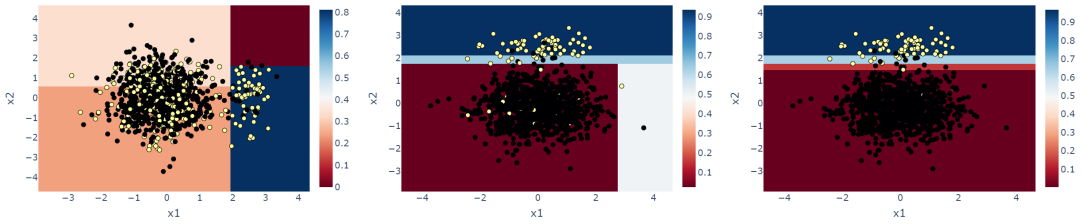
En présence du déséquilibre, le bruit des étiquettes (Weiss, 2004) est plus influent pour deux raisons :

— L'inversion des étiquettes d'instances de la classe minoritaire mène à une perte additionnelle d'information sur la représentation de cette classe.

— Le changement d'étiquette d'instances de la classe majoritaire vers la classe minoritaire risque de générer des sous-concepts accentuant la dispersion de la classe minoritaire.

Afin d'illustrer cette propriété, on a créé un ensemble de données avec les propriétés suivantes :

— Nombre d'instances = 1000, Nombre de variables = 2, Ratio de déséquilibre = 7 : 100 , Degré de séparabilité = 0.5, Bruit $\in \{1\%,\ 5\%,\ 50\%\}$



(a) Bruit = 0.5 (b) Bruit = 0.05 (c) Bruit = 0.01

FIGURE 2.4 – Surface de probabilité de la classe minoritaire (bleu) générée par un arbre de décision de profondeur = 2

On constate que les surfaces de probabilités qui déterminent les frontières créées par le modèle s'améliorent en réduisant le bruit. À cause de l'inversion des étiquettes on remarque sur la figure 2.4 l'apparition des petits disjoints de la classe minoritaire construite à partir d'instances bruitées, ainsi qu'un haut niveau de chevauchement entre les classes ce qui induit à l'incertitude

du modèle entraîné. Ce dernier s'améliore sur la construction des frontières de décision en réduisant le degré de bruit.

2.3 Revue des méthodes de traitement des données déséquilibrées

Les approches proposées dans la littérature pour traiter la classification à partir de données déséquilibrées (Barandela et al., 2003) se regroupent principalement en deux catégories. La première catégorie porte sur des approches de prétraitement de données afin de diminuer le déséquilibre entre les classes et la deuxième sur des méthodes de modification de la fonction objective qui se basent sur l'adaptation des classificateurs pour qu'ils prennent en considération le déséquilibre entre les classes.

2.3.1 Approches au niveau des données

Le ré-échantillonnage est une technique répandue dans le prétraitement des données déséquilibrées. Il consiste à équilibrer l'ensemble d'entraînement en réduisant l'asymétrie des distributions des classes (Batista et al., 2004). Il existe trois méthodes de ré-échantillonnage :

— Le sous-échantillonnage consiste à réduire les instances de la classe majoritaire

— Le sur-échantillonnage consiste à générer des instances de la classe minoritaire.

— L'échantillonnage hybride est une combinaison des deux méthodes précédentes.

Méthodes de sous-échantillonnage

Les méthodes de sous-échantillonnage consistent à sélectionner ou à générer des ensembles $\mathcal{S}'_0$ à partir de l'ensemble original d'entraînement de la classe majoritaire $\mathcal{S}_0$, tel que $|\mathcal{S}'_0| < |\mathcal{S}_0|$. Dans le cas de sélection d'ensembles, les instances sont choisies à partir des données initiales d'où $\mathcal{S}'_0 \subset \mathcal{S}_0$. Or lors de la génération d'ensemble, la méthode implémentée réduit la classe majoritaire et génère synthétiquement un ensemble d'entraînement majoritaire tel que $\mathcal{S}'_0 \not\subset \mathcal{S}_0$.

Exemples de méthodes de sélection

— Sous-échantillonnage aléatoire : un nombre prédéterminé d'instances de la classe majoritaire est sélectionné au hasard à partir de $\mathcal{S}_0$. Cette méthode non-heuristique est qualifiée de naïve puisqu'aucune hypothèse n'est faite sur les données. Le risque de perte d'information est élevé.

— Les liens de Tomek ou Tomek Links (Tomek, 1976) : représentent les instances dans l'ensemble de données dont le plus proche voisin appartient à la classe différente. Soit $\mathbf{x}_i$ et $\mathbf{x}_j$ deux instances appartenant à deux classes différentes avec une distance $d(\mathbf{x}_i, \mathbf{x}_j)$,

la paire $(\mathbf{x}_i, \mathbf{x}_j)$ est dite un lien de Tomek si :

$$\nexists\ k \in \{1, ..., n\} \setminus \{i, j\} \text{ tel que } d(\mathbf{x}_i, \mathbf{x}_k) < d(\mathbf{x}_i, \mathbf{x}_j) \text{ ou bien } d(\mathbf{x}_j, \mathbf{x}_k) < d(\mathbf{x}_i, \mathbf{x}_j)$$

Cette méthode de sous-échantillonnage consiste donc à éliminer les instances de la classe majoritaire qui forment des liens Tomek avec la classe minoritaire. Cette opération est effectuée jusqu'à l'obtention d'un ensemble de données $\mathcal{S}_0'$ tel que chaque paire de voisins les plus proches et les moins distants appartiennent à la même classe.

Exemples de méthodes de génération

— Centroïdes des partitions (Huajuan et al., 2020) : avec la supposition que l'ensemble de données est constitué de partitions, cette méthode consiste à remplacer les instances majoritaires par k centroïdes des partitions générées avec un algorithme de partitionnement. L'algorithme k-means fait parti des algorithmes les plus utilisés dans ce cas.

Méthodes de sur-échantillonnage

Le sur-échantillonnage consiste à générer des instances synthétiques de la classe minoritaire afin d'augmenter sa cardinalité. Il en résulte un ensemble $\mathcal{S}_1'$ contenant l'ensemble d'entraînement original de la classe minoritaire : $\mathcal{S}_1 \subset \mathcal{S}_1'$. Les méthodes les plus utilisées sont présentées ci-après :

Exemples de méthodes

— Sur-échantillonnage aléatoire consiste à dupliquer aléatoirement les instances de la classe minoritaire selon le niveau d'équilibre souhaité. Cette méthode peut causer ou augmenter le surapprentissage des classificateurs (He and Garcia, 2009)

— La méthode SMOTE (*Synthetic Minority Over-Sampling TEchnique*) introduite par Chawla et al. (2002), consiste à générer des instances synthétiques en interpolant plusieurs instances de la classe minoritaire en fonction des voisins les plus proches. Soit $\mathbf{x}_i \in \mathcal{S}_1$ tirée aléatoirement et ses k plus proches voisins $\{\mathbf{x}_{i1}, .., \mathbf{x}_{ik}\} \in \mathcal{S}_1$. La méthode SMOTE crée par interpolation aléatoire k nouvelles instances $\{r_{i1}, .., r_{ik}\}$. La formule suivante 2.1 est utilisée pour générer les nouvelles instances.

$$r_{ij} = x_i + \mathrm{U}(0,1) d(x_i - x_{ij}) \text{ avec } \ j \in \{1, ..., k\} \quad \text{et U la distribution uniforme dans } [0,1] \tag{2.1}$$

Cette méthode peut accentuer le chevauchement entre les classes vu qu'aucune condition sur la distribution des classes dans le sous-espace ré-échantillonné n'est prise en considération (Wang and Japkowicz, 2004).

— La méthode ADASYN (*ADAptive SYNthetic*) (He et al., 2008) repose sur la même formule d'interpolation pour dupliquer les instances de la classe minoritaire. Cependant,

ADASYN se différencie par l'utilisation d'une distribution pondérée différente pour chaque instance minoritaire à sur-échantillonner et définie à partir de la difficulté de classifier chaque instance de la classe minoritaire. La méthode utilise l'algorithme k-plus proches voisins pour déterminer la difficulté à classifier correctement une instance de la classe minoritaire. Plus la difficulté est élevée, plus son nombre d'instances synthétiques est élevé, plus sa chance d'être dupliquée est importante.

Méthode hybride

La combinaison des deux méthodes de ré-échantillonnage peut améliorer les performances des classificateurs. Le principe est d'utiliser conjointement les deux méthodes afin de mettre en relief la classe minoritaire et diminuer les chances d'avoir un classificateur simpliste biaisé vers la classe majoritaire (Seiffert et al., 2008).

2.3.2 Modification de la fonction objective

Cette approche consiste à incorporer explicitement la notion de la perte sensible aux attributs de l'instance dans la conception de l'algorithme. Il s'agit d'augmenter la pénalité de certaines ou toutes les instances de la classe minoritaire en fonction de leurs attributs. Deux stratégies principales existent.

La première consiste à choisir manuellement un ratio permettant de calculer les poids des instances de la classe minoritaire en fonction d'un poids choisi pour la classe majoritaire. Par exemple, on peut définir $d_{+1} = 5.d_{-1}$ signifie qu'une erreur de classification $+1$ coûte cinq fois l'erreur de classification d'une instance de la classe -1. Ce ratio peut être défini par un besoin d'affaire qui quantifie explicitement le coût d'une erreur de classification en métriques d'affaires (coût monétaire, temps de traitement de dossier, satisfaction de client, etc.). La deuxième approche consiste à optimiser ces poids en utilisant un algorithme d'apprentissage. La tache considérée peut être celle de la classification binaire ou une tache différente souvent dite tache prétexte.

Le premier cas correspond aux algorithmes de *Boosting* qui calculent itérativement des poids mettant en valeur les erreurs de classification. Ces dernières englobent des exemples de la classe minoritaire pour les premières itérations. Le calcul des poids par *Boosting* peut inclure des stratégies de ré-échantillonnage, comme c'est le cas par exemple pour SMOTE-Boost (Chawla et al., 2003) et EasyEnsemble (Liu et al., 2009).

Le deuxième cas consiste à résoudre une tache différente de la classification pour en extraire des bons choix de pondération des instances en espérant favoriser leurs classifications. Par exemple, Lichtenwalter and Chawla (2009) quantifie la disparité entre les distributions d'instances d'entraînement et celles de validation, et pondère les instances d'entraînement pour

la réduire. Comme résultat, les instances d'entraînement proches de l'ensemble de validation auront des poids plus importants

Conclusion

Ce chapitre est une présentation des données déséquilibrées. On a illustré leur influence sur la classification binaire supervisée, leurs propriétés ainsi qu'une revue des techniques utilisées dans la littérature. On a abordé le ré-échantillonnage afin de réduire l'asymétrie des distributions et la modification de la fonction objective afin de prendre en considération le déséquilibre par conception des classificateurs. Le chapitre suivant porte sur la méthodologie proposée.

Chapitre 3

Méthode proposée

Introduction

Par souci d'interprétabilité, les modèles simples et efficaces seront privilégiés face aux modèles plus complexes, et éventuellement plus performants, mais moins compréhensibles. Comme les modèles simples performent mieux comme votants dans des classificateurs de type vote de majorité, on s'intéressera alors aux méthodes d'ensembles. Ce chapitre présente la méthodologie suivie afin de prédire efficacement le départ des clients avec des règles de décisions simples. Une approche d'amélioration des résultats sera présentée et se concentrera plutôt sur l'adaptation des données aux algorithmes par ingénierie d'attributs.

3.1 Métrique à optimiser

Dans le sujet de rétention de clients en assurance, la classe d'intérêt correspond aux clients qui ont résilié. En effet, ne pas prédire le départ d'un client qui annulera sa police d'assurance (faux négatif) est plus pénalisant pour l'assureur que de faussement prédire le départ d'un client fidèle (faux positif).

Ne pas rater un vrai départ signifie la maximisation du rappel. Or, la maximisation du rappel seule pourra aboutir à un classificateur trivial prédisant naïvement la classe 'résiliation' pour toute donnée. Comme les résiliations représentent environ 7% des données, cela serait associé à un très faible pourcentage de prédictions correctes de résiliations. En d'autres termes, une très faible précision. Notre deuxième critère en découle alors, à savoir, maximiser en plus la précision de la classe 'résiliation'.

Avoir 100% de précision et 100% de rappel pour la classe d'intérêt est un cas idéal. Selon la section 1.4, un compromis doit être formulé. Pour la compagnie d'assurances, il est tolérable d'avoir une légère diminution du rappel si cela permet d'obtenir un classificateur plus précis. Cela donne un classificateur qui sert à automatiser une grande partie de la tâche de prédiction

du départ des clients, et qui pourra éventuellement être déployé en production.

Augmenter la précision en échange d'une faible diminution du rappel signifie en d'autres termes que :

— Lorsque le classificateur prédit 'résiliation', sa prédiction a de grandes chances d'être correcte ;

— Lorsque le classificateur prédit 'renouvellement', sa prédiction a des chances importantes, mais plus faibles, d'être correcte ;

Dans ce cas, le classificateur est crédible pour la prédiction des résiliations. Les résiliations non-détectées par le classificateur seront étudiées et éventuellement détectées par les anciens processus d'affaires mis en place par l'assureur, en attendant la création d'un classificateur plus performant.

Si le rappel du classificateur est non satisfaisant, on propose de créer un classificateur de mêmes propriétés, mais pour la classe 'renouvellement'. C'est un classificateur crédible et précis dans la prédiction des renouvellements. La combinaison intelligente des prédictions des deux classificateurs pourra aboutir à un classificateur final maximisant la précision et le rappel pour les deux classes.

Nous allons donc optimiser la mesure F_1. S'il est nécessaire de réaliser un compromis entre la précision et le rappel, on choisira de créer deux classificateurs complémentaires maximisant respectivement $F_\beta(+1)$ et $F_\beta(-1)$ pour différentes valeurs de β légèrement supérieures à 1 (c.f. section 1.4).

3.2 Énoncé de la méthodologie

La tâche de prédiction de la rétention est abordée comme un problème de classification. L'approche générique suivie consiste à créer des espaces de représentation (ou de coordonnées) dans lesquels les classes deviennent linéairement séparables. Cela est naturellement implémenté dans la majorité des algorithmes de classification performants pour le traitement des données déséquilibrées (c.f. paragraphe 1.3.4).

Les algorithmes simples et compréhensibles par un agent humain seront favorisés, à savoir les seuils de décision et les séparateurs linéaires clairsemés ($\mathbf{w}$ contenant un faible nombre de coefficients non nuls). On sait déjà que ces prédicteurs fonctionnent mieux dans un vote de majorité. Cependant, un tel classificateur efficace risque d'être non-interprétable. Nous allons donc simplifier le modèle en le transformant en une combinaison de branches d'arbres de décision, ou fonctions indicatrices, et un modèle linéaire. Ensuite, en utilisant une méthode supervisée de sélection des attributs, nous allons identifier les fonctions les plus utiles sans dégrader les performances conjointement à l'apprentissage d'un nouveau modèle linéaire qui remplace les sorties des feuilles des arbres déjà appris. Étant donné que la sortie des branches

est binaire dans $\{0, 1\}$, le seul seuil de décision possible est 0.5 [1]. Ainsi, un arbre de décision de profondeur 1 prenant la sortie d'une fonction indicatrice modélise la sortie en un vecteur à deux composantes. La première est identique à la sortie, la deuxième est sa négation. En considérant seulement la branche identique à la sortie, et sachant qu'un arbre de profondeur 1 peut être modélisé par un modèle linéaire biaisé (c.f. paragraphe 1.3.4) d'une de ses branches, on conclut qu'un arbre de décision de profondeur 1 est un modèle linéaire de son entrée. Nous pouvons construire une nouvelle combinaison des branches en utilisant un ensemble d'arbres de décision de profondeur 1. Les méthodes d'ensembles, particulièrement celles du *boosting* permettent de sélectionner itérativement des fonctions indicatrices. Elles ont donc l'avantage d'utiliser un sous-ensemble pertinent de fonctions, contrairement aux méthodes de création de séparateurs linéaires basées sur la descente du gradient, où on considère que toutes les entrées sont utiles. Une fois l'ensemble d'arbres de profondeur 1 construit, nous allons sélectionner davantage les fonctions les plus pertinentes en utilisant une méthode supervisée de sélection des attributs. La figure 3.1 montre la prédiction avec le modèle et avec les transformations finales apprises.

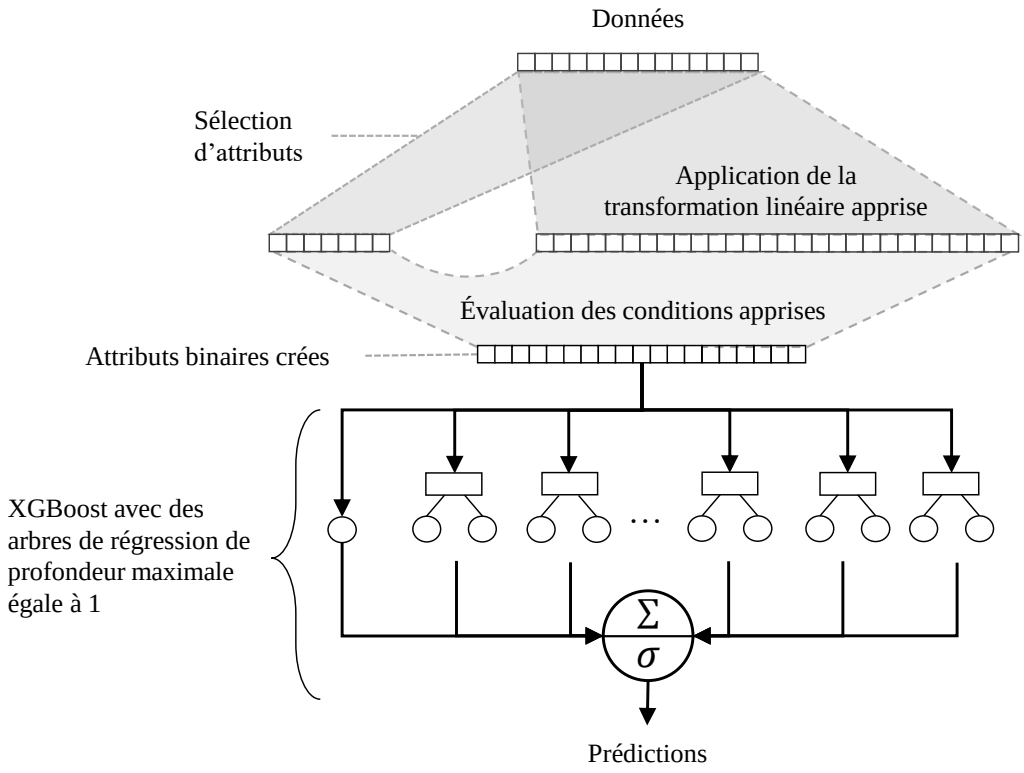


FIGURE 3.1 – Prédiction avec le modèle final appris

En première étape, une partie des colonnes en entrée sera retenue sans transformation, grâce à

1. Par construction de la fonction de perte, elle suppose que le seuil de séparabilité est 0.5 et optimise le modèle pour être efficace avec ce seuil

une sélection supervisée des attributs. Ensuite, une transformation linéaire sera appliquée aux entrées pour générer des projections. L'apprentissage de ces projecteurs sera effectué en deux temps. Le premier consiste à générer des directions de projections candidates, et le second consiste à filtrer les plus utiles pour les arbres de décision d'une méthode d'ensemble. Une fois l'ensemble d'arbres de décisions créé, il sera éclaté en branches. Dans la figure 3.1, ces branches sont appelées conditions apprises, et leurs sorties forment un vecteur d'attributs binaires. Ce dernier est alimenté à un modèle linéaire créé par XGBoost avec arbres de décision de profondeur 1. La prédiction finale est définie par la sigmoïde du score résultant. Elle quantifie la probabilité d'appartenance à la classe 1 au sens du classificateur.

Dans ce qui suit, nous allons justifier le choix de XGboost comme méthode de création de modèle linéaire généralisé, illustrer comment générer les projections candidates et finalement proposer une méthode rapide de leur création et qui s'intègre explicitement dans les itérations de XGBoost.

3.3 Choix de la famille de classificateurs

L'exemple le plus répandu des méthodes basées sur la représentation des données dans un espace où elles deviennent linéairement séparables est celui des méthodes à vecteurs de support. Elles sont mathématiquement formulées comme des séparateurs linéaires opérant sur un espace de caractéristiques défini par un noyau. Les réseaux de neurones vérifient également cette propriété. En effet, l'avant-dernière couche fournit des représentations apprises de sorte qu'un séparateur linéaire (dernière couche entièrement connectée) permet de classifier efficacement les instances.

Un arbre de décision ou un ensemble d'arbres de décision peuvent être vus comme un modèle linéaire généralisé (c.f. section 1.3.4). Contrairement aux algorithmes de *boosting*, les séparateurs linéaires, méthodes à noyaux et réseaux de neurones supposent que tous les attributs de l'instance sont utiles. Le modèle généré sera donc non-clairsemé, ce qui nuit à l'interprétabilité. De plus, d'une part, l'apprentissage des réseaux de neurones s'effectue par descente du gradient dans l'espace des poids, ce qui impose une pondération préalable de la classe minoritaire. D'autre part, leur apprentissage suppose un choix préalable de l'architecture. En somme, nous aurons un grand nombre d'hyperparamètres incluant la spécification de l'architecture et le ratio des pondérations des deux classes. Cependant, les approches de *boosting* construisent le modèle par descente du gradient dans l'espace des fonctions. C'est l'équivalent de modifier l'architecture (définie par le nombre de votants) à chaque itération dans la direction du gradient négatif de la fonction objective par rapport aux anciennes prédictions. Chaque fonction est typiquement un arbre de décision, construit de manière incrémentale en choisissant un attribut pertinent à chaque noeud de décision. De plus, chaque noeud apprend un seuil de décision par recherche exhaustive. Il est donc insensible à la distribution des étiquettes condi-

tionnellement aux instances. Finalement, dans le cas particulier des données utilisées pour ce projet, nous avons remarqué un grand nombre d'attributs nominaux. La représentation *one-hot* de ces derniers est bien adaptée aux arbres de décision.

Ainsi, nous avons opté pour l'utilisation des méthodes de *boosting*, en particulier XGBoost grâce à sa capacité de générer les autres algorithmes de *boosting* par injection des dérivées dans la fonction objective, et grâce à son implémentation incluant une pénalité sur la complexité des votants dans la même fonction. XGBoost, de manière similaire à Adaboost, implémente un mécanisme de pondération des exemples afin d'atténuer l'importance des bonnes prédictions et de valoriser les erreurs de classification dans la fonction objective à une itération donnée, et qui sera considérée par le prochain votant. Cela nous invite à penser qu'il est adapté aux données déséquilibrées.

Les arbres de décision permettent d'apprendre des séquences de seuils de décision sur des attributs. Cela permet entre autres d'identifier des interactions entre les attributs et des instances types de chaque classe. Pour les attributs scalaires (continus), les arbres de décision pourront être complètement incompréhensibles, pourront fournir une longue séquence de seuils, et pourront comparer les mêmes attributs de manière redondante.

On choisit donc d'appliquer des transformations linéaires (projections) sur ces attributs en espérant que des seuils de décisions sur les nouvelles coordonnées séparent efficacement les classes. En somme, il s'agit d'apprendre des arbres de décision sur des attributs scalaires transformés linéairement, et des attributs binaires.

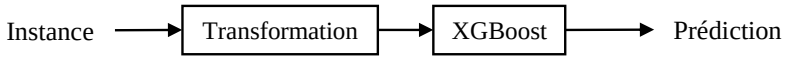


FIGURE 3.2 – Approche générale de classification

L'approche de classification consiste donc à transformer l'instance en requête, et lui appliquer XGBoost afin d'obtenir sa classe prédite (figure 3.2). Les transformations visent à adapter les données à XGBoost et réaliser des gains en termes de :

1. Performances, vitesse de convergence, et stabilité ;
2. Réduction de la complexité sans compromettre sur les performances ;

3.4 Transformation des données

3.4.1 Transformation identité

La transformation identité est équivalente à ne pas transformer les données. Les attributs bruts non-transformés seront donc utilisés pour apprendre des arbres de régression par XGBoost.

Ces arbres apprennent un partitionnent de l'espace des instances en des régions disjointes définis par la famille d'équations $\Delta_{j,\theta} : x[j] = \theta$ pour tout $j \in \{1, ..., d\}$ et pour θ un seuil quelconque appris. Les équations des frontières peuvent être également exprimées comme un produit scalaire.

$$\Delta_{j,\theta} : \mathbf{e}_j^T \cdot \mathbf{x} = \theta,\ \mathbf{e}_j \in \mathbb{R}^d \text{ tel que } e_j[j] = 1 \text{ et } e_j[k] = 0\ \forall k \in \{1, ..., d\} \setminus \{j\}$$

Ainsi, $\Delta_{j,\theta}$ définit un hyperplan orthogonal à la droite paramétrique $E_j = \{t.\mathbf{e}_j\ , t \in \mathbb{R}\}$ et l'intercepte au niveau $t = \theta$.

La figure 3.3 montre les frontières de décisions pour des données synthétiques à deux dimensions.

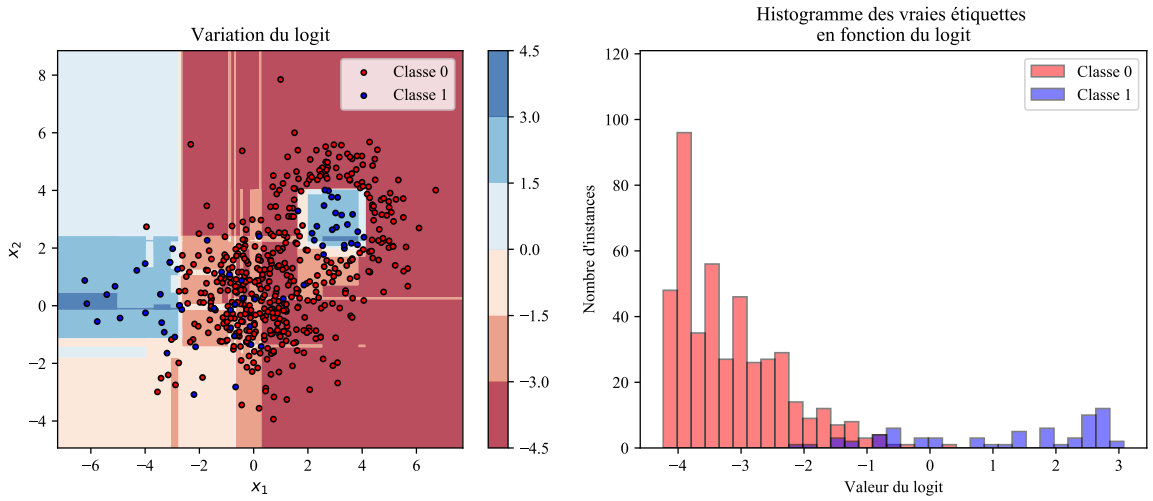


FIGURE 3.3 – XGBoost avec des données brutes

Le modèle définit des régions de $\mathbb{R}^2$ délimitées par des droites (hyperplans de $\mathbb{R}^2$) orthogonales aux directions canoniques $(0, 1)^T$ et $(1, 0)^T$. Le modèle définit également une fonction constante par région.

D'après la figure 3.3, on voit que ces frontières sont assez complexes (effet zigzag autour des exemples) sans pour autant fournir une séparation totale, car certains intervalles du logit ($[-2.1, -1.9]$ et $[-1.5, -0.75]$) contiennent des exemples de différentes classes.

L'identification des directions qui permettent de mieux séparer les étiquettes par des seuils de décision aura comme objectif de réduire le nombre total de seuils nécessaires pour classifier efficacement les instances.

3.4.2 Apprentissage des directions par PCA et RCA

Comme on ignore la distribution des étiquettes selon les directions principales, nous avons projeté les données en gardant toutes les composantes principales. XGBoost optimisera le choix des composantes pertinentes pour la classification. La figure 3.4 montre les frontières de décision sur notre jeu de données synthétique.

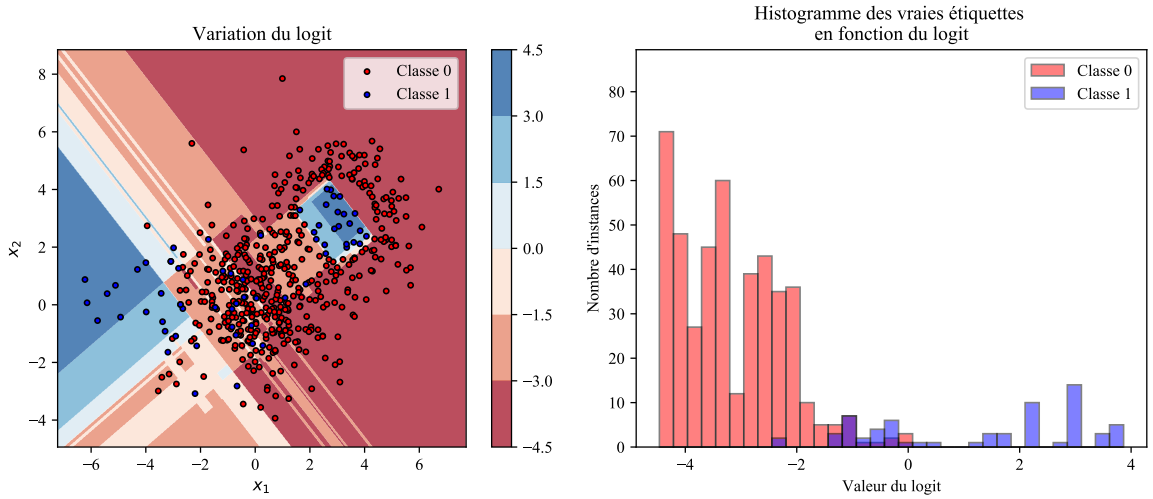


FIGURE 3.4 – Effet de la transformation PCA sur les frontières apprises par XGBoost

Notons que comparés aux frontières de la figure 3.3, les nouvelles frontières sont moins complexes. On voit des droites bien définies et qui délimitent la région où les instances de la classe 1 décrivent des groupes denses et quelques fluctuations dans la région $\{(x_1, x_2), x_1 \in [-4, -2], x_2 \in [-4, -1]\}$ sans effet sur le résultat de la classification. Cependant, avec ces frontières, XGBoost échoue également dans la classification des instances de classe 1 clairsemées dans un groupe dense d'instances de classe 0. Ceci se manifeste également par des intervalles de logit contenant un mélange d'instances des deux classes.

Notons que l'orthogonalité des directions n'est pas nécessaire pour la classification. Il suffit d'avoir des directions non liées (qui ne s'expriment pas comme combinaison linéaire des directions restantes) pour pouvoir générer des frontières pour la classification.

La figure 3.5 montre les frontières de décision générées par XGBoost, en utilisant des données transformées par RCA.

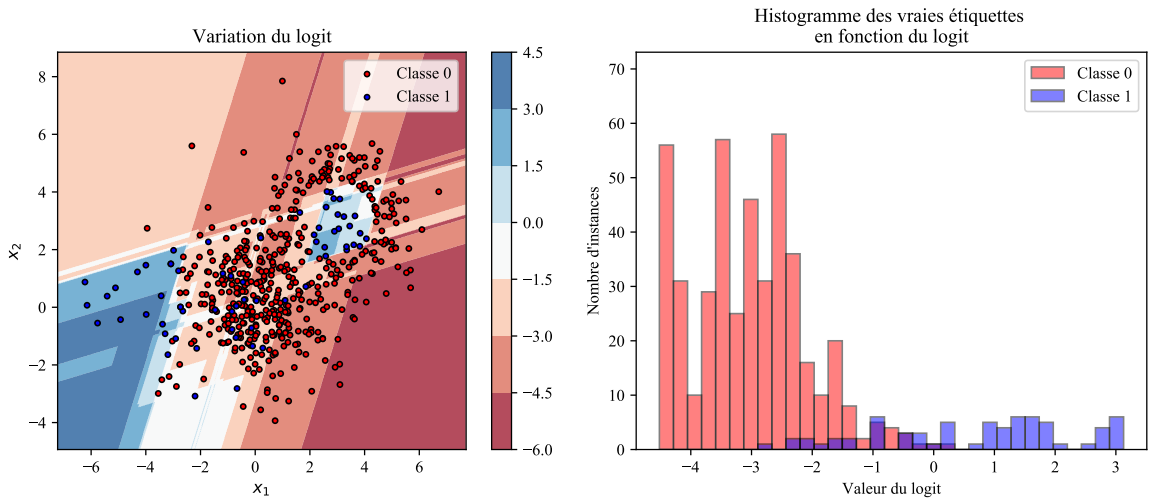


FIGURE 3.5 – Effet de la projection RCA supervisée sur les frontières apprises par XGBoost

Notons que ces frontières sont plus lisses que les frontières obtenues avec PCA. Les clusters denses de classe 1 sont mieux délimités. Les exemples instances de classe 1 éparpillés dans des groupes d'instances de classe 0 sont toujours mal classifiés.

3.4.3 Apprentissage adaptatif de projections

Au lieu d'apprendre des projections communes à tous les arbres de décision dans l'ensemble appris par XGBoost, on propose d'apprendre des projections adaptées aux données reçues par chaque arbre. L'idée est de légèrement modifier la méthode PCA afin de tenir compte des pondérations des exemples. On l'appellera alors PCA pondérée. Rappelons que XGBoost génère à chaque itération t de nouveaux poids $\{d_i^{(t)}, \forall i\}$ et de nouvelles étiquettes $\{y'_i, \forall i\}$ qui valorisent les exemples d'entraînement mal classifiés et produisent un votant capable de baisser la valeur de la fonction objective. Ainsi, une PCA pondérée qui donne plus d'importance à ces mêmes exemples capturera leurs tendances et donnera éventuellement des projecteurs adaptés à la classification par un arbre de décision. Dans notre implémentation, nous avons utilisé les mêmes pondérations fournies par XGBoost, correspondants à la dérivée seconde de la fonction de perte.

Le diagramme de la figure 3.6 montre l'apprentissage adaptatif des directions.

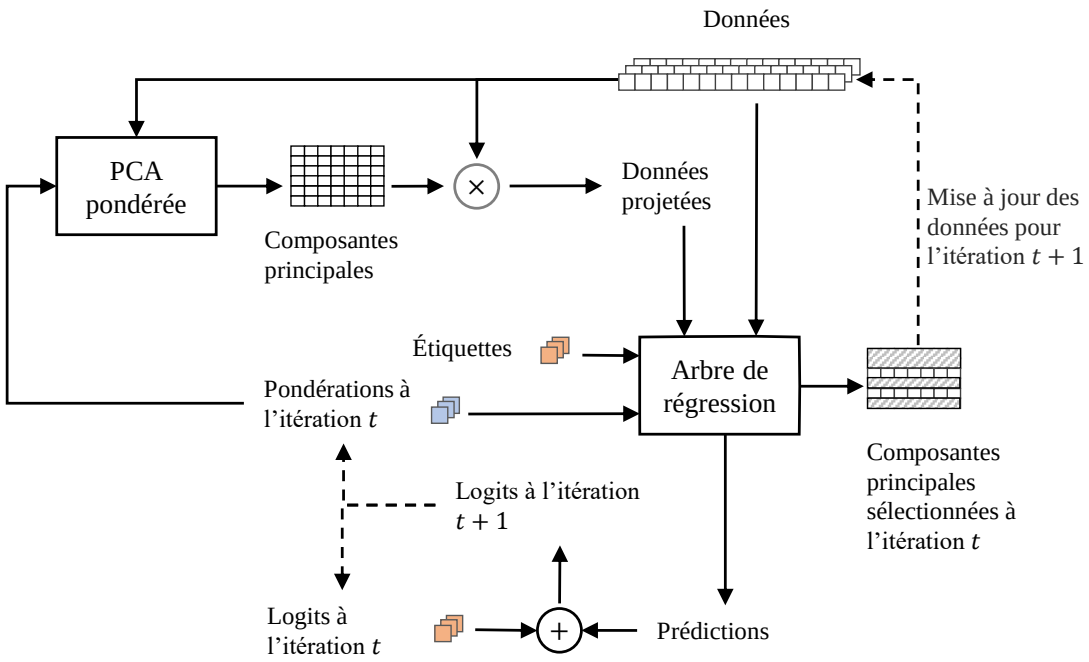


FIGURE 3.6 – Apprentissage des projections conjointement au *Boosting*

L'algorithme démarre avec des prédictions initialisées à -1, un choix du logit correspondant à la classe majoritaire. Cette itération est appelée itération 0. Les pondérations à la première itération et les nouvelles étiquettes sont alors calculées avec XGBoost. Nous avons choisi de minimiser l'erreur logarithme (ou logistique, ou entropie croisée binaire). Ensuite, on apprend des directions principales avec PCA pondérée en tenant compte des pondérations calculées. Les données sont ensuite projetées dans l'espace généré par les directions principales et ajoutées comme colonnes additionnelles aux données initiales. Ceci laissera l'apprentissage de l'arbre de régression décider d'utiliser les données initiales si elles sont plus pertinentes que les prédictions. Suite à l'apprentissage de l'arbre, on extrait la liste des attributs choisis. Celle-ci peut éventuellement inclure des projections. Dans ce cas, les directions principales correspondantes seront retenues comme directions pertinentes pour la classification et les projections correspondantes seront ajoutées aux données afin de pouvoir les réutiliser dans les prochaines itérations.

En résumé, l'arbre de décision à l'itération t apprend à partir des données initiales, des données transformées par PCA pondérée à la même itération, et des données transformées par des directions principales retenues depuis les itérations précédentes.

Comme dans XGBoost, les prédictions fournies par l'arbre appris seront ajoutées aux logits des exemples d'entraînement. Les nouvelles valeurs des Logits servent pour calculer les nouvelles pondérations et les nouvelles étiquettes pour l'itération suivante. L'algorithme s'arrête quand on atteint un maximum de nombre d'itérations, ou quand il cesse de choisir de nouvelles

directions pendant un certain nombre d'itérations.

La transformation linéaire apprise est modélisée par une matrice M par bloc avec un premier bloc identité suivie par les k directions apprises, stockées comme colonnes additionnelles.

$$M = \left(I_{m \times m} \;\middle|\; \mathbf{pc}_1 \;, \mathbf{pc}_2, \; ..., \; \mathbf{pc}_k \right)$$

Ensuite, l'approche de la figure 3.2 est appliquée en utilisant l'ensemble des directions apprises comme projecteurs. En d'autres termes, la matrice X de taille $n \times m$ où chaque instance occupe une ligne est multipliée par la matrice M apprise.

$$X' = X.M$$

Le bloc identité de M sert à traiter des colonnes de X sans subir des transformations. Ensuite XGBoost est appris de nouveau en utilisant les données X' et les étiquettes.

La figure 3.7 montre les frontières générées par XGBoost avec PCA pondérée. Notons que les données sont maintenant séparables. La variation du logit en fonction de x_1 et x_2 montre que cette approche permet de générer des frontières non orthogonales, assez complexes, et dont le nombre est supérieur à la dimension de l'entrée. De plus, les frontières ont une allure similaire à celles obtenues par des méthodes à noyaux.

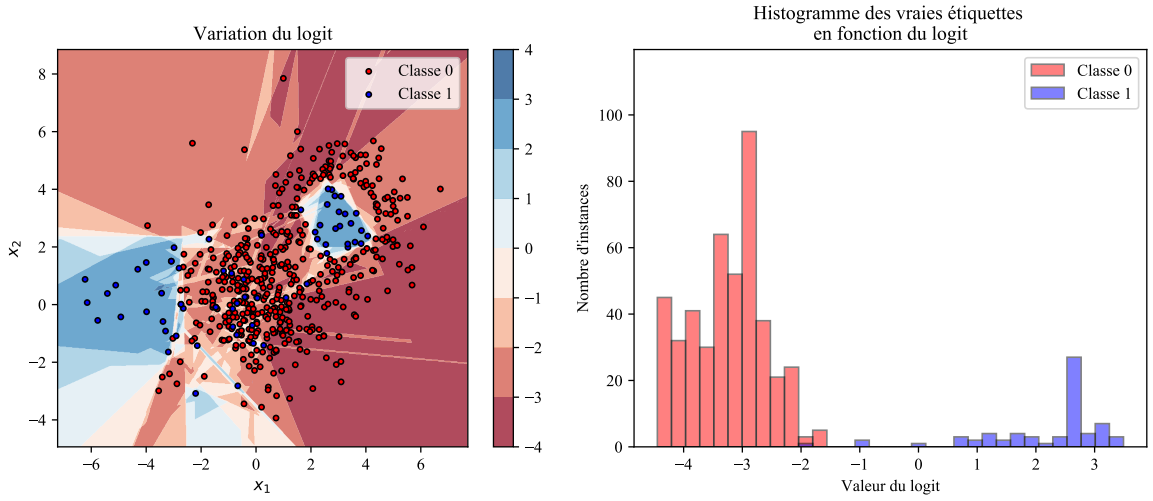


FIGURE 3.7 – Effet des projections adaptatives sur la frontière de XGBoost

3.5 Entraînement du modèle avec sélection supervisée des attributs

L'approche classique d'entraînement des modèles est adoptée. Les données sont divisées en trois ensembles ;

— Ensemble d'entraînement, (60% des données)

— Ensemble de validation, (20% des données)

— Ensemble de test.(20% des données)

L'ensemble d'entraînement sert à apprendre un modèle avec un choix spécifique d'attributs et des hyperparamètres. L'ensemble de validation sert à choisir les meilleurs hyperparamètres et les meilleurs attributs permettant de maximiser la mesure F1. Pour les hyperparamètres, nous avons choisi de varier le taux d'apprentissage et la profondeur maximale des arbres de décisions formant l'ensemble appris par XGBoost. Le nombre optimal d'arbres de décision est déterminé par arrêt prématuré. La sélection des colonnes est effectuée par la méthode RFE qui consiste à éliminer itérativement les colonnes d'importance nulle, et la colonne ayant un minimum d'importance jusqu'à éliminer toutes les colonnes. Ensuite, l'ensemble de colonnes donnant un maximum de mesure F1 sur l'ensemble de validation est retenu pour chaque choix d'hyperparamètres.

Ainsi, pour chaque choix des hyperparamètres, on déduit par RFE un ensemble de colonnes pertinentes, et par arrêt prématuré le nombre optimal d'arbres. La mesure résultante de F1 sur l'ensemble de validation est alors reportée et sera le critère de choix des meilleurs hyperparamètres.

3.6 Simplification du modèle

La transformation la plus adaptée à XGBoost sera retenue. Ensuite, chaque arbre de décision est décomposé en un ensemble de branches, qu'on appelle conditions ou fonctions indicatrices. Ces dernières sont évaluées sur les données précédemment transformées par les projections apprises pour générer des vecteurs binaires. Ensuite, un nouveau modèle XGBoost avec des arbres de profondeur maximale égale à 1 est appris sur ces données.

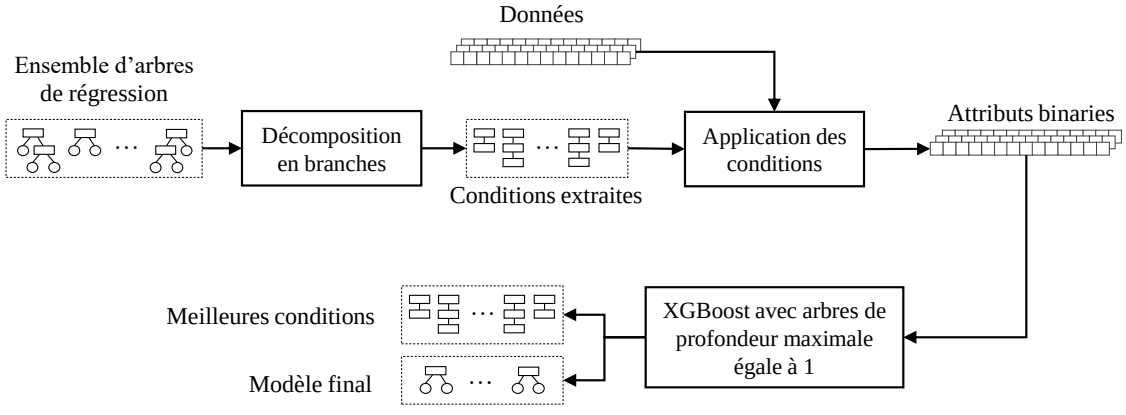


FIGURE 3.8 – Approche de simplification d'ensembles d'arbres de décision

La sélection des colonnes pour ce dernier modèle permet d'extraire un ensemble (de taille souhaitée faible) de conditions pertinentes. L'examen de ces conditions permet de dégager les projecteurs et les attributs initiaux les plus pertinents.

Conclusion

Dans cette partie, nous avons démontré le choix de XGBoost comme algorithme principal de classification. Sa capacité d'exprimer les relations entre les instances et leurs étiquettes en générant des directions de projections susceptibles d'être sélectionnées par les arbres de régression formant XGBoost a été enrichi. Nous avons également introduit une technique permettant de rendre PCA sensible aux pondérations générées par chaque itération de XGBoost, et ce afin de fournir des projections plus pertinentes pour atténuer la fonction objective. Finalement, comme la performance est impactée par une augmentation de la complexité de l'algorithme, nous avons proposé une méthode de restructuration de l'ensemble appris sous forme d'une combinaison linéaire d'un ensemble compact et efficace de fonctions indicatrices, construites par sélection des branches utiles de chaque arbre de l'ensemble.

Chapitre 4

Résultats

Introduction

Ce chapitre présente les données d'entraînement et les performances des approches listées dans la méthodologie. Dans un premier temps, on compare les méthodes de création de projections candidates. Ensuite, la meilleure méthode est choisie et le classificateur XGBoost correspondant sera simplifié en un modèle linéaire généralisé comme dans la section 1.3.4 opérant sur un ensemble minimal de fonctions indicatrices. Les performances du nouveau classificateur seront étudiées afin d'établir si cette simplification s'accompagne par une baisse des métriques surveillées. Finalement, on élimine la dernière couche linéaire du prédicteur, et on observe dans quelle mesure l'espace de représentation sépare les données.

4.1 Données

L'ensemble de données utilisé est fourni par l'entreprise LCASF. Il contient plus de 300 000 dossiers clients ayant le produit d'assurance vie temporaire avec 40 variables. Les données sont réparties exclusivement sur deux classes dont 93% de renouvellement (classe 0) et 7% de résiliation (classe 1). Le tableau 4.1 présente la liste de variables avec leur signification et leur type. On a donc 19 variables scalaires et 21 nominales anonymisées par l'entreprise. Lorsque l'attribut est scalaire, son amplitude varie entre 0 et 1. Si elle est nominale à $k \in \mathbb{N} \setminus \{0\}$ niveaux, ses valeurs sont dans l'ensemble $\{0, ..., k-1\}$. Puisqu'on considère l'apprentissage d'arbres de décision par XGBoost, on a considéré l'encodage *one-hot* comme transformation éventuellement pertinente pour faciliter la sélection des niveaux des attributs nominaux avec des arbres moins complexes. Par exemple, la sélection du deuxième niveau d'un attribut nominal à trois niveaux nécessite deux seuils lorsque l'attribut est non-encodé. Par contre, elle nécessite un seul seuil appliqué à la colonne correspondant au niveau si l'attribut est encodé *one-hot* . À ce stade, on ignore si cette transformation est réellement adaptée au jeu de données. On considère les deux variantes, soit avec et sans encodage, et on choisira la transformation

associée aux meilleurs résultats.

Tableau 4.1 – Sommaire des attributs

ID	Variable	Signification	Type de donnée
Var0	lineofbusiness	Secteur d'activité	Nominal
Var1	classbase	Type de couverture d'assurance	Nominal
Var2	inmode	Mode de paiement de la prime	Nominal
Var3	age-em	Age de l'assuré	Scalaire
Var4	sexe	Genre de l'assuré	Nominal
Var5	infum	Usage du tabac	Nominal
Var6	issst	Statut de vente	Nominal
Var7	rsdst	Statut de résidence	Nominal
Var8	policystatus	Statut de la police d'assurance	Nominal
Var9	numberofunits	Nombre d'unités	Scalaire
Var10	valueperunit	Valeur par unité	Scalaire
Var11	elimpdcode	Code d'élimination	Nominal
Var12	benefitcode	Code avantage	Nominal
Var13	benefitcount	Nombre d'avantages	Nominal
Var14	benpmtcode	Code de paiement des avantages	Nominal
Var15	ahedp	CK4* ou EDP	Scalaire
Var16	plnplan	Code produit	Scalaire
Var17	faceamount	Montant de couverture annuel	Scalaire
Var18	bfolben	Code de dossier bénéfice	Nominal
Var19	bender	Règles de dérivation des bénéfices	Nominal
Var20	dishospind	Type de couverture	Nominal
Var21	ifee	Indicateur de frais de service	Nominal
Var22	xreins	Indicateur de réassurance	Nominal
Var23	irating	Indicateur de code supplémentaire	Nominal
Var24	iincr	Indicateur d'augmentation des primes	Nominal
Var25	pfee	Frais d'utilisation du ROP	Nominal
Var26	reinsunit	Unités de réassurance	Scalaire
Var27	extraprem	Supplément de prime	Scalaire
Var28	duree-emission	Durée de l'émission	Scalaire
Var29	duree-rop	Période retour de prime	Scalaire
Var30	duree-ropmat	Période d'échéance Maturité	Scalaire
Var31	lastemission	Période de la dernière émission	Scalaire
Var32	hausprim-bool	Indicateur de la hausse de prime	Nominal
Var33	modal-premium	Prime modale	Scalaire
Var34	grossannual-prem	Prime de la couverture	Scalaire

Var35	totalamtpaid	Total AMT payé	Scalaire
Var36	valuationunits-1000	Nombre d'unités fois valeur	Scalaire
Var37	planvalunits-1000	Évaluation finale des unités	Scalaire
Var38	incanprem-1000	Montant de la hausse de prime	Scalaire
Var39	reinsprem	Prime de réassurance	Scalaire
Classe	Classe	Classe à prédire	Binaire

4.2 Comparaison des résultats des algorithmes

Dans cette section, on évalue les effets des différentes transformations sur les performances et la complexité de XGBoost. Pour les performances, on utilise la précision, le rappel, et la mesure F_1 pour la classe minoritaire. Ces métriques sont rapportées pour l'ensemble de validation. Pour la complexité, on rapporte le nombre de variables (ou la dimension de l'instance) utilisées pour prédire ainsi que le nombre total de feuilles présentent dans le modèle final, après validation des hyperparamètres. Le nombre de variables initial est de 19 pour les attributs scalaires et de 21 pour les attributs nominaux. Lors de l'encodage *one-hot*, les variables nominales se transforment en 264 variable binaire. Lors de la transformation par PCA ou RCA, les variables scalaires généreront le même nombre de composantes principales, et donc le même nombre de variables transformées. Cependant, pour PCA pondérée, à chaque itération de XGBoost, il y aura création de 19 composantes principales dont une ou plusieurs seront sélectionnées par l'arbre de régression appris. Le nombre de variables ne peut pas être estimé au préalable. Il est plutôt conditionné par le nombre d'itérations et l'apprentissage des arbres de régression.

Le tableau 4.2 présente les métriques de performance et de complexité mentionnées. Notons que ces performances sont rapportées après validation des hyperparamètres et sélection de colonnes par la méthode de sélection récursive supervisée des attributs (RFE).

Tableau 4.2 – Comparaison des effets des transformations sur les performances et la complexité de XGBoost

Transformation	**Nombre de**		**Performances pour la classe 1**		
	Variables	**Feuilles**	**F1**	**Précision**	**Rappel**
Aucune	**36**	23026	76,73	91,35	66,15
RCA	38	5463	89,34	98,41	81,80
PCA	40	8928	92,02	99,27	85,75
Encodage *one-hot*	38	50596	86,38	90,27	82,81
Encodage *one-hot* et RCA	94	4662	93,73	97,75	90,20
Encodage *one-hot* et PCA	117	4037	94,32	98,43	90,54
Encodage *one-hot* et PCAs pondérées	149	**1074**	**97,41**	**99,86**	**95,08**

Les transformations encodage *one-hot*, RCA, PCA, et PCAs pondérées, améliorent les métriques F_1, précision et rappel. Pour l'encodage *one-hot* , F_1 passe de $76,73\%$ à $86,38\%$ et que le nombre total de feuilles passe de 23026 à 50596. On voit également que l'utilisation de transformations RCA et PCA sur les attributs scalaires permet de décroître le nombre de feuilles apprises de 23026 feuilles à 5463 et 8928 feuilles respectivement en utilisant les attributs nominaux non-transformés, et de 50596 feuilles à 4662 et 4037 feuilles en utilisant les attributs nominaux encodés *one-hot* . Dans les deux cas (attributs nominaux encodés ou non), la diminution de la complexité par introduction des transformations RCA et PCA s'accompagne par une amélioration significative de F_1, de la précision et du rappel.

La transformation PCA a un meilleur effet sur les performances et la complexité de XGBoost que celui de RCA. On reporte une mesure $F_1 = 94,30\%$ avec 4037 feuilles. Cet effet est encore accentué lorsque la création et la sélection des composantes principales sont effectuées au niveau des itérations de XGBoost, avant l'apprentissage des arbres de régression. Précisément, elle permet d'atteindre les meilleures mesures F_1, précision et rappel égaux à $97,41\%$, $99,86\%$ et $95,08\%$ respectivement, avec seulement 1074 feuilles. Cette transformation sera alors retenue pour la sélection finale des feuilles et la simplification de XGBoost.

4.3 Analyse des comportements des algorithmes

Dans cette partie, on considère l'utilisation des attributs nominaux encodés et on décrit qualitativement les résultats de XGBoost en utilisant chacune des transformations RCA, PCA et PCAs pondérées. On s'intéresse particulièrement aux courbes d'apprentissage décrivant l'évolution du F_1 en fonction des itérations de XGBoost, et au compromis précision/rappel en variant le seuil sur la prédiction de la probabilité que l'instance soit de classe 1.

4.3.1 Courbes d'apprentissage

Étant donné que les courbes de variation de la mesure F_1 risquent d'être confondues lorsque $F_1 \simeq 1$, nous avons opté à la représentation de la métrique $1 - F_1$ en échelle logarithmique. Ceci a pour effet de séparer les courbes lorsque $F_1 \simeq 1$ et de les confondre lorsque $F_1 \simeq 0$. La valeur $1 - F_1$ sera notée F_1^c. Le c en exposant indique qu'il s'agit du complément de F_1 à 1. Puisqu'on souhaite maximiser la mesure F_1, alors, sur les courbes d'apprentissage, une plus faible valeur de F_1^c indique un meilleur résultat. Dans ce qui suit, nous allons noter E, V, T les ensembles d'entraînement, de validation et test respectivement.

XGBoost avec des données non transformées

La figure 4.1 illustre la variation de F_1^c pour les ensembles d'entraînement, de validation et de test. D'abord, l'algorithme s'est arrêté à l'itération 100, marquant l'itération à partir de laquelle, F_1^c de l'ensemble de validation cesse de diminuer. Aux dernières itérations, la courbe

associée à l'ensemble de validation est légèrement en dessous de celle de l'entraînement (une différence de $0,1$), ce qui indique un léger surapprentissage. Elle est également légèrement en dessous de celle de l'ensemble de test. Cela indique que, relativement à l'ensemble d'entraînement, l'ensemble de validation semble présenter plus de nouveautés que celui du test. Ce qui empêche la mesure F_1 de validation d'être plus importante que celle du test.

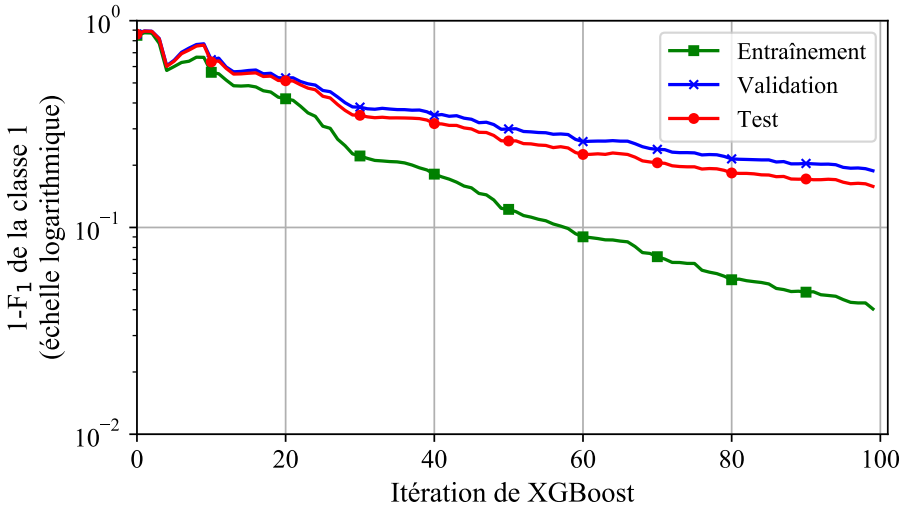


FIGURE 4.1 – Courbes d'apprentissage de XGBoost
sans transformation des données

XGBoost avec des données transformées par RCA

La figure 4.2 montre un comportement similaire à celui de XGBoost sans transformation des données avec certaines différences. Les valeurs finales de F_1^c pour les trois ensembles sont améliorées et l'écart entre F_1^c de l'ensemble d'entraînement et de validation est réduit ($\simeq 0,1$). Les courbes associées aux ensembles de validation et de test présentent une allure asymptotique (décroissance lente) à partir de la $100^{\text{ième}}$ itération alors que la courbe d'entraînement semble décroître constamment. Ceci indique que les règles de décisions apprises à partir de la $100^{\text{ième}}$ itération sont quasiment exclusives à l'ensemble d'entraînement, et n'impactent pas la mesure F_1 pour les ensembles de validation et de test. On identifie au moins deux explications possibles pour ce comportement :

— L'ensemble de validation est similaire à l'ensemble d'entraînement, mais le modèle appris présente un surapprentissage élevé ;

— L'ensemble de validation est très différent de l'ensemble d'entraînement au sens de la distribution des exemples. Il y aura alors toujours un écart important entre $F_1^c(E)$ et $F_1^c(V)$.

Dès qu'on trouve un classificateur à forte généralisation, on pourra conclure que la deuxième explication ne s'applique pas à nos données.

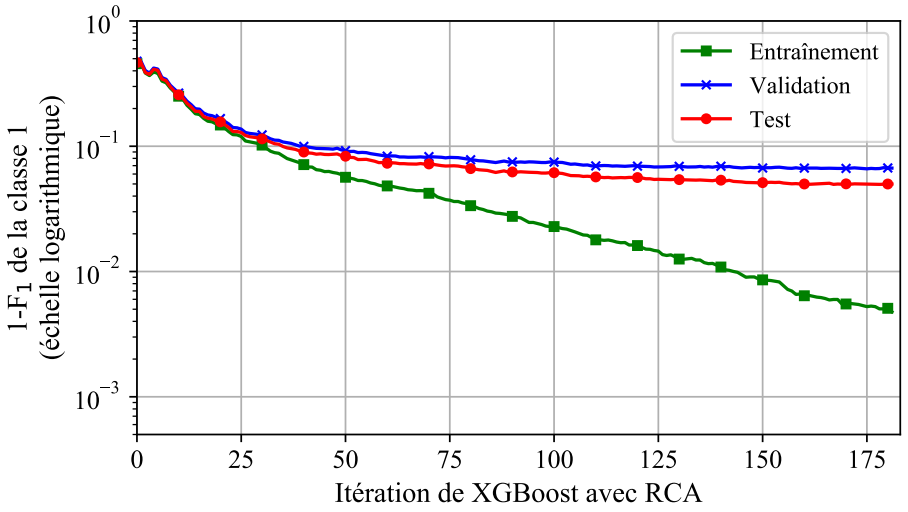


FIGURE 4.2 – Courbes d'apprentissage de XGBoost avec RCA

XGBoost avec des données transformées par PCA

On remarque sur la figure 4.3 que la métrique F_1^c d'entraînement décroît rapidement (exponentiellement, ou linéairement dans l'échelle logarithmique) pour converger asymptotiquement, à partir de l'itération 130, vers une valeur finale proche de 10^{-3}. Cette valeur est meilleure que celle obtenue sans transformation des données, à savoir, une valeur supérieure à 10^{-2}. La transformation des données par PCA a amélioré la capacité de XGBoost à représenter et à classifier les données d'entraînement.

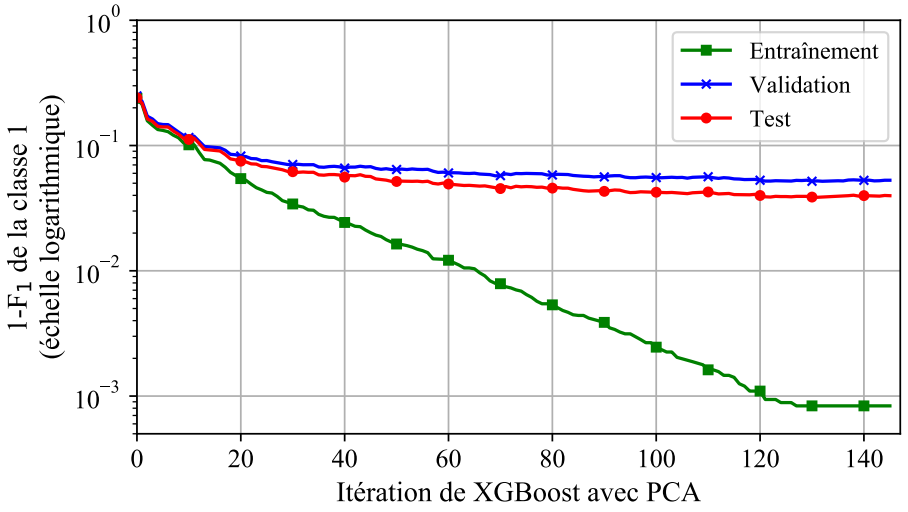


FIGURE 4.3 – Courbe d'apprentissage de XGBoost de XGBoost avec PCA

Les courbes de F_1^c pour les ensembles de validation et de test ont convergé depuis les premières

itérations. En effet, on voit que la courbe de validation décroît de plus en plus lentement à partir de l'itération 60 avant de croître vers l'itération 140 (arrêt prématuré). L'allure des courbes d'apprentissage d'entraînement et de validation indique que les règles de décision apprises grâce à PCA servent à bien classifier l'ensemble d'entraînement, mais ne sont pas totalement réutilisables pour bien classifier l'ensemble de validation. Finalement, on assiste à un écart important entre la valeur finale de la métrique d'entraînement et celle de validation, ce qui indique un surapprentissage.

XGBoost avec des données transformées par PCAs pondérées

On rappelle qu'à chaque itération de XGBoost, de nouvelles étiquettes et pondérations sont calculées et utilisées en deux occasions. La première consiste à apprendre des directions principales par analyse spectrale de la matrice XWX^T où W est une matrice diagonale contenant les pondérations. La seconde consiste à apprendre un arbre de régression minimisant l'erreur quadratique moyenne entre les prédictions et les nouvelles étiquettes correspondantes, et dont les termes sont pondérés par les pondérations fournies par XGBoost. À chaque itération, seules les composantes principales retenues par l'arbre de régression appris seront considérées comme utiles pour la classification. Elles sont alors acheminées aux itérations subséquentes afin de pouvoir les réutiliser.

La figure 4.4 montre la phase d'apprentissage des directions principales utiles pour la classification avec un ensemble d'arbres de régression.

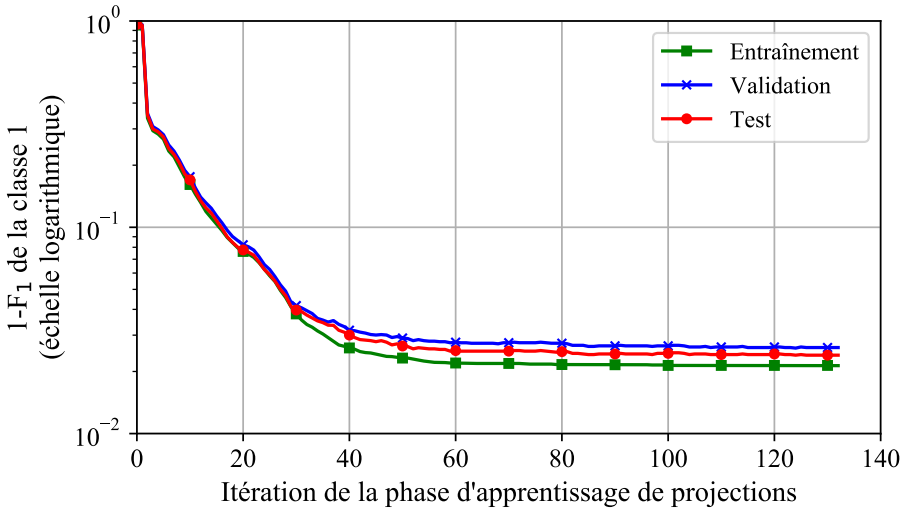


FIGURE 4.4 – Courbe d'apprentissage des conditions par XGBoost et PCA pondérée

Les trois courbes convergent asymptotiquement vers des valeurs finales meilleures que celles obtenues avec PCA. De plus, elles sont confondues aux 30 premières itérations avant de devenir presque confondus aux dernières itérations (entre 60 et 133). Cela indique d'une part une

meilleure généralisation de l'algorithme appris, et de l'autre que les ensembles d'entraînement, de validation et de test proviennent effectivement d'une même distribution. Il s'ensuit que l'allure asymptotique des courbes de validation avec PCA et RCA sont dues à l'incapacité du modèle de généraliser plutôt qu'à la présence de nouveautés dans l'ensemble de validation. Finalement, les courbes de validation et celle de l'entraînement cessent de décroître à la même itération. Ceci indique que les arbres peu utiles pour classifier l'ensemble d'entraînement sont également peu utiles pour classifier l'ensemble de validation. Ces arbres sont éventuellement corrélés avec des arbres appris aux premières itérations et il sera intéressant de les simplifier par sélection supervisée de branches utiles.

4.3.2 Courbes de précision-rappel

On s'intéresse à l'analyse de la variation de la précision de la classe 1 en fonction du rappel pour la même classe. D'abord, on utilise les prédictions effectuées par XGBoost pour générer des probabilités indiquant si une instance appartient à la classe 1. L'ensemble des probabilités est ensuite trié et utilisé pour générer des seuils. Ensuite, pour chaque seuil, on calcule la précision et le rappel pour un ensemble de données et on reporte un point de coordonnées (précision, rappel) sur la courbe de précision-rappel. Ensuite, on identifie un seuil qui maximise la mesure F_1 sur l'ensemble de validation et on marque le point correspondant sur la courbe par un cercle plein. Finalement, on réutilise le même seuil pour calculer un point sur la courbe de test. Ce dernier sera marqué par un carré plein. L'analyse des courbes de précision-rappel visera donc à :

— Trouver un seuil à appliquer sur la probabilité prédite par XGBoost afin d'améliorer la mesure $F_1(V)$

— Valider si le seuil identifié améliore également la mesure $F_1(T)$

— Comparer les transformations en termes de compromis entre la précision et le rappel

XGBoost avec des données non transformées

La figure 4.5 montre la variation de la précision de la classe 1, minoritaire, en fonction de son rappel. Les deux courbes présentent presque la même variation. Pour cette raison, et pour une meilleure visibilité, nous avons choisi de focaliser l'illustration aux valeurs de précision et de rappel variant entre 60% et 100%. En utilisant le seuil de probabilité 0.5 par défaut, on obtient une mesure $F_1 = 81,2\%$ pour l'ensemble de validation. Le seuil optimal $0,2402$ permet d'atteindre une mesure $F_1 = 86,38\%$ pour le même ensemble. Pour le même seuil, on rapporte également le point correspondant sur la courbe associée à l'ensemble de test et on observe une augmentation de la mesure F_1 de $84,2\%$ vers $86,38\%$. On remarque également que pour un rappel inférieur à 60%, le classificateur donne une précision proche de 100%. L'augmentation du rappel au-delà de 75% s'accompagne par une baisse de la précision. Cela signifie que le

prédicteur améliore le rappel en tolérant certaines erreurs de classification, d'où la baisse de précision.

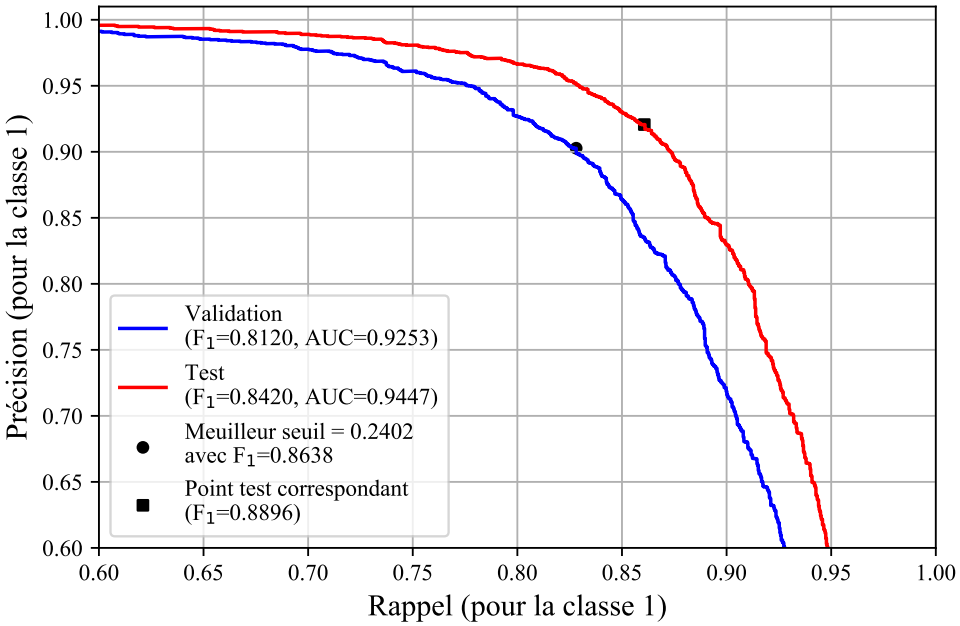


FIGURE 4.5 – Courbes précision/rappel de XGBoost sans transformation des données

XGBoost avec des données transformées par RCA

Pour RCA, la figure 4.6 montre un meilleur compromis de précision et de rappel. En effet, le classificateur permet d'atteindre une précision totale avec un rappel de 85%. Pour un rappel entre 85% et $90,5\%$, une légère diminution de la précision s'accompagne d'une augmentation considérable du rappel. Finalement, une faible augmentation du rappel s'accompagne par une baisse importante de la précision dès que le rappel dépasse $92,5\%$. Notons que le seuil de $0,2116$ identifié améliore la métrique F_1 pour les ensembles de test et de validation. En effet, $F_1(V)$ passe de $93,35\%$ à $94,48\%$ et $F_1(T)$ passe de $95,04\%$ à $95,8\%$.

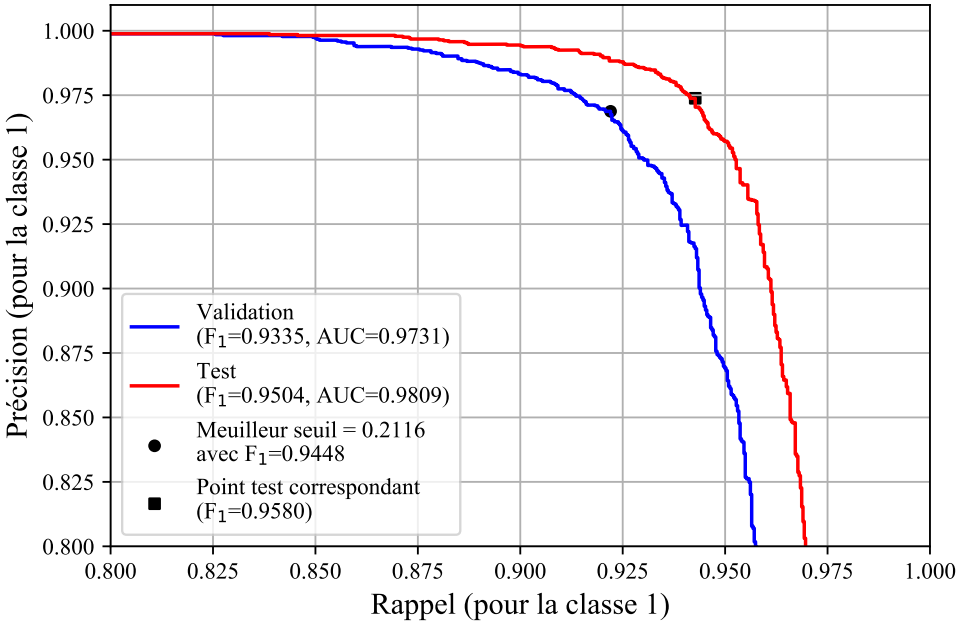


FIGURE 4.6 – Courbes précision/rappel de XGBoost avec RCA

XGBoost avec des données transformées par PCA

La figure 4.7 montre que pour la transformation PCA, on a un compromis de précision et de rappel similaire à celui obtenu avec RCA. Le classificateur atteint une précision totale avec un rappel de $87,5\%$. Pour un rappel entre $87,5\%$ et $92,5\%$, une légère diminution de la précision s'accompagne d'une augmentation considérable du rappel. Finalement, une faible augmentation du rappel s'accompagne par une baisse importante de la précision dès que le rappel dépasse $92,5\%$. Le seuil $0,3938$ identifié améliore la métrique F_1 pour les ensembles de test et de validation. En effet, $F_1(V)$ passe de $94,78\%$ à $94,92\%$ et $F_1(T)$ passe de $96,14\%$ à $96,15\%$. Notons que ces nouvelles valeurs sont également proches de celles obtenues avec RCA.

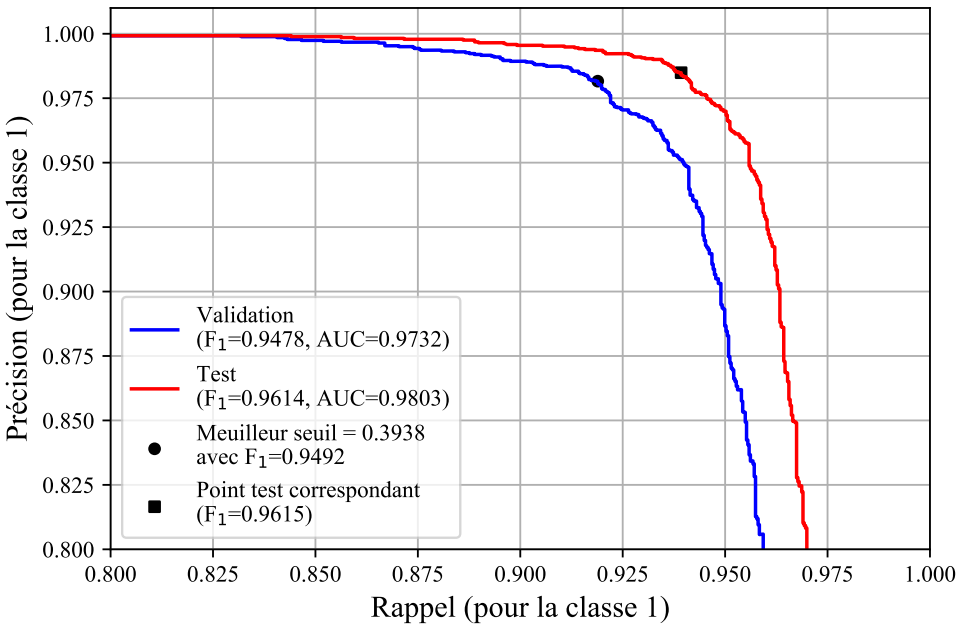


FIGURE 4.7 – Courbes précision/rappel de XGBoost avec PCA

XGBoost avec des données transformées par PCAs pondérées

La figure 4.8 montre le meilleur compromis précision-rappel parmi ceux obtenus avec les données brutes, PCA ou RCA. En effet, l'intervalle du rappel pour lequel une augmentation du rappel s'accompagne d'une diminution de la précision est le plus étroit et est de borne inférieure élevée. On parle d'un rappel entre 94% et 96%. Le maximum de la métrique $F_1(V)$ est atteint pour un seuil de $0,2031$, et il est plus élevé que $F_1(V) = 97,42\%$ obtenu avec un seuil de $0,5$. Pour le même seuil identifié, on assiste à une légère augmentation de $F_1(T)$. Elle passe de $97,61\%$ à $97,68\%$.

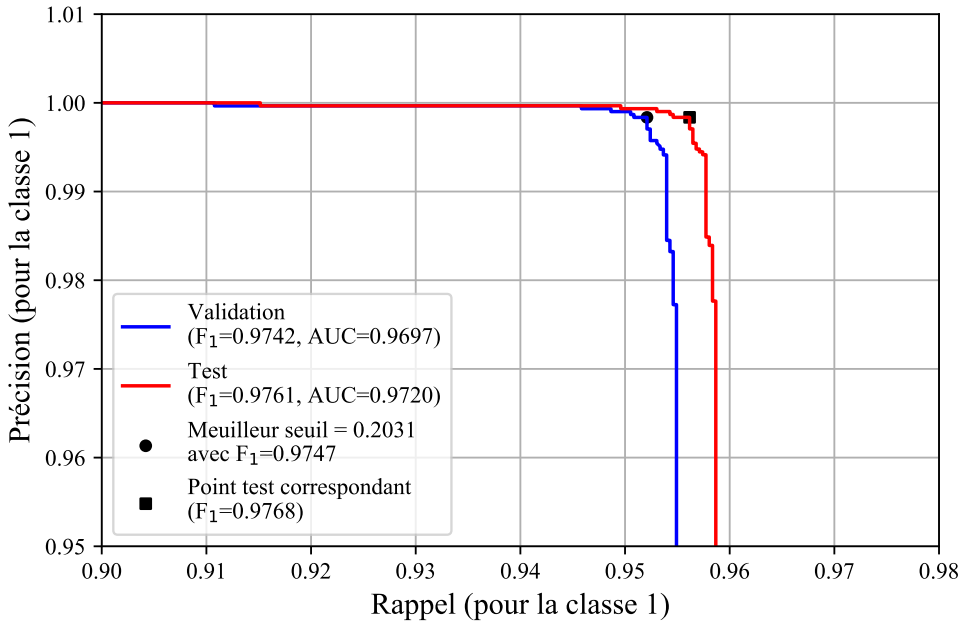


FIGURE 4.8 – Courbes précision/rappel de XGBoost avec les projections apprises

La transformation des données par des PCAs pondérées donne la meilleure convergence, les meilleurs résultats sur l'ensemble de validation avec un seuil de probabilités égal à 0.5 et un meilleur compromis précision-rappel. Elle donne également les meilleurs résultats sur l'ensemble de test lorsque le seuil de probabilité est optimisé pour maximiser $F_1(V)$. Cette transformation sera alors retenue pour la suite du chapitre afin de créer un modèle simplifié basé sur des branches des arbres de régression apprises par XGBoost.

4.4 Simplification du meilleur modèle

On retient la transformation des données par PCAs pondérées comme meilleure transformation des données. Les arbres de décisions apprises par XGBoost sont éclatés en branches. Ensuite, un ensemble d'arbres de décision de profondeur 1 est appris en utilisant les sorties binaires des branches comme instances. L'ensemble optimal des branches, le nombre total des arbres, et les hyperparamètres restants de XGBoost sont respectivement déterminés par RFE, arrêt prématuré, et validation hold-out. Une fois la phase d'apprentissage complétée, les branches sélectionnées sont examinées pour déterminer les variables initiales et les composantes principales à retenir.

La figure 4.9 illustre l'apprentissage des arbres de décisions de profondeur 1. Les valeurs de F_1^c démarrent entre 10^{-2} et 10^{-1} pour l'ensemble d'entraînement, validation et test. Ce qui représente des profils types (majoritaires) de clients qui renouvellent ou résilient et qui sont détectables avec une seule branche de trois seuils. On remarque que les trois courbes

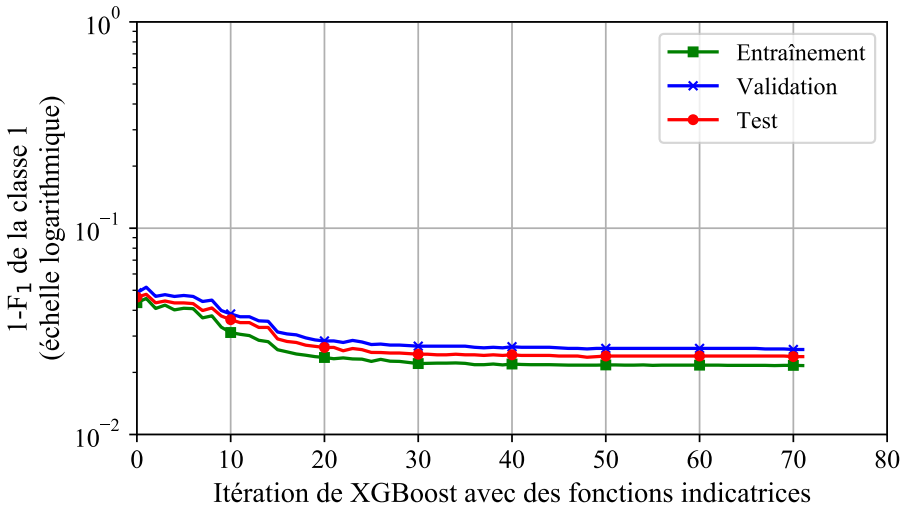


FIGURE 4.9 – Courbe d'apprentissage de XGBoost avec les conditions apprises

s'améliorent constamment avant de saturer à la 30ème itération. En plus elles sont quasi-confondues ce qui indique la forte capacité de généralisation du classificateur et l'absence du surapprentissage.

TABLEAU 4.3 – Tableau sommaire de la simplification de l'algorithme choisi et ses nouvelles performances

Approche	**Nombre de**			F_1 **pour la classe 1**	
	Variables en entrée	**Variables choisies**	**Conditions générées**	**Validation**	**Test**
XGBoost encodage one-hot et PCAs pondérées	287 : 19 scalaires 268 binaires	149 : 135 composantes principales générées, 14 variables sélectionnées	1074 branches, chacune est formée d'un à trois seuils	97,40	97,63
XGBoost simplifié, encodage one-hot et PCAs pondérées	1223 : 1074 conditions, 149 variables choisies par l'algorithme non simplifié	**30** : **29 conditions** à trois seuils, et utilisant 27 composantes, une variable binaire et une scalaire, **une condition** avec une variable binaire	118 branches d'arbres de profondeur 1 (59 seuils)	**97,53**	**97,77**

4.5 Analyse de la représentation apprise

La transformation t-SNE (van der Maaten and Hinton, 2008) est une méthode non-linéaire de réduction de dimensionnalité pour des fins de visualisation. Elle consiste à modéliser chaque instance par un point à deux ou trois dimensions de sorte à conserver les voisinages locaux. Si deux instances sont similaires dans les données originales, elles seront représentées par deux points proches avec la transformation t-SNE.

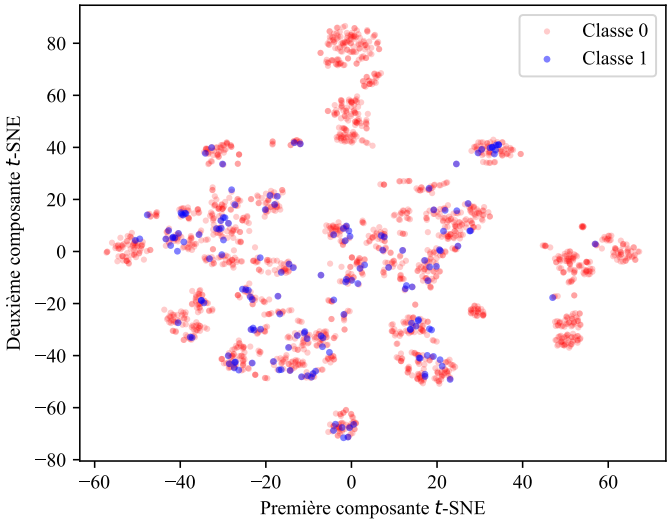


FIGURE 4.10 – Visualisation des données brutes

La figure 4.10 montre la transformation t-SNE à deux dimensions d'un échantillon des données brutes, on constate que les points de données appartenant à la même classe sont dispersés.

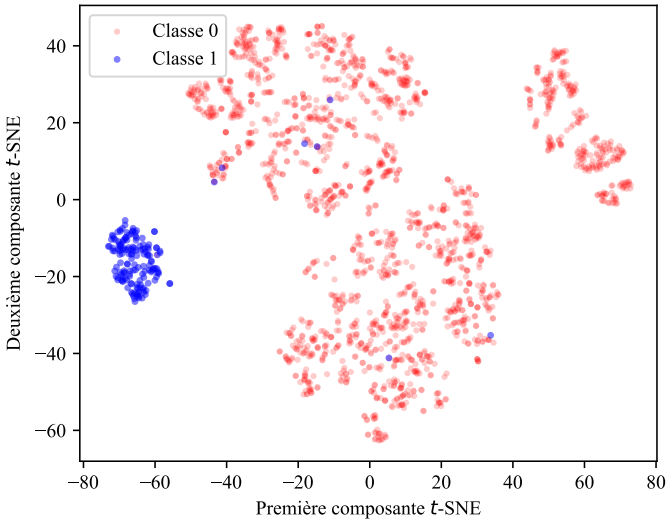


FIGURE 4.11 – Visualisation des données transformées par les conditions apprises

Contrairement aux données brutes, la figure 4.11, montre la représentation des conditions projetées par t-SNE, les points appartenant à la même classe sont des voisins regroupés dans le même sous-espace. Ce nouvel ensemble de données a donc fait en sorte que les ensembles de même classe deviennent plus proches. Un algorithme de *clustering* pourra donc identifier ce groupe. Il existe également quelques exemples granulaires de la classe +1 d'où le rôle d'un classificateur appris par apprentissage supervisé pour les détecter.

4.6 Interprétation des conditions apprises

Le tableau 4.4 liste les conditions retenues par le modèle final. Pour chaque condition, la première colonne indique son nom, la deuxième indique son expression comme séquence de un à trois seuils de décision, et la dernière colonne indique le gain relatif qu'elle réalise pour minimiser la fonction objective. Les conditions, les composantes principales générées, et les variables utilisées sont référées respectivement par les suffixes *cond*, *c*, et Var. Les conditions et les composantes principales sont indexées par leur ordre de création donné par l'algorithme non-simplifié. Comme l'algorithme simplifié a retenu 30 conditions pertinentes parmi 1047, les indices des conditions indiqueront les itérations les plus importantes. En particulier, on voit que ces indices varient entre 4 et 181. Cela indique que l'algorithme a rejeté les trois premières conditions comme simplistes (ou biaisées) et de nombreuses conditions complexes qui modélisaient un bruit (ou à haute variance). Ceci peut être vu comme une sélection du premier votant de XGBoost ultérieur à ceux appris par les premières itérations car ceux-ci risquent d'être simplistes. C'est donc une approche complémentaire à la sélection des votants dans XGBoost (sélection de la bonne itération de début et de celle de fin). Un comportement différent est observé dans les indices des composantes. En effet, les composantes sélectionnées incluent entre autres des composantes générées aux premières itérations (0, 4, 5) et aux dernières itérations (122, 131, 125) de XGBoost non-simplifié avec PCA pondérée.

TABLEAU 4.4 – Importance relative des conditions apprises

Nom	**Expression**	**importance relative**
$Var_{25,0}$	$Var_{25,0} = 1$	**.061224**
$cond_{59}$	$c_{122} > -.4915$ et $c_{42} > .0027$ et $c_{79} \leq -.0008$	**.061224**
$cond_{120}$	$Var_3 \leq .8576$ et $c_5 \leq .0076$ et $c_{50} \leq -.8379$	**.061224**
$cond_{181}$	$c_{54} \leq -.0050$ et $c_{68} > .1978$ et $c_{82} > -.0041$	**.061224**
$cond_{36}$	$c_{54} \leq .0814$ et $c_{79} > .0011$ et $c_{84} \leq -0.0011$	.040816
$cond_{44}$	$c_{103} \leq .3630$ et $c_{79} \leq .0075$ et $c_{84} > .0002$	.040816
$cond_{60}$	$c_{19} \leq .1152$ et $c_{66} \leq .0041$ et $c_{79} > -.0008$	.040816
$cond_{76}$	$c_{54} \leq -.0021$ et $c_{82} > -0.0028$ et $c_{84} \leq 0.0018$	.040816
$cond_{107}$	$Var_{34} \leq .2380$ et $c_{38} > -.002$ et $c_{79} > -.0028$	.040816
$cond_{117}$	$c_{131} > -4.0543$ et $c_{79} \leq .0109$ et $c_{84} > .0024$	.040816
$cond_{97}$	$c_0 > -.6888$ et $c_{109} \leq .0016$ et $c_{122} \leq -.0247$	.040816
$cond_{156}$	$c_{61} > -1.0669$ et $c_{80} \leq -.5087$ et $c_{93} \leq 1.3860$	.040816
$cond_{166}$	$c_{61} > -1.0669$ et $c_{70} \leq -.0025$ et $c_8 > -.3838$	.040816
$cond_{128}$	$Var_{34} \leq -.3407$ et $c_{57} \leq .0035$ et $c_{66} \leq .0081$	.040816
$cond_{136}$	$Var_{34} \leq .7115$ et $c_{57} \leq .0015$ et $c_{66} \leq .0070$	.040816
$cond_{15}$	$Var_{25,0} = 1$ et $c_{70} > .0013$ et $c_{79} > -.0013$	.020408
$cond_{21}$	$c_{125} \leq .0144$ et $c_{70} > .0013$ et $c_{74} > -1.7711$	.020408
$cond_{27}$	$Var_{25,0} = 1$ et $c_{84} \leq -0.0001$ et $c_{85} > -.2049$	.020408
$cond_{33}$	$c_{70} \leq .0009$ et $c_{79} \leq .0011$ et $c_{84} > -.0017$	.020408
$cond_{175}$	$c_{112} > -.7211$ et $c_{50} > -.4535$ et $c_{82} > -.0037$	.020408
$cond_{38}$	$c_{54} > .0814$ et $c_{60} \leq 13.8527$ et $c_{79} > .0011$	.020408
$cond_{124}$	$c_4 \leq -.0181$ et $c_{50} > -.8379$ et $c_{63} \leq 1.5184$	.020408
$cond_{51}$	$c_{42} > -.0984$ et $c_{53} > -.3523$ et $c_{84} \leq .0008$	.020408
$cond_{115}$	$c_{83} > -.4727$ et $c_{84} \leq .0024$ et $c_{84} > 0$	.020408
$cond_{144}$	$c_{110} \leq .9241$ et $c_{68} \leq .0038$ et $c_{70} \leq -.0007$	.020408
$cond_{65}$	$c_{107} > -2.0461$ et $c_{122} \leq -.0125$ et $c_{84} \leq .0006$	.020408
$cond_{81}$	$c_{122} \leq -.0208$ et $c_{59} > -2.3748$ et $c_{66} \leq .0065$	.020408
$cond_{135}$	$Var_{34} > -.3407$ et $c_{125} > -.1277$ et $c_{42} > .0025$	.020408
$cond_{100}$	$c_{106} \leq -.0845$ et $c_{122} > -.0247$ et $c_8 \leq -.0452$	.020408
$cond_4$	$c_{26} \leq 1.7387$ et $c_{56} \leq .0455$ et $c_{70} > .0016$	.020408

Les seuils sur les conditions, les composantes principales et les variables scalaires s'interprètent comme suit :

— $cond_i$ signifie que la condition d'indice i est vérifiée, et équivaut à $cond_i = 1$ ou $cond_i \geq .5$. Cela signifie que toutes les comparaisons qu'elle contient sont vérifiées. Sa négation est $\neg cond_i$, qui signifie $cond_i = 0$. Cela arrive quand au moins une de ses comparaisons est non-vérifiée.

— $c_i \leq \theta$ signifie que la projection de l'instance $\mathbf{x}$ selon la composante $\mathbf{pc}_i$, de norme unité, est inférieure ou égale à un seuil θ. Pour une instance $\mathbf{x}$, c_i est défini comme un produit scalaire $< \mathbf{pc}_i, \mathbf{x} > = ||\mathbf{x}||\ ||\mathbf{pc}_i||\ cos(\mathbf{x}; \mathbf{pc}_i)$ où la norme de l'instance est calculée seulement en utilisant ses coordonnées non-binaires. La comparaison $c_i \leq \theta$ peut aussi être interprétée comme une similarité entre le vecteur créé par les variables non-binaires de $\mathbf{x}$ et la direction $\mathbf{pc}_i$:

$$cos(\mathbf{x}; \mathbf{pc}_i) \leq \frac{\theta}{||\mathbf{x}||\ ||\mathbf{pc}_i||} = \frac{\theta}{||\mathbf{x}||}$$

— $Var_i \leq \theta$ signifie que la variable scalaire normalisée (par standardisation) est inférieure ou égale à un seuil appris. Afin de donner un sens à ce seuil, on utilise la moyenne et l'écart type μ_i et σ_i utilisés dans la normalisation pour retrouver un seuil dans le domaine de la variable i brute (avant normalisation).

$$Var_i^{brute} \leq \sigma_i(\theta) + \mu_i$$

Le score de l'importance relative indique l'apport de chaque condition dans la séparation des classes de l'ensemble de l'entraînement. L'utilisation de variables binaires définies par des seuils sur des variables scalaires présente un avantage quant à l'interprétablitié. Par exemple, dire que "$Var_3 < .8$ sépare bien les données" donne plus d'information que de simplement dire "Var_3 est important". En utilisant la signification des attributs du tableau 4.1, on peut facilement exprimer les conditions en langage naturel. Par exemple, les deux premières conditions du tableau 4.4 sont que "le client paye des frais d'utilisation du ROP" et que "le client est plus jeune que 58 ans, et les projections c_5 et c_{50} sont inférieures respectivement à 0.0076 et -0.8379". Or, c_5 est la coordonnée de l'instance selon la direction $\mathbf{pc}_5$, dont la projection sur l'espace de variables est donné au tableau 4.5. On remarque que cette coordonnée est dominée par la coordonnée Var37. Alors la comparaison $c_5 \leq .0076$ est presque équivalente à dire $-0.98 \times Var37 \leq .0076$ ou encore $Var37 \geq 0,0077$. En utilisant les données non-transformées, cette condition est équivalente à $planvalunits_{1000} \geq 0$. Le même processus peut être suivi pour analyser la comparaison $c_{50} \leq -.8379$. La deuxième condition se lit alors comme "le client est plus jeune que 58 ans, a une évaluation finale des unités plus grande que 0, et le montant de la hausse de prime est faible."

Tableau 4.5 – Projection des composantes principales 5 et 50

Var	$\mathbf{pc}_5$	$\mathbf{pc}_{50}$
Var3	-0.003306	-0.000118
Var9	-0.012096	0.008958
Var10	-0.047575	0.00046
Var15	-0.000004	0.001528
Var16	-0.000545	0.000052
Var17	0.001681	-0.000002
Var26	-0.000419	-0.013760
Var27	-0.008570	-0.003336
Var28	0.016368	0.001547
Var29	-0.001982	0.002211
Var30	0.011068	0.001096
Var31	-0.029167	-0.007134
Var33	0.001788	-0.000147
Var34	-0.002159	-0.000499
Var35	-0.001819	-0.045703
Var36	0.137522	0.039023
Var37	-0.988435	-0.041052
Var38	0.008980	0.056495
Var39	-0.015787	-0.002561

Les scores du tableau 4.4 montrent la contribution relative de chaque condition à la minimisation de la fonction objective. Avec un arbre de décision de profondeur 1, le seuil appris divise l'espace en deux régions disjointes et associe un poids à chaque région. Lorsque les deux poids sont à forte valeur absolue, la condition est importante pour la prédiction des deux classes. Lorsque l'un des poids est presque nul (distribution aléatoire d'exemples positifs et négatifs) et que le deuxième est à forte valeur absolue (région dominée par des exemples de même classe) alors le seuil est important pour l'une des classes exclusivement. En plus, si son gain est très faible, alors on peut s'attendre qu'il apparaisse aux derniers rangs du tableau 4.4 alors qu'il est un des seuils les plus importants pour l'une des classes.

Pour mieux comprendre l'apport de chaque condition, il sera intéressant d'illustrer la classe qu'elle isole en fonction de sa valeur (condition vérifiée ou non). Ceci est représenté dans le tableau 4.6. On observe que $Var_{25,0} = 0$, $cond_{120} = 1$ et $cond_{59} = 1$ augmentent les chances qu'une instance appartienne à la classe 0. Ces trois conditions sont aussi les plus importantes dans le tableau 4.4. Toutefois pour la classe 1 l'ordre est diffèrent. $cond_{76} = 1$, $cond_{181} = 1$ et $cond_{136} = 1$ augmentent les chances qu'une instance appartienne à la classe 1 alors qu'elles apparaissent respectivement aux 8ème, 4ème et 15ème rangs dans le tableau 4.4.

Par construction de notre algorithme, une instance apparie par défaut à la classe 0. Ainsi, une instance prédite de classe 1 ira satisfaire une ou plusieurs conditions de la colonne de cette classe dans le tableau 4.4. La figure 4.12 montre, en utilisant un échantillon de 1000

exemples d'entraînement, comment les instances changent progressivement de classe lors de l'ajout progressif des conditions de la classe 1.

TABLEAU 4.6 – Contribution des conditions pour chaque classe

Classe 1		**Classe 0**	
Condition	**Poids**	**Condition**	**Poids**
$cond_{76}$	2.485683	$\neg\mathbf{Var_{25,0}}$	-3.234091
$\mathbf{cond_{181}}$	2.359328	$\mathbf{cond_{120}}$	-1.920960
$cond_{136}$	2.159906	$\mathbf{cond_{59}}$	-1.906313
$cond_{128}$	2.145617	$cond_{156}$	-1.755519
$cond_{107}$	1.811129	$cond_{166}$	-1.473719
$cond_{15}$	1.710164	$cond_{117}$	-1.418940
$cond_{36}$	1.558582	$\neg cond_{15}$	-1.249417
$cond_{97}$	1.382636	$cond_{44}$	-1.212738
$cond_{4}$	1.324792	$cond_{38}$	-1.139946
$cond_{60}$	1.268960	$\neg cond_{4}$	-1.072865
$cond_{65}$	1.219348	$\neg cond_{21}$	-1.009431
$cond_{51}$	1.210255	$cond_{33}$	-0.982499
$cond_{115}$	1.194033	$\neg cond_{27}$	-0.977018
$cond_{100}$	1.169648	$cond_{144}$	-0.971192
$\neg cond_{44}$	1.131399	$\neg cond_{51}$	-0.693772
$cond_{21}$	1.122885	$\neg\mathbf{cond_{181}}$	-0.680818
$\neg\mathbf{cond_{59}}$	1.106845	$\neg cond_{107}$	-0.648702
$cond_{81}$	1.057580	$cond_{135}$	-0.557180
$cond_{27}$	1.030630	$\neg cond_{60}$	-0.530837
$cond_{124}$	.937341	$\neg cond_{128}$	-0.462963
$\neg cond_{33}$	.904567	$\neg cond_{97}$	-0.443435
$cond_{175}$	.866677	$\neg cond_{76}$	-0.391117
$\mathbf{Var_{25,0}}$	.625692	$\neg cond_{136}$	-0.379155
$\neg\mathbf{cond_{120}}$	.491167	$\neg cond_{124}$	-0.354038
$\neg cond_{117}$	.472281	$\neg cond_{36}$	-0.341196
$\neg cond_{166}$	.404703	$\neg cond_{175}$	-0.337094
$\neg cond_{144}$	.338384	$\neg cond_{81}$	-0.256399
$\neg cond_{156}$	0.321445	$\neg cond_{115}$	-0.247507
$\neg cond_{38}$	0.230459	$\neg cond_{65}$	-0.246213
$\neg cond_{135}$	0.156374	$\neg cond_{100}$	-0.155336

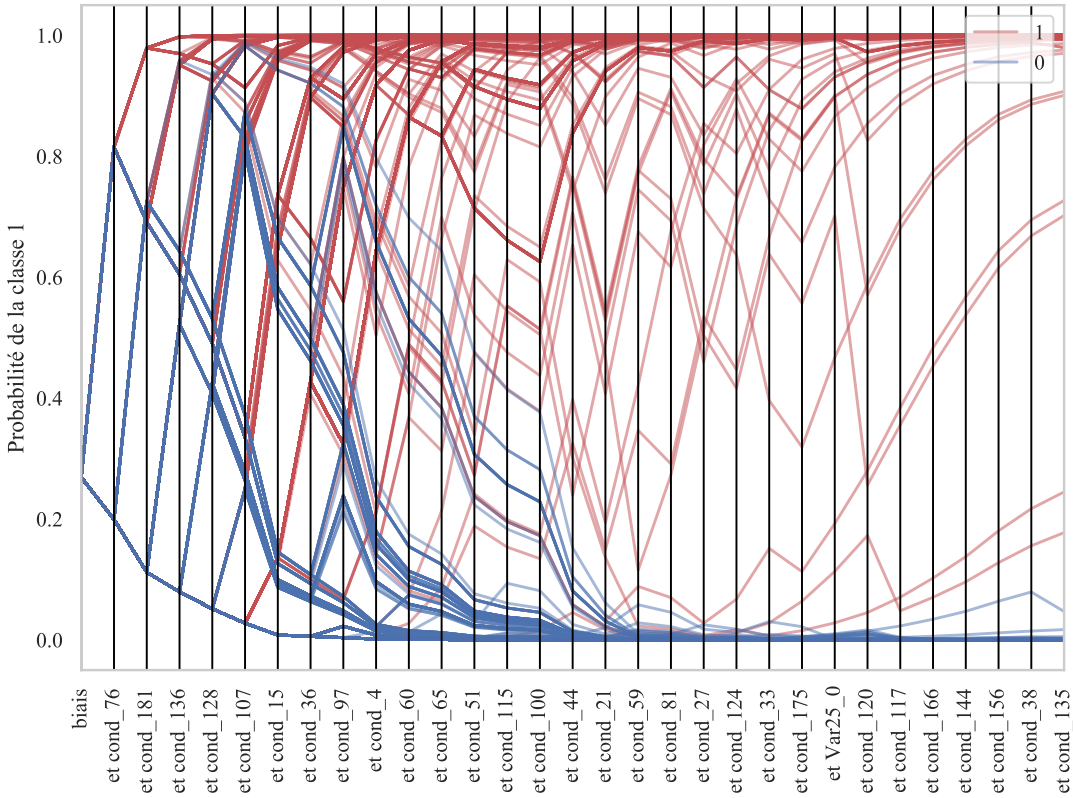


FIGURE 4.12 – Visualisation de 1000 instances classées par les conditions apprises.

On observe que certains exemples de classe 1 (courbes rouges) sont classés correctement avec des probabilités $> .5$ à partir des quelques premières conditions. D'autres plus difficiles manifestent une augmentation de la probabilité (initialement nulle) à partir de la $21^{\text{ième}}$ condition ($cond_{33}$) avant d'atteindre une probabilité $> .5$ dans la majorité des cas. On remarque également plusieurs points où des instances de classe 0 (courbes bleues) qui avaient reçu une probabilité élevée de classe 1 par les conditions validées initialement voient cette probabilité décroître rapidement. Les trois instants les plus marqués sont aux $cond_{76}$, $cond_{136}$, et $cond_{107}$. Dans les trois cas, les instances avaient atteint des valeurs de probabilité assez élevés ($\simeq .8$, $\simeq .9$ et $\simeq 1$ respectivement), puis les conditions suivantes les ont diminuées jusqu'à des valeurs finales presque nulles.

Conclusion

L'algorithme XGboost avec PCA pondérée a donné les meilleures performances et a été choisi pour la phase de simplification en combinaison de fonctions indicatrices. Cette étape, comme elle était effectuée par un algorithme supervisé, s'est accompagnée par une augmentation de performance et d'une meilleure généralisation. En plus, le nombre de conditions finales est tellement réduit qu'il est inférieur à l'ensemble d,attributs initiaux. De plus chaque fonction

contribue à la prédiction seulement lorsqu'elle est activée, ce qui rend le diagramme d'importance des conditions plus significatif que celui des attributs. Ces deux caractéristiques contribuent à une meilleure interprétabilité. Finalement, l'espace généré par ces conditions sépare linéairement les données, il devient donc adapté à une grande famille d'algorithmes d'apprentissage automatique, supervisée ou non.

Conclusions et perspectives

Ce travail aborde la tâche de prédiction de la rétention comme une classification binaire avec des données déséquilibrées. La revue des méthodes existantes a démontré qu'elles sont équivalentes à pondérer efficacement les observations pour mettre en valeur la classe d'intérêt, minoritaire. Parmi les algorithmes performants pour cette tâche, XGBoost se distingue comme une approche générique pour tout choix de la fonction de perte, implémentant la pondération des exemples à partir d'une descente de gradient de l'erreur dans l'espace des votants, et permet de trier les attributs par leur pertinence pour la prédiction. Cette dernière caractéristique le rend adapté à des méthodes heuristiques de sélection des attributs. Cela a un impact important dans l'interprétabilité.

En plus d'enrichir la classe des frontières de décision apprises par XGBoost sans nuire à l'interprétabilité ni aux performances, l'approche proposée apporte cinq contributions.

— Générer les directions principales candidates à sélectionner par les votants à chaque itération de XGBoost ;

— Adapter l'algorithme de génération des directions pour tenir compte des observations ayant les poids les plus élevés, à chaque itération ;

— Retenir à chaque itération, les directions choisies par le votant pour une réutilisation éventuelle à une itération ultérieure ;

— Éliminer par un algorithme supervisé, les feuilles inutiles des arbres appris (élagage d'XGBoost) ;

— Créer une méthode, efficace et indépendante de la distribution des données (recherche exhaustive rapide), de calcul d'un ensemble réduit de fonctions permettant de projeter les données dans un espace où elles deviennent linéairement séparables ;

La méthode est validée sur des données synthétiques et testée sur les données de la LCSF, où les résultats étaient spectaculaires.

Comme perspective de travaux futurs, on propose de créer une architecture de réseaux de neurones qui exploite des propriétés clés des méthodes de *boosting*. Précisément, pour poursuivre les travaux, on souhaiterait implémenter :

— La descente de gradient de l'erreur dans l'espace des architectures permettant d'incrémenter itérativement (ou à chaque époque) l'architecture (largeur des couches et leur nombre) en garantissant que chaque ajout déplace les performances vers la diminution de la fonction objective ;

— Une architecture adversaire qui défait les performances du réseau de neurones afin d'apprendre de nouveaux neurones qui lui sont complémentaires ;

Annexe A

Propriétés de l'algorithme Adaboost

A.1 Énoncé de l'algorithme

Adapté de (Schapire, 2003)

A.2 Propriétés

Propriété A.2.1. *Reproduite de (Schapire, 2003)*

Soit $t \in \{1, ..., T\}$. L'erreur de classification de $h^{(t)}$ pondérée par $\mathcal{D}^{(t+1)}$ est exactement égale à $\frac{1}{2}$.

$$\sum_{i=1}^{n} d_i^{(t+1)}.\mathbb{1}[h_t(\mathbf{x}_i \neq y_i)] = \frac{1}{2}$$

Preuve.

$$\begin{aligned}
\sum_{i=1}^{n} d_i^{(t+1)}\mathbb{1}[h^{(t)}(\mathbf{x}_i) \neq y_i] &= \sum_{i=1}^{n} d_i^{(t+1)}\mathbb{1}[h^{(t)}(y_i.\mathbf{x}_i) = -1] \\
&= \frac{\sum_{i=1}^{n} d_i^{(t)}.exp\big(-\alpha_t.y_i.h^{(t)}(\mathbf{x}_i)\big)\mathbb{1}[y_i.h^{(t)}(\mathbf{x}_i) = -1]}{\sum_{j=1}^{n} d_j^{(t)}.exp\big(-\alpha_t.y_j.h^{(t)}(\mathbf{x}_j)\big)} \\
&= \frac{e^{\alpha_t}.\sum_{i=1}^{n} d_i^{(t)}\mathbb{1}[h^{(t)}(\mathbf{x}_i).y_i = -1]}{\sum_{j=1}^{n} d_i^{(t)}\Big(e^{-\alpha_t}\mathbb{1}[h^{(t)}(\mathbf{x}_i) = y_i] + e^{\alpha_t}\mathbb{1}[h^{(t)}(\mathbf{x}_i) \neq y_i]\Big)} \\
&= \frac{e^{\alpha_t}.\epsilon_t}{e^{-\alpha_t}.(1-\epsilon_t) + e^{\alpha_t}.\epsilon_t} \\
&= \frac{\epsilon_t}{\epsilon_t + e^{-2\alpha_t}(1-\epsilon_t)}
\end{aligned}$$

Algorithme 1 : Adaboost

Données : $\mathcal{S} = \{(\mathbf{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}, i \in \{1, ..., n\}\}$; `/* l'ensemble d'entraînement */`

Données : $\mathcal{D} = \{d_1, d_2, ..., d_n\}$ tel que $\sum_{d \in \mathcal{D}} d = 1$; `/* poids des observations */`

Données : Un algorithme d'apprentissage des votants $ABase(\bullet, \bullet)$ supportant la pondération des exemples

Données : $T \in \mathbb{N} \setminus \{0\}$; `/* le nombre d'itérations */`

Initialisation : $\mathcal{D}^{(1)} = \mathcal{D}$

1 **pour** $t = 1, ..., T$ **faire**

2 Apprendre un votant $h^{(t)} \leftarrow ABase(\mathcal{S}, \mathcal{D}^{(t)})$; `/*` $h^{(t)} : \mathcal{X} \longrightarrow \{\pm 1\}$ `*/`

3 Calculer l'erreur de classification zéro-un pondérée

$$\epsilon_t = \sum_{i=1}^{n} \mathcal{D}^{(t)}(i).\mathbb{1}[h^{(t)}(\mathbf{x}_i) \neq y_i] = \sum_{i=1}^{n} d_i^{(t)}.\mathbb{1}[h^{(t)}(\mathbf{x}_i) \neq y_i]$$

4 Calculer le poids $\alpha_t = \frac{1}{2} \, . \, ln\big(\frac{1-\epsilon_t}{\epsilon_t}\big)$

5 Mettre à jour les pondérations $\forall \; i \in \{1, ..., n\}$

$$d'_i \leftarrow d_i^{(t)} \, . \, exp(-y_i \; \alpha_t \; h^{(t)}(\mathbf{x}_i))$$

$$d_i^{(t+1)} \leftarrow \frac{d'_i}{Z_{t+1}} \text{ avec } Z_{t+1} = \sum_{i=1}^{n} d'_i$$

6 **fin**

Résultat : Un classificateur binaire $h : \mathcal{X} \longrightarrow \{\pm 1\}$ défini par :

$$\forall \mathbf{x} \in \mathcal{X}, h(\mathbf{x}) = signe(g^{(T)}(\mathbf{x})) = signe\big(\sum_{t=1}^{T} \alpha_t.h^{(t)}(\mathbf{x})\big)$$

Or, par définition de α_t, nous avons :

$$\begin{aligned} 2\alpha_t &= ln\big(\frac{1-\epsilon_t}{\epsilon_t}\big) \\ \Leftrightarrow \quad e^{-\alpha_t} &= \frac{\epsilon_t}{1-\epsilon_t} \end{aligned}$$

Ainsi, l'erreur de classification de $h^{(t)}$ pondérée par $\mathcal{D}^{(t+1)}$ devient :

$$\sum_{i=1}^{n} d_i^{(t+1)} \mathbb{1}[h^{(t)}(\mathbf{x}_i) \neq y_i] = \frac{\epsilon_t}{\epsilon_t + \frac{\epsilon_t}{1-\epsilon_t}(1-\epsilon_t)} = \frac{\epsilon_t}{\epsilon_t + \epsilon_t} = \frac{1}{2}$$

□

Dans le cas de données équilibrés pondérées par une distribution uniforme $\mathcal{D}^{(0)} = \{\frac{1}{n}, ..., \frac{1}{n}\}$, on a :

$$\sum_{i=1}^{n} \mathbb{1}[y_i = 1] = \sum_{i=1}^{n} \mathbb{1}[y_i = -1] \tag{A.1}$$

Soit c_{-1} le classificateur constant prédisant la classe majoritaire -1.

$$\forall \mathbf{x} \in \mathcal{X}, c_{-1}(\mathbf{x}) = -1$$

Ce classificateur a une erreur $\epsilon = \frac{1}{2}$ sur ces données équilibrées.

Toutefois, dans le cas de données déséquilibrés où la classe d'intérêt $+1$ est minoritaire, et pondérées uniformément :

$$\sum_{i=1}^{n} d_i \mathbb{1}[y_i = 1] \ll \sum_{i=1}^{n} d_i \mathbb{1}[y_i = -1]$$

c_{-1} donne une très faible erreur de classification $\epsilon \simeq 0$ sans détecter la classe minoritaire. Cette mesure est donc trompeuse et pour la corriger, une des méthodes consiste à re-pondérer les observations.

Grâce à la propriété A.2.1, Adaboost implémente un méchanisme d'équilibrage des pondérations. En effet, en posant $h^{(0)} = c_{-1}$ et en effectuant une seule itération d'Adaboost, la nouvelle distribution $\mathcal{D}^{(1)}$ pondère les observations de sorte que $h^{(0)}$ donne une erreur égale à $\frac{1}{2}$ au sens de $\mathcal{D}^{(1)}$.

Il s'ensuit que Adaboost est naturellement adapté au traitement des données déséquilibrés. Adaboost requiert que l'algorithme d'apprentissage de base retourne un votant légèrement meilleur qu'un choix aléatoire. Ces votants sont donc des algorithmes simples. Avec des données déséquilibrés pondérés équitablement, ces algorithmes n'auront pas assez de complexité pour séparer la classe minoritaire et pourront donc converger vers un classificateur proche de c_{-1} (prédisant -1 pour la majorité de la classe minoritaire). Ceci n'aura pas d'impact sur l'apprentissage des prochains votants grâce au calcul de la nouvelle distribution qui balance la contribution de la classe $+1$ dans l'erreur de classification.

Propriété A.2.2. *Inspirée de (Kégl, 2013) et (Schapire, 2003)*

Soit $t \in \{1, ..., T\}$ et $h^{(t)}$ un classificateur produit à l'itération t par l'algorithme d'apprentissage de base. En notant $\gamma_t = \sum_{i=1}^{n} d_i^{(t)}.y_i.h^{(t)}(\mathbf{x}_i)$, nous avons :

$$\alpha_t = \frac{1}{2} ln\Big(\frac{1+\gamma_t}{1-\gamma_t}\Big) \tag{A.2}$$

Preuve. Soit $t \in \{1, ..., T\}$ et $h^{(t)}$ un classificateur produit à l'itération t par l'algorithme

d'apprentissage de base. Notons $\gamma_t = \sum_{i=1}^{n} d_i^{(t)}.y_i.h^{(t)}(\mathbf{x}_i)$. On a alors :

$$\begin{aligned}
\gamma_t &= \sum_{i=1}^{n} d_i^{(t)}.y_i.h^{(t)}(\mathbf{x}_i) \\
&= \sum_{i=1}^{n} d_i^{(t)} \mathbb{1}[h^{(t)}(\mathbf{x}_i) = y_i] - \sum_{i=1}^{n} d_i^{(t)} \mathbb{1}[h^{(t)}(\mathbf{x}_i) \neq y_i] \\
&= (1 - \epsilon_t) - \epsilon_t \\
&= 1 - 2.\epsilon_t \Leftrightarrow \epsilon_t = \frac{1 - \gamma_t}{2}
\end{aligned}$$

Ainsi, α_t s'exprime en fonction de γ_t comme suit.

$$\begin{aligned}
\alpha_t &= \frac{1}{2} ln\Big(2\frac{1 - \frac{1-\gamma_t}{2}}{1 - \gamma_t}\Big) \\
&= \frac{1}{2} ln\Big(\frac{1 + \gamma_t}{1 - \gamma_t}\Big)
\end{aligned}$$

□

Un classificateur performant $h^{(t)}$ ayant une erreur $\epsilon_t = 0$ (ou $\gamma_t = 1$) est associé à un poids infini. En pratique, cela veut dire qu'on retient ce classificateur et on rejette les classificateurs précédemment produits. L'ensemble est donc réduit à $h^{(t)}$. Cependant, les classificateurs ayant une faible erreur $\epsilon_t \simeq 0$ se voient associés à des poids élevés. De plus, une faible différence entre deux classificateurs performants se manifeste par une grande différence entre les poids. Ainsi, pour stabiliser les poids d'Adaboost, il est courant (scikit-learn et multiboost) d'utiliser la demi marge moyenne à la place de la marge moyenne dans le calcul du poids α_t.

$$\alpha_t(\gamma_t/2) = \frac{1}{2} ln\Big(\frac{1 + \frac{\gamma_t}{2}}{1 - \frac{\gamma_t}{2}}\Big) \tag{A.3}$$

Ceci est équivalent à forcer la marge moyenne à être dans l'intervalle $[0, 1/2]$ en divisant sa valeur par 2 ou bien en utisant une nouvelle fonction $\alpha(\gamma/2)$, définie par A.3 qui déplace l'asymptote $\gamma = 1$ associée à A.2 vers $\gamma = 2$, au-delà des valeurs possibles de γ. Dans cette configuration, et comme la variation de $\alpha(\gamma/2)$ est presque linéaire sur $[0, \frac{1}{2}]$, il est pertinent de l'approcher par son développement limité d'ordre 1 autour de $\gamma = 0$.

$$\forall \gamma \in [0, 1],\ \alpha(\gamma/2) \simeq \gamma$$

La figure suivante illustre cette approximation.

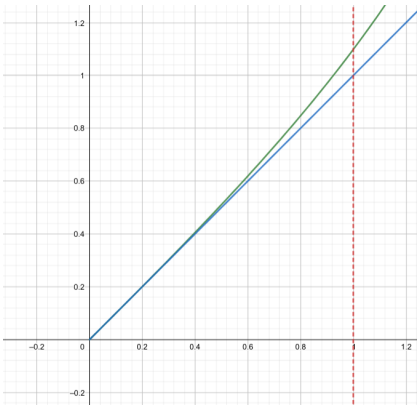


FIGURE A.1 – Approximation efficace du calcul du poids α à une itération de Adaboost

Pour $\epsilon_t = 1/2$ (ou encore $\gamma_t = 0$), Adaboost associe un poids nul au classificateur $h^{(t)}$. La distribution $\mathcal{D}^{(t+1)}$ est identique à $\mathcal{D}^{(t)}$ et l'algorithme cesse de progresser. Cela marque l'arrêt de l'apprentissage et il est conseillé de changer l'algorithme $ABase$.

Propriété A.2.3. *Reproduite de (Schapire, 2003)*

Si à chaque itération t, $ABase(\mathcal{D}^{(t)}, \mathcal{S})$ retourne une classificateur $h^{(t)} : \mathcal{X} \longrightarrow \{\pm 1\}$ légèrement meilleur qu'un classificateur aléatoire au sens de $\mathcal{D}^{(t)}$, c'est à dire :

$$\exists \delta tel\ que\ \forall t \in \{1, ..., T\}, \epsilon_t \leq \frac{1}{2} - \delta$$

alors le classificateur final $H^{(T)}$ retourné par Adaboost vérifie :

$$E_{0-1}(H^{(T)}, \mathcal{S}) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}[H^{(t)}(\mathbf{x}_i) \neq y_i] \leq e^{-2\ \delta^2\ T} \tag{A.4}$$

Démonstration. Comme pour toute itération t, l'algorithme de base retourne un classificateur $h^{(t)}$ meilleur d'un choix aléatoire, il existe $\delta_t > 0$ telque $\epsilon_t \leq \frac{1}{2} - \delta_t$. Alors, en notant $\delta = min_t(\delta_t)$, nous avons :

$$\forall t \in \{1, ..., T\}, \epsilon_t \leq \frac{1}{2} - \delta$$

Comme $E_{0-1}(H^{(T)}, \mathcal{S}) \leq E_{exp}(\sum_{t=1}^{T} \alpha_t h^{(t)}, \mathcal{S})$ est supérieure à l'erreur zéro-un, il suffit de prouver A.4 pour l'erreur exponentielle. Notons $g^{(T)} = \sum_{t=1}^{T} \alpha_t h^{(t)}$ et $Z_T = E_{exp}(g^{(T)}, \mathcal{S})$. On a alors :

$$Z_T = \frac{Z_T}{Z_0} = \frac{Z_T}{Z_{T-1}} . \frac{Z_{T-1}}{Z_{T-2}} ... \frac{Z_2}{Z_1} . \frac{Z_1}{Z_0}$$

Avec $Z_0 = 1$ l'erreur exponentielle pour la fonction nulle $g^{(0)}$ (par définition).

Il suffit donc de montrer que pour tout t, on a :

$$\frac{Z_{t+1}}{Z_t} \leq e^{-2\ \delta^2}$$

Comme à une itération t, $g^{t+1} = g^t + \alpha_{t+1}.h^{(t)}$, on a :

$$\begin{aligned}
\frac{Z_{t+1}}{Z_t} &= \frac{\sum_{i=1}^{n} exp(-y_i.g^{(t+1)}(\mathbf{x}_i))}{\sum_{i=1}^{n} exp(-y_i.g^{(t)}(\mathbf{x}_i))} \\
&= \frac{\sum_{i=1}^{n} exp(-y_i.g^{(t)}(\mathbf{x}_i)).exp(-\alpha_{t+1}\ y_i\ h^{(t+1)}(\mathbf{x}_i))}{\sum_{i=1}^{n} exp(-y_i.g^{(t)}(\mathbf{x}_i))} \\
&= \sum_{i=1}^{n} \mathcal{D}_i^{(t+1)} exp\big(-\alpha_{t+1}\ y_i\ h^{(t+1)}(\mathbf{x}_i)\big) \\
&= e^{\alpha_{t+1}}\epsilon_{t+1} + e^{-\alpha_{t+1}}(1-\epsilon_{t+1}) \\
&= \epsilon_{t+1}\sqrt{\frac{1}{\epsilon_{t+1}} - 1} + \frac{1-\epsilon_{t+1}}{\sqrt{\frac{1}{\epsilon_{t+1}} - 1}} \\
&= 2\sqrt{\epsilon_{t+1}(1-\epsilon_{t+1})}
\end{aligned}$$

Comme $\epsilon_{t+1} \leq \frac{1}{2} - \delta$ et que $a : x \mapsto x(1-x)$ est croissante sur $[0, \frac{1}{2}]$, nous avons :

$$\begin{aligned}
2\sqrt{\epsilon_{t+1}(1-\epsilon_{t+1})} &\leq 2\sqrt{a(\frac{1}{2} - \delta)} \\
&= \sqrt{\big(\frac{1}{2} - \delta\big)\big(\frac{1}{2} + \gamma\big)} \\
&= \sqrt{1 - 4\delta^2}
\end{aligned}$$

Comme $1 - x \leq e^{-x}$, on a :

$$\sqrt{1 - 4\delta^2} \leq \sqrt{e^{-4\delta^2}} = e^{-2\delta^2}$$

□

La courbe d'apprentissage d'Adaboost sur l'ensemble d'entraînement aura donc une allure exponentielle décroissante. La convergence vers une erreur nulle signifie que $E_{0-1}(H^{(T)}, \mathcal{S}) < \frac{1}{n}$, et donc $exp(-2\delta^2 T) < 1/n$ d'où $T > \frac{ln(n)}{2\delta^2}$. Ainsi, des plus petites valeurs de δ nécessitent de plus grandes valeurs de T pour annuler l'erreur sur l'ensemble d'entraînement. Cela invite à penser que des apprenants *ABase* performants donneront de meilleures courbes d'apprentissage. En pratique, l'inverse est vrai (article la force de la faible apprenabilité). En effet, choisir des votants de hautes performances a de grandes chances de donner une courbe d'apprentissage qui sature à une valeur finale non minimale à partir d'une itération T_1. Par contre, choisir des votants faibles et les amplifier pour $T \gg T_1$ donnera une meilleure courbe d'apprentissage qui sature à une valeur finale plus proche de 0.

Annexe B

Propriétés de l'algorithme XGBoost

B.1 Énoncé de l'algorithme

Adapté de (Chen and Guestrin, 2016)

Algorithme 2 : XGBoost

Données : $\mathcal{S} = \{(\mathbf{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}, i \in \{1, ..., n\}\}$; `/* l'ensemble d'entraînement */`

Données : $\mathcal{D} = \{d_1, d_2, ..., d_n\}$ tel que $\sum_{d \in \mathcal{D}} d = 1$; `/* poids des observations */`

Données : Un algorithme d'apprentissage des votants $ABase(\bullet, \bullet)$ minimisant l'erreur quadratique moyenne pondérée

Données : $T \in \mathbb{N} \setminus \{0\}$; `/* le nombre d'itérations */`

Données : Un choix de la fonction de perte

Initialisation : $g^{(0)} : \mathbf{x} \in \mathcal{X} \mapsto 0$

1 Calculer $\forall i \in \{1, ..., n\}$ $a_{i,1}$ et $b_{i,1}$;

Initialisation : $\mathcal{D}^{(1)} \leftarrow \emptyset$, $\mathcal{S}^{(1)} \leftarrow \emptyset$

2 **pour** *i = 1,...,n* **faire**

3 $\quad \mathcal{D}^{(1)}$.insert$(b_{i,1}/2)$;

4 $\quad \mathcal{S}^{(1)}$.insert$((\mathbf{x}_i, a_{i,1}/a_{i,1})$;

5 **fin**

6 **pour** $t = 1, ..., T$ **faire**

7 $\quad$ Apprendre un votant $f^{(t)} \leftarrow ABase(\mathcal{S}^{(t)}, \mathcal{D}^{(t)})$; `/*` $f^{(t)} : \mathcal{X} \longrightarrow \mathbb{R}$ `*/`

8 $\quad$ Mettre à jour les logits $\forall\ i \in \{1, ..., n\},\ g_{i,t} = g_{i,t-1} + f(\mathbf{x}_i)$;

9 $\quad$ Calculer les dérivées $a_{i,t+1}$ et $b_{i,t+1}$ $\forall i \in \{1, ..., n\}$;

10 $\quad$ Mettre à jour les pondérations $\mathcal{D}^{(t+1)}$ et $\mathcal{S}^{(1)}$; `/* Comme en ligne 2 */`

11 **fin**

Résultat : Une fonction $h : \mathcal{X} \longrightarrow [0, 1]$ définie par :

$$\forall \mathbf{x} \in \mathcal{X}, h(\mathbf{x}) = \sigma(g^{(T)}(\mathbf{x})) = \sigma\big(\sum_{t=1}^{T} f^{(t)}(\mathbf{x})\big)$$

B.2 Application pour différentes fonctions de perte

À chaque itération t, XGBoost ajoute un arbre de régression f minimisant la fonction objective suivante.

$$E = \sum_{i=1}^{n} b_{i,t}\Big[f(\mathbf{x}_i) - \frac{a_{i,t}}{b_{i,t}}\Big]^2$$

$$\text{avec } a_{i,t} = -\frac{\partial \ell(y_i, g_{i,t-1})}{\partial g_{i,t-1}},\ b_{i,t} = \frac{\partial^2 \ell(y_i, g_{i,t-1})}{\partial g_{i,t-1}^2}$$

et ℓ la fonction de perte choisie.

Dans cette section, on exprimera les poids $d_i^{(t)} = b_{i,t}$ et les pseudo étiquettes $y_i^{(t)} = -a_{i,t}/b_{i,t}$ pour les choix usuels de la fonction de perte, et on comparera qualitativement les fonctions objectives résultantes.

B.2.1 Perte exponentielle

La perte exponentielle pour un exemple $(\mathbf{x}_i, y_i)$ est définie à l'itération $(t-1)$ par :

$$\ell(y_i, \hat{y}_i^{(t-1)}) = e^{-y_i.g_{i,t-1}}$$

D'où, les dérivées $a_{i,t}$ et $b_{i,t}$:

$$a_{i,t} = -(-y_i).e^{-y_i.g_{i,t-1}} = y_i.e^{-y_i.g_{i,t-1}}$$
$$b_{i,t} = (-y_i)^2.e^{-y_i.g_{i,t-1}} = e^{-y_i.g_{i,t-1}}$$

Ainsi,

$$d_i^{(t)} = e^{-y_i.g_{i,t-1}} \text{ et } y_i^{(t)} = y_i$$

L'Algorithme générant le votant faible prend les étiquettes y_i de l'ensemble d'entraînement et des poids proportionnels aux poids crées par Adaboost. Le votant généré est appris pour effectuer des prédictions corrélés avec les étiquettes associées aux pondérations $d_i^{(t)}$ les plus importantes. Celles-ci correspondent aux erreurs de classification commises par l'ensemble crée à l'itération précédente (de manière similaire à Adaboost).

B.2.2 Perte logistique

La perte logistique pour un exemple $(\mathbf{x}_i, y_i)$ est définie à l'itération $(t-1)$ par :

$$\ell(y_i, \hat{y}_i^{(t-1)}) = ln(1 + e^{-y_i.g_{i,t-1}})$$

D'où, les dérivées $a_{i,t}$ et $b_{i,t}$:

$$a_{i,t} = -(-y_i).\sigma(-y_i.g_{i,t-1}) = y_i.\sigma(-y_i.g_{i,t-1})$$
$$b_{i,t} = (-y_i)^2.\sigma(-y_i.g_{i,t-1}).\sigma(+y_i.g_{i,t-1}) = \sigma(y_i.g_{i,t-1}).\sigma(-y_i.g_{i,t-1}) \simeq \frac{1}{4}exp(g_{i,t-1}^2)$$

Ainsi,

$$d_i^{(t)} \propto exp(g_{i,t-1}^2) \text{ et } y_i^{(t)} = \frac{1}{y_i \sigma(y_i.g_{i,t-1})} = y_i.(1 + e^{-y_i.g_{i,t-1}})$$

Le votant faible donne plus d'importance pour les exemples associés aux prédictions $g_{i,t-1}$ proches de 0. Pour ces mêmes exemples, le votant f produit fournira des prédictions corrélés avec le double des étiquettes originales ($y_i^{(t)} \simeq (1 + e^0)y_i = 2y_i$), ce qui est suffisant pour corriger les erreurs de prédictions à faible logit. Les termes de l'erreur associés aux logits $g_{i,t-1}$ d'amplitude importante sont associés à des poids presque nuls. Pour les observations à marge positive (bonnes prédictions), les nouvelles étiquettes sont presque égaux aux étiquettes de l'ensemble d'entraînement. Pour chaque observation à marge négative, le votant essaiera de produire une prédiction proportionnelle à l'étiquette originale, avec un coefficient de proportionnalité très important (en $1 + exp(|g_{i,t-1}|)$). Cela peut compenser la baisse de l'importance de ces observations engendrées par $d_i^{(t)} \simeq 0$.

En choisissant un faible taux d'apprentissage, on peut forcer les logit de varier proche de zéro. La pénalité induite par les pondérations calculées n'aura pas d'impact sur les observations à marge négative à l'itération précédente.

B.3 Lien entre XGBoost et Adaboost

Dans le cas particulier où f est proportionnelle à un classificateur binaire $\varphi : f(\mathbf{x}) = \alpha.\varphi(\mathbf{x})$, et dans le cas de la perte exponentielle, nous avons $a_{i,t} = y_i \ . \ exp(-y_i \ . \ g_{i,t-1})$ et $b_{i,t} = exp(-y_i \ . \ g_{i,t-1})$. Et donc, f minimise l'erreur quadratique moyenne définie par :

$$\begin{aligned} F &= \sum_{i=1}^{n} exp(-y_i \ . \ g_{i,t-1}) \Big[\alpha.\varphi(\mathbf{x}_i) - y_i\Big]^2 \\ &\propto \sum_{i=1}^{n} d_i^{(t)} \Big[\alpha.\varphi(\mathbf{x}_i) - y_i\Big]^2 \end{aligned}$$

Ceci est équivalent à dire que le passage de la fonction objective d'Adaboost vers celle d'XGBoost avec perte exponentielle revient à remplacer la perte exponentielle $exp(-\alpha \ . \ y_i \ . \ \varphi^{(t)}(\mathbf{x}_i))$ par la perte quadratique $[\alpha.\varphi(\mathbf{x}_i) - y_i]^2$ en gardant les mêmes pondérations. De plus, par dérivation, le poids α donné par XGBoost est défini par la marge moyenne suivante :

$$\alpha_{XGB} = \sum_{i=1}^{n} d_i^{(t)} \ . \ y_i \ . \ \varphi(\mathbf{x}_i)$$

Notons γ cette marge. L'expression du poids fourni par Adaboost peut être exprimé en fonction de γ

$$\alpha_{ADA} = \frac{1}{2} ln(\frac{1+\gamma}{1-\gamma})$$

On observe alors que le poids $\alpha_{XGB}(\gamma)$ varie sur la tangente à la courbe $\alpha_{ADA}(\gamma)$ à l'origine. La figure A.1 montre les deux variations. En particulier, on voit que pour des classificateurs performants (γ proche de 1), les poids fournis par Adaboost tendent vers l'infini. L'ensemble sera alors dominé par ces classificateurs. Toutefois, pour XGBoost, les classificateurs performants seront associés à des poids proches de 1.

Bibliographie

ACCAP. Canadian life and health insurance facts, 2018. URL http://clhia.uberflip.com/i/1030763-canadian-life-and-health-insurance-facts-2018/0? (Page consultée le 3 février 2020).

Assurland. Résiliation d'assurance :où en sont les français ?, 2019. URL https://www.assurland.com/presse/communiques-de-presse-assurance/resiliation-d-assurance-ou-en-sont-les-francais.html.

R Barandela, J.S Sanchez, V Garcia, and E Rangel. Strategies for learning in class imbalance problems. *Pattern Recognition*, 36(3) :849–851, 2003. ISSN 0031-3203. doi : https://doi.org/10.1016/S0031-3203(02)00257-1. URL https://www.sciencedirect.com/science/article/pii/S0031320302002571.

Gustavo E. A. P. A. Batista, Ronaldo C. Prati, and Maria Carolina Monard. A study of the behavior of several methods for balancing machine learning training data. 6(1) :20–29, jun 2004. ISSN 1931-0145. doi : 10.1145/1007730.1007735. URL https://doi.org/10.1145/1007730.1007735.

Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote : synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16 : 321–357, 2002.

Nitesh V. Chawla, Aleksandar Lazarevic, Lawrence O. Hall, and Kevin W. Bowyer. Smoteboost : Improving prediction of the minority class in boosting. In Nada Lavrač, Dragan Gamberger, Ljupčo Todorovski, and Hendrik Blockeel, editors, *Knowledge Discovery in Databases : PKDD 2003*, pages 107–119, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-39804-2.

Nitesh V. Chawla, Nathalie Japkowicz, and Aleksander Kotcz. Editorial : Special issue on learning from imbalanced data sets. *SIGKDD Explor. Newsl.*, 6(1) :1–6, jun 2004. ISSN 1931-0145. doi : 10.1145/1007730.1007733. URL https://doi.org/10.1145/1007730.1007733.

Tianqi Chen and Carlos Guestrin. Xgboost : A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

Isabelle Guyon, Jason Weston, Stephen D. Barnhill, and Vladimir Naumovich Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46 : 389–422, 2004.

Haibo He and E.A. Garcia. Learning from imbalanced data. *Knowledge and Data Engineering, IEEE Transactions on*, 21(9) :1263–1284, Sept 2009. ISSN 1041-4347. doi : 10.1109/TKDE.2008.239. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5128907&tag=1.

Haibo He, Yang Bai, Edwardo Garcia, and Shutao Li. Adasyn : Adaptive synthetic sampling approach for imbalanced learning. *Proceedings of the International Joint Conference on Neural Networks*, pages 1322 – 1328, 07 2008. doi : 10.1109/IJCNN.2008.4633969.

Duan Huajuan, Yongqing Wei, Peiyu Liu, and Hongxia Yin. A novel ensemble framework based on k-means and resampling for imbalanced data. *Applied Sciences*, 10 :1684, 03 2020. doi : 10.3390/app10051684.

Taeho Jo and Nathalie Japkowicz. Class imbalances versus small disjuncts. *SIGKDD Explor. Newsl.*, 6(1) :40–49, jun 2004. ISSN 1931-0145. doi : 10.1145/1007730.1007737. URL https://doi.org/10.1145/1007730.1007737.

Balázs Kégl. The return of adaboost. mh : multi-class hamming trees. *arXiv preprint arXiv :1312.6086*, 2013.

Ryan Lichtenwalter and Nitesh Chawla. Adaptive methods for classification in arbitrarily imbalanced and drifting data streams. pages 53–75, 04 2009. ISBN 978-3-642-14639-8. doi : 10.1007/978-3-642-14640-4_5.

Xu-Ying Liu, Jianxin Wu, and Zhi-Hua Zhou. Exploratory undersampling for class-imbalance learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39 (2) :539–550, 2009. doi : 10.1109/TSMCB.2008.2007853.

Ronaldo C. Prati, Gustavo E. A. P. A. Batista, and Maria Carolina Monard. Class imbalances versus class overlapping : An analysis of a learning system behavior. 2004.

S.J. Raudys and A.K. Jain. Small sample size effects in statistical pattern recognition : recommendations for practitioners. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(3) :252–264, 1991. doi : 10.1109/34.75512.

Robert E Schapire. The boosting approach to machine learning : An overview. *Nonlinear estimation and classification*, pages 149–171, 2003.

Chris Seiffert, Taghi M. Khoshgoftaar, and Jason Van Hulse. Hybrid sampling for imbalanced data. pages 202–207, 2008. doi : 10.1109/IRI.2008.4583030.

Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning : From theory to algorithms*. Cambridge university press, 2014.

Petr Somol, Jana Novovicova, and Pavel Pudil. *Efficient Feature Subset Selection and Subset Size Optimization*, volume 56. 02 2010. ISBN 978-953-7619-90-9. doi : 10.5772/9356.

Dan Steinberg. Cart : classification and regression trees. In *The top ten algorithms in data mining*, pages 193–216. Chapman and Hall/CRC, 2009.

Yanmin Sun, Andrew KC Wong, and Mohamed S Kamel. Classification of imbalanced data : A review. *International journal of pattern recognition and artificial intelligence*, 23(04) : 687–719, 2009.

Lynn Thomas. Customer loyalty and retention primer, 2020. URL https://www.iiadallas.org/page/75. (Page consultée le 3 février 2020).

Ivan Tomek. Two modifications of cnn. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-6(11) :769–772, 1976. doi : 10.1109/TSMC.1976.4309452.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9 :2579–2605, 2008. URL http://www.jmlr.org/papers/v9/vandermaaten08a.html.

Benjamin X Wang and Nathalie Japkowicz. Imbalanced data set learning with synthetic samples. 19 :435, 2004.

Gary M. Weiss. Mining with rarity : A unifying framework. 6(1) :7–19, jun 2004. ISSN 1931-0145. doi : 10.1145/1007730.1007734. URL https://doi.org/10.1145/1007730.1007734.

Qiang Yang and Xindong Wu. 10 challenging problems in data mining research. *International Journal of Information Technology & Decision Making*, 05(05) :597–604, 2006.