Projet "Ensemble Methods in ML"

ATTALI, Hugo
hugo.attali@univ-lyon2.fr

DJAALEB, Tom
tom.djaaleb@univ-lyon2.fr

18 octobre 2022

Table des matières

1 Introduction

Dans le cadre de la deuxieme année du Master MALIA (Machine Learning pour L'Intelligence Artificielle) nous réalisons un projet pour l'enseignement d'*Ensemble Learning*. L'objecif est d'implémenter un certain nombre de méthodes pour une tâche d'apprentissage supervisé en mettant en perspective l'efficacité des méthodes ensemblistes par rapport à des méthodes plus classiques. Nous étudierons pour finir des méthodes non supervisées. Dans ce projet nous appliquerons nos méthodes pour résoudre une tâche de classification binaire.

2 Données

Nos données traitent de prévisions de défauts de paiment ou non par rapport à des variables socio-économiques. Le jeu de données sur lequel nous appliquerons nos méthodes est disponible ici, il s'agit d'un échantillon d'informations sur des clients possèdant une carte bancaire, résident de Taïwan. Les informations ont été observées entre avril et novembre 2005. Notre jeu de données est composé de 30 000 individus et 24 variables (incluant la variable à prédire). La variable à prédire est binaire (défaut de paiment ou non), détaillons maintenant les variables explicatives :

- X_1 : Montant du crédit accordé (dollars) : il comprend à la fois le crédit à la consommation individuel et le crédit familial.
- X_2 : Sexe (1 : H, 2 : F).
- X_3 : Niveau d'études (1 : études supérieures, 2 : université, 3 : lycée, 4 : autres).
- X_4 : État civil (1 : marié, 2 : célibataire, 3 : autres).
- X_5 : Âge (année).
- X_6 - X_{11} : Historique des paiements antérieurs (d'avril à septembre 2005) : X_6 = le statut de remboursement en septembre 2005, X_7 : le statut de remboursement en août 2005, ..., X_{11} : le statut de remboursement en avril 2005. Si la modalité est -1, le delai du paiement est respecté, 1 : retard de paiement d'un mois, 2 : retard de paiement de deux mois..., 9 : retard de paiement de neuf mois et plus.
- X_{12}-X_{17} : Montant de la facture (en dollars). X_{12} : montant de la facture de septembre 2005, X_{13} : montant de la facture d'août 2005, ..., X_{17} : montant de la facture d'avril 2005.
- X_{18}-X_{23} : Montant du paiement précédent (dollar). X_{18} : montant payé en septembre 2005, X_{19} : montant payé en août 2005, ..., X_{23} : montant payé en avril 2005.

Au vue de la description on peut s'attendre à travailler dans un contexte déséquilibré, autrement dit une proportion de personnes qui ne peuvent pas payer bien plus faible que l'inverse. On regarde ainsi la répartition de notre variable à prédire disponible en Table 1. La fréquence de non paiement est de 22 %.

Défaut de paiement		
Non	23364	77.88%
Oui	6636	22.12%

TABLE 1 – Description de la variable à prédire.

Un point qui est important a notifié est l'objectif des experts métiers : nous supposons que les services bancaires accordent une importance notable à détecter les futurs défauts de paiement, quitte à en surévaluer le nombre, plutôt que dans louper. Cette objectif étant défini, nous verrons comment le traduire en métrique scientifique dans nos tests par la suite.

3 Apprentissage supervisé

Cette partie revient sur les méthodes d'apprentissage supervisé vues en cours d'*Ensemble Learning* et cherche à déterminer quel modèle sera le plus performant sachant nos données. Pour chaque méthode, nous présentons rapidement le principe ainsi que sa construction puis nous déterminons les hyperparamètres optimaux. La dernière sous partie fait une comparaison globale de toutes les méthodes et

conclut sur la méthode la plus efficace. Pour l'implémentation du code, nous utilisons Python ainsi que la libraire Scikit-Learn, il existe pour certaines méthodes des algorithmes plus performant en temps, pouvant être exécutés sur GPU, cependant l'accès à un GPU étant généralement difficile, nous nous restreignons à une implémentation sur CPU. La mértique principale pour comparer nos modèles est l'*accuracy*, cependant pour les comparaisons globales nous étudierons également le *recall* et la *precision*. Le dernier outil pour comparer les méthodes sera le temps d'implémentation nécessaire pour créer un modèle, nos données n'étant qu'un faible échantillon par rapport aux données que les banques peuvent recueillir, nous devons également penser à ce facteur. Aussi, pour que nos résultats soient reproductibles, nous avons décidé arbitrairement d'un *random state* (14).

3.1 Méthodes simples

3.1.1 Régression logistique

La régression logistique est une méthode simple utilisée pour la classification. Cette méthode cherche à modéliser la probabilité conditionnelle de chaque classe sachant un vecteur $X = (X_1, ...X_p)$. Cette probabilité est définie telle que

$$p(Y = 1|X) = \frac{e^{\beta_0 + \sum_{i=1}^{p} \beta_i X_i}}{1 + e^{\beta_0 + \sum_{i=1}^{p} \beta_i X_i}} \quad \beta_0, ..., \beta_p \in \mathbb{R} \tag{1}$$

pour un problème de classification binaire où Y peut prendre deux modalités, 0 ou 1. Pour estimer les β optimaux, la méthode du maximum de vraissemblance est utilisée.
Il existe des variantes dites pénalisées de la régression logistique appliquant les méthodes de régularisations LASSO, Ridge ou une combinaison de celles-ci. Les β optimaux peuvent donc être obtenus en résolvant le problème d'optimisation suivant :

$$\max_{\beta} \left\{ \sum_{j=1}^{n} \left[y_j \left(\beta_0 + \sum_{i=1}^{p} \beta_i x_{ij} \right) - \log(1 + e^{\beta_0 + \sum_{i=1}^{p} \beta_i x_{ij}}) \right] - \lambda_1 \sum_{i=1}^{p} |\beta_i| - \lambda_2 \sum_{i=1}^{p} \beta_j^2 \right\} \tag{2}$$

où λ_1 et λ_2 sont des réels positifs. Lorsque λ_1 est nul, la méthode Ridge est appliquée et lorsque λ_2 est nul, la méthode LASSO est appliquée.

En premier lieu nous testons une régression logistique simple sans pénalisation, nous obtenons une *accuracy* de **0.778** après une CV-5. Ensuite nous testons la régression logistique avec les méthodes de régularisations Ridge, LASSO et Elasticnet pour différents coefficients, toujours en CV-5. Il apparaît que pénaliser notre modèle n'entraîne ni une baisse ni une hausse de l'*accuracy*. Les résultats du GridSearchCV5 pour les paramètres de régulatisation apparaissent ci-dessous en Table 2.

Méthode	Amplitude	Coefficient ℓ_1	*Accuracy*
RIDGE	0.1		0.778
	0.01		0.778
LASSO	0.1		0.778
	0.01		0.778
ELASTIC NET	0.1	0.1	0.778
	0.01		0.778
	0.1	0.5	0.778
	0.01		0.778
	0.1	0.9	0.778
	0.01		0.778

TABLE 2 – Résultats pour la régression logistique pénalisée.

3.1.2 k plus proches voisins

La méthode des k plus proches voisins pour la classification est une méthode non paramétrique, elle ne nécessite pas un modèle pour être entraînée. Cette méthode projette les données d'entraînement

dans l'espace et pour une nouvelle observation x donnée, elle calcule les distances entre ce x et les x_i d'entraînements. Les k x_i les plus proches de l'observation sont sélectionnés et l'on regarde la valeur y_i associée à chaque point. La réponse $\hat{y}$ pour l'observation x sera la modalité la plus représentée parmi les y_i (vote majoritaire). Il existe une variante utilisant un vote pondéré parmi les k plus proche voisins, un x_i très proche de x aura plus de poids dans le vote qu'un x_i éloigné (mais parmi les k plus proches). Cette méthode se basant sur un calcul de distance euclidienne, il est nécessaire de normaliser les données en amont car celles-ci peuvent être dans différentes unités selon les axes. De plus, il est difficile d'imaginer des distances pour des variables qualitatives, nous devrons donc faire abstraction de celles-ci pour cette méthode.

Dans notre implémentation, nous testons la méthode avec vote majoritaire et la méthode avec vote pondéré. De plus nous choisissons 5 valeurs de k arbitraires et nous réalisons un GridSearch avec CV5 sur ces deux paramètres. Il en ressort que les hyperparamètres qui maximisent l'*accuracy* sont $k = 10$ et vote majoritaire, et nous obtenons une accuracy de **0.782** en CV5. Les détails des tests sur les hyperparamètres sont exposés en Table 3.

Vote	k	*Accuracy*
Majoritaire	3	0.744
	5	0.765
	10	0.782
	15	0.779
	20	0.781
Pondéré	3	0.742
	5	0.763
	10	0.774
	15	0.779
	20	0.781

TABLE 3 – Résultats pour les kNN.

3.1.3 Arbre de décision

Un arbre de décision est une structure hiérarchique pouvant être représentée par un graphe. La racine contient l'ensemble $\mathbb{X}$ des observations. Cette racine est ensuite divisée en deux sous ensemble. La division optimale est calculée à partir d'une seule variable (cas univarié). Nous répétons ainsi l'opération plusieurs fois, à chaque fois sur les nouveaux sous ensembles créés, afin d'obtenir notre modèle. Les sous ensembles finaux sont appelés feuilles et forment une partition de l'ensemble $\mathbb{X}$. Dans notre cas, nous utiliserons un arbre de classification, le critère de séparation d'un noeud est basé sur l'entropie

$$\text{Ent}(p) = -p \log_2(p) - (1-p) \log_2(1-p) \tag{3}$$

ou p est la probabilité conditionnelle sachant X d'appartenir à la classe 1. Cette formule de l'entropie n'est valable que lorsque le problème de classification est binaire, ce qui est le cas dans notre étude. La meilleure séparation est donc celle qui minimise l'entropie. Pour éviter les problèmes de surentraînement, il est nécessaire de contrôler la profondeur de l'arbre, un arbre pûr (ou chaque élément de $\mathbb{X}$ est une feuille) aura une erreur en classification égale à 0 sur les données d'apprentissage mais une variance trop élevée.

Le seul paramètre qui sera testé pour construire notre arbre de décision sera la taille minimale de l'échantillon lors d'un *split*. Nous remarquons dans la Table 4 que plus cette valeur est élevée, plus l'*accuracy* augmente. En effet en augmentant la taille de ce paramètre, nous réduisons la variance engendrée, qui apparement décroît plus rapidement que le biais croît pour nos paramètres choisis. Notre meilleur arbre est donc celui qui nécessite au minimum 1000 observations pour qu'un nouvel échantillon soit séparé, et cet arbre à une *accuracy* de **0.812**.

Min Split	*Accuracy*
2	0.732
10	0.733
100	0.791
500	0.808
1000	0.812

TABLE 4 – Résultats pour les arbres de décisions.

3.2 Méthodes d'aggrégation

Les méthodes d'aggrégation consistent à entraîner une multitude de modèle divers et de choisir la prédiction la plus représentée parmi les sorties de ces modèles. Cette méthode suivant le principe du vote, celui-ci peut donc être majoritaire ou pondéré. Cette méthode ne nécessite pas d'avoir des modèles très performants en amont mais leur taux d'erreur en classification doit toujours être inférieur à 0,5. De plus les modèles sans biais sont favorisés car cela permet d'assumer l'hypothèse d'indépendance des votants requise par le Théorème du jury de Condorcet. Ce théorème montre également qu'un grand nombre de modèles améliore théoriquement les performances des méthodes d'aggrégation.

Le seul paramètre propre à la méthode d'aggrégation est de choisir le type de vote, majoritaire ou pondéré. Cependant, nous pouvons également agir sur les paramètres des *weak learner*. Dans un premier temps nous regardons l'*accuracy* pour le vote majoritaire et le vote pondéré avec des modèles non biaisés, ceux-ci étant généralement préférés pour l'aggrégation. Nous ferons un troisième essai en sélectionnant la méthode de vote la plus performante et en utilisant cette fois des estimateurs biaisés (10−NN et DT(Min Split=1000)), qui produisent de meilleures performances seuls. Les *weak learners* choisis sont la régréssion logistique non pénalisée, la méthode des kNN et un arbre de décision.

Vote	Biais	*Accuracy*
Majoritaire	Non	0.783
Pondéré	Non	0.773
Majoritaire	Oui	0.788

TABLE 5 – Résultats pour les méthodes d'aggrégation.

En observant la Table 5, nous remarquons que la troisième méthode est légérement meilleurs avec une *accuracy* de **0.788**, ce qui n'est pas le résultat le plus évident à prédire. Cependant ce résultat est logique car selon le Théorème de Condorcet, l'aggrégation nécessite un grand nombre de votants pour être performante, ce qui n'est pas le cas ici.

3.3 Stacking

Le stacking est une méthode d'ensemble largement utilisées dans l'apprentissage automatique. L'idée est d'empiler et de former plusieurs modèles, généralement avec différents types d'algorithmes, sur les données d'entrainement, puis plutôt que de choisir le meilleur modèle, tous les modèles sont agrégés à l'aide d'un autre modèle pour faire la prédiction finale. Les entrées pour le derniere modèle sont les sorties de prédiction des apprenants de la couche précédente.

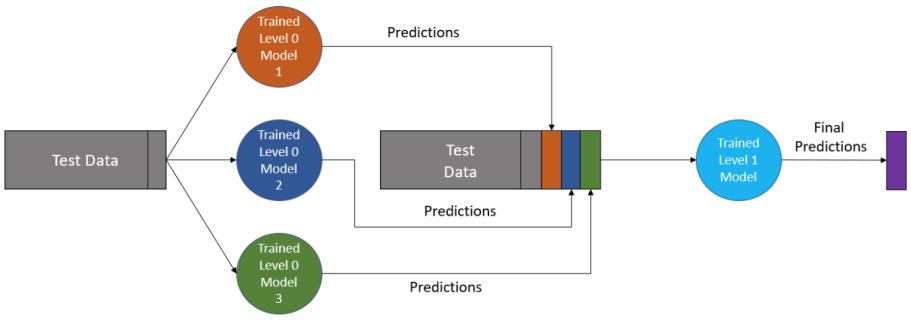

FIGURE 1 – Architecture Stacking.

Pour nos tests, nous reprenons les mêmes *weak learners* que pour l'aggrégation, cependant, nous ne traiterons ici que le cas des estimateurs non biaisés. Pour le prédicteur final, nous avons décidé arbitrairement de choisir les Support Vector Machines [4]. L'unique paramètre que nous avons testé a été la méthode du noyau choisie parmi trois noyaux : linéaire, polynomial et gaussien (*radial basis function*). La SVM avec noyau gaussien s'est avérée être la plus performante avec une *accuracy* de **0.785** comme l'illustre la Table 6.

Noyau	*Accuracy*
Linéaire	0.778
Polynomial	0.784
Gaussien	0.785

TABLE 6 – Résultats pour le Stacking.

3.4 Random Forest

Les Random Forest cherchent à réduire la variance du Bagging [2] en réduisant la corrélation entre les arbres de décisions. Pour ce faire, avant la division de chaque noeud, on choisit aléatoirement un nombre d'attributs inférieur ou égal au nombre de variables de l'ensemble d'entraînement pour mesurer l'entropie. Les Random Forest appliquent le principe du *random subspace*. Tout comme un arbre de décision simple, nous devons contrôler la profondeur de chaque arbre. S'ajoute à cela d'autres hyperparamètres à estimer : r le nombre de variables aléatoirement choisies à chaque noeud, et t le nombre d'échantillons *bootstrap*, généralement égal au nombre d'arbres de décisions de la forêt. Un avantage fort des Random Forest est l'ensemble *Out-Of-Bag*, qui peut être utilisé pour calculer l'erreur en généralisation, et évite donc de passer par la validation croisée.

Comme conseillé dans le papier de Cutler [5], nous utilisons l'erreur *Out-of-Bag* pour tester les hyperparamètres. Nous en testons trois, le nombre de variables aléatoirement sélectionnées lors d'un *split*, le nombre d'arbres dans la forêt et la profondeur des arbres, déterminé par la taille minimale de l'échantillon lors d'un *split*. Les modalités testées pour chaque hyperparamètre sont décrites dans la Table 7.

Nombre de variables	1	$\sqrt{p} \approx 5$	$p/3 \approx 8$	15	23
Nombre d'arbres	10	100	200	500	1000
Min Split	2	10	100	500	1000

TABLE 7 – Modalités testées pour les hyperparamètres des Random Forest.

Pour choisir nos paramètres optimaux, nous avons effectué un GridSearch, et nous avons donc entraîné 125 modèles. La méthode Random Forest la plus performante est celle utilisant toutes les variables (23), avec 1000 arbres et avec une taille minimale de l'echantillon lors d'un split égale à 2.

Ce modèle renvoie une *accuracy* de 0.818 sur l'erreur *Out-of-Bag*. Les Figure 4 5 6 (A.1) décrivent l'évolution de l'erreur *Out-of-Bag* en fonction de la variation d'un hyperparamètre.

Les Random Forest étant le seul modèle à pouvoir évaluer sa performance grâce à l'erreur *Out-of-Bag*, nous avons réentraîné le modèle avec les paramètres optimaux. Nous avons ensuite réalisé une CV5 dans le but d'obtenir une *accuracy* en validation croisée pour pouvoir comparer ce modèle aux autres. Il en ressort que notre modèle optimal de Random Forest obtient une *accuracy* de **0.816**.

3.5 AdaBoost

AdaBoost est un algorithme itératif et à chaque itération, une nouvelle fonction de prédiction est estimée de façon à palier les erreurs de la fonction précédente. Cette méthode requiert un *weak learner* en entrée, c'est à dire un prédicteur faiblement corrélé à Y, et construit un *strong learner*, i.e. un modèle performant. Cette méthode ressemble étroitement aux Random Forest, mais elle diffère dans la distribution de la probabilités de tirer les données bootstrap : la probabilité de tirer un élément qui a été mal classé à l'itération précédente augmente, et inversement pour les éléments correctement classés. On dit que les prédicteurs d'AdaBoost ne sont pas indépendants mais complémentaires. La prédiction finale par le modèle est un vote pondéré des *weak learners*. Il existe également une version réelle d'AdaBoost qui cherche à minimiser une fonction de perte exponentielle, mais nous ne l'utiliserons pas dans nos essais.

La détermination des hyperparamètres c'est orchestrée en deux parties : un premier GridSearchCV5 déterminant la valeur pour le *learning rate* et la profondeur des arbres, chaque modèle étant entraîné avec 50 itérations. Un second GridSearchCV5 a ensuite été implémenté pour déterminer combien d'itérations étaient nécessaire pour avoir le modèle le plus performant. Pour déterminer la profondeur de l'arbre nous avons utilisé un paramètre différent par rapport aux méthodes précédentes, qui est la profondeur maximum que celui-ci peut avoir, pour alléger les calculs de la machine. La Table 8 regroupent les résultats de la première expérimentation. Nous remarquons qu'un *learning rate* faible semble être plus performant, de plus la profondeur maximum des arbres qui donne les meilleurs résultats est égale à 5. L'évolution de l'*accuracy* en fonction de ses deux paramètres peut être visionnée en Figure 7 et 8 (A.2).

Profondeur Max	*Learning rate*	*Accuracy*
1	0.1	0.803
	1	0.805
	10	0.774
5	0.1	0.815
	1	0.809
	10	0.659
10	0.1	0.814
	1	0.789
	10	0.548

TABLE 8 – Premiers résultats pour AdaBoost.

La seconde phase de tests cherche à déterminer le nombre d'itérations optimal pour notre modèle. En observant la Table 9, nous concluons que 100 itérations suffisent pour obtenir le meilleur modèle, de plus ajouter des arbres dégrade progressivement notre modèle après que ce cap soit passé. Notre meilleur modèle AdaBoost obtient donc une *accuracy* de **0.816**. L'évolution de l'*accuracy* en fonction du nombre d'itérations est décrite sur la Figure 9 (A.2).

Nombre d'itérations	*Accuracy*
5	0.813
10	0.815
20	0.815
50	0.815
100	0.816
500	0.815
1000	0.814

TABLE 9 – Seconds résultats pour AdaBoost.

3.6 Gradient Boosting

Le Gradient Boosting est la généralisation de la méthode AdaBoost réelle évoquée dans la section précédente. Plutôt que d'utiliser une fonction de perte exponentielle, le Gradient Boosting laisse le choix à l'utilisateur sur la fonction de perte, seulement si celle-ci est différentiable. Plus particulièrement, dans le cas de la classification binaire, il est recommandé d'utiliser la fonction de perte appelée *negative binomial log likelihood* et qui est définie par

$$\ell_{nbll}(f, X, y) = \log(1 + e^{-2yf(X)}) \tag{4}$$

où f est la fonction de prédiction, X un vecteur colonne et y la réponse associée. C'est cette fonction de perte que nous utiliserons pour entraîner notre modèle.

Les paramètres testés de notre modèle ainsi que les modalités prises sont disponible en Table 10. Comme pour les autres méthodes nous appliquons un GridSearchCV5 pour déterminer les hyperparamètres qui maximisent l'*accuracy*. Le meilleur modèle s'avère être celui avec des arbres ayant une profondeur maximale de 5, choisissant 5 variables pour *split* un noeud, et qui effectue le tout sur 100 itérations. Ce modèle à une *accuracy* de **0.814**.

Nombre de variables	1	$\sqrt{p} \approx 5$	$p/3 \approx 8$	15	23
Nombre d'itérations	100	500	1000		
Profondeurs Max des arbres	1	5	10		

TABLE 10 – Modalités testées pour les hyperparamètres du Gradient Boosting.

L'évolution de l'*accuracy* en fonction des hyperparamètres est retranscrite sur les Figure 10 11 12 (A.3).

3.7 Comparaison des méthodes

Comme nous travaillons dans un contexte déséquilibré nous ne pourvons pas prendre en compte uniquement l'accuracy pour comparer nos méthodes. Nous évaluons les modèles en fonction de leur performance en *accuracy* mais aussi en *precision* (fréquence des modalités "1" réelles parmi les modalités "1" prédites) et en *recall* (fréquence des modalités "1" réelles prédites correctement parmi les modalités "1" réelles). Cette troisième métrique est très importante car l'objectif principal des banques et d'anticiper les défauts de paiement qui vont arriver. La Table 11 ainsi que la Figure 2 illustrent ces comparaisons.

Modèle	*Accuracy*	*Precision*	*Recall*
Régression Logistique	0.778	0.499	0.024
kNN	0.781	0.535	0.103
Arbre de décision	0.812	0.638	0.351
Aggrégation	0.788	**0.709**	0.073
Stacking	0.785	0.620	0.074
Random Forest	**0.817**	0.653	**0.377**
AdaBoost	0.815	0.651	0.358
GradientBoosting	0.814	0.641	0.367

TABLE 11 – Comparaisons des modèles en performance.

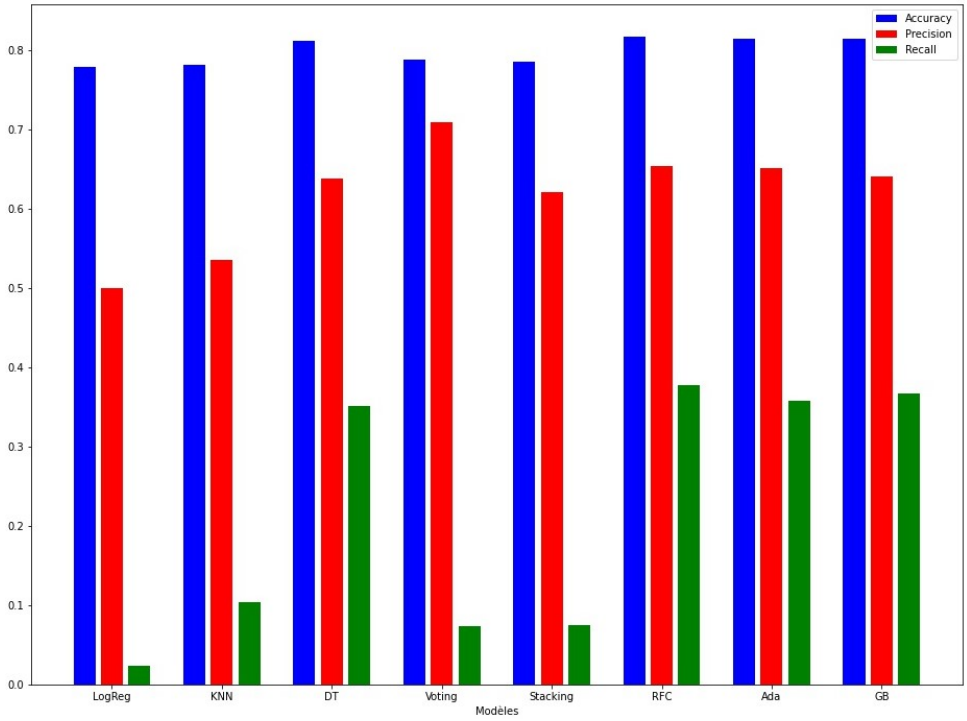

FIGURE 2 – Comparaison des modèles en performance.

Il en ressort que les Random Forest ont la meilleure *accuracy* ainsi que le meilleur *recall*, là où notre méthode d'aggrégation obtient la *precision* la plus élévé. Si nous devions recommander une méthode en se basant uniquement sur les résultats, nous préconiserions les Random Forest. En effet la robustesse de cette méthode permet de mieux traiter les données déséquilibrés par rapport aux autres méthodes.

Un facteur qui est important à prendre en compte dans la comparaison des modèle est le temps d'entraînement. En effet avec la démocratisation du deep Learning de plus en plus de méthodes sont dévélopées avec des millions de paramatres rendant leur adaptabilité limité pour un cas pratique. Néanmoins nous travaillons ici sur un échantillon et nous nous doutons que les banques possèdent un grand nombre de données (plusieurs millions d'observations sur des dizaines d'années). En regardant la Table 12, nous remarquons que le Gradient Boosting s'exécute plus rapidement que les Random Forest en ayant des performances proches. Ceci s'explique au nombre plus faible d'itérations requises par le Gradient Boosting pour approcher ces performances. Cette méthode peut être une alternative pour gagner du temps de computation. Notre meilleur arbre de décision a également des performances correctes pour un temps d'exécution très court, cette méthode peut également être intéressante pour les jeux de données très volumineux. A noter que le temps pour les kNN est faussé car cette méthode à moins de variables que les autres. Enfin, la méthode du Stacking est très longue a implémentée, ceci est dû au choix des SVM avec kernel gaussien qui sont lourdes en mémoire et en temps d'exécution.

Modèle	Temps d'entraînement (s)
Régression Logistique	45
kNN	0.1
Arbre de décision	1.1
Aggrégation	44
Stacking	181
Random Forest	14
AdaBoost	29
Gradient Boosting	8

TABLE 12 – Temps d'entaînement pour chaque modèle.

4 Apprentissage non supervisé

Pour la suite de ce travail, nous allons étudier des techniques d'apprentissage non supervisé sur notre jeu de données, afin de voir si différentes méthodes sont capables de détecter les individus en défaut de paiement ou non et de les classer dans un même groupe (i.e. *cluster*). Pour de pas surchager le rapport nous n'allons seulement présenter que quelques résultats. Tous les résultas des autres variantes sont entièrement reproductibles avec notre code. Nous appliquerons nos algorithmes sur l'ensemble des données mais pour visualiser les potentiels *clusters*, nous réalisons en amont une réduction de la dimension pour visualiser les données en deux dimensions. Nous utilisons deux méthodes pour réaliser cette réduction de dimension, ACP et UMAP[6].

UMAP L'auteur propose de reconstruire l'espace topologique de grande dimension grâce à un outil géométrique appelé complexes simpliciaux très utilisé dans le domaine de l'étude des espaces topologiques. Celui-ci définit une métrique locale dépendante de l'espace topologique construit à l'aide des complexes simpliciaux. Le choix du nombres de voisins dépend donc de l'interprétation topologique de l'espace capturé. Autrement dit le choix d'un petit nombre de voisins signifie que nous voulons une représentation qui capture une structure très locale tandis qu'un choix du nombres de voisins plus grand signifie que nos estimations seront basées sur des régions plus grandes, et donc représenteront moins bien une structure locale mais mieux une structure globale.

La projection de nos données avec la méthode UMAP est visualisable sur la Figure 13 (B.1) tandis que la projection avec l'ACP se trouve en Figure 14 (B.2). On observe que les individus en défaut de paiement (en blanc) ne sont pas facilement différentiables sur nos visualisations. Pour les méthodes nécessitant de fixer le nombre de *cluster* en amont, nous utilisons un nombre de *cluster* égal à 2 pour comparer facilement nos résultats avec les véritables labels. Comme nous travaillons dans un contexte déséquilibré, il ne faut pas s'intérésser uniquement à l'*accuracy* de nos méthodes car si l'on classifie tous les individus dans la classe "pas de défaut de paiement" nous aurons près de 78% d'*accuracy* mais notre méthode n'apportera finalement aucune information intéréssante.

4.1 Kmeans

Une premiere méthode que nous avons expérimenté est l'algorithme kmeans. C'est l'un des algorithmes de clustering les plus connus. L'algorithme est basé sur la similarité entre les données grâce à la notion de distance entre individus. On associe alors à chaque donnée son centroïde le plus proche, afin de créer des *clusters*. Les moyennes des distances des données d'un *cluster* définissent la position de leur centroïde. Le fait d'utiliser la moyenne pose un certain nombre de problèmes car l'algorithme est très sensible aux *outliers*. La représentation des centroïdes par la méthode des kmeans est visualisable sur la Figure 15 (C.1). Intéressons nous maintenant à la prediction de nos *clusters* via notre kmeans en observant la matrice de confusion en Table 13 ainsi que les différentes métriques calculées en Table 14.

	0	**1**
Cluster 0	11956	11408
Cluster 1	2213	4423

TABLE 13 – Comparaison des *clusters* avec un kmeans++ et les prédictions réelles.

	Precision	*Recall*	*Accuracy*
0	0.82	0.51	
1	0.26	0.61	
			0.53

TABLE 14 – Analyse des *clusters* avec l'approche kmeans++.

On remarque que le kmeans à une *accuracy* de 0.53 sur nos données. Un résultat moyen et plutôt attendu au vu des représentations en petite dimension des données. Le déséquilibrage des données accentue la difficulté de la classification non supervisée. On obtient une bonne précision pour la classe majoritaire mais les résultats des autres indicateurs sont limités. A titre de comparaison un Kmeans avec une initialisation random obtient les résultats présentés en Table 15.

	Precision	*Recall*	*Accuracy*
0	0.74	0.49	
1	0.18	0.39	
			0.47

TABLE 15 – Analyse des *clusters* avec l'approche kmeans en initialisation *random*.

Ceux-ci sont moins bons. Ce résultat est attendu car la méthode des kmeans est très sensible à l'initialisation.

4.2 Classification hiérarchique ascendante

La classification ascendante hiérarchique (CAH) est une méthode de clustering itérative basée sur la notion de distance. On peut décrire la méthode en trois points :

- On commence par calculer la matrice de distance de nos individus.
- On assemble les deux individus où la distance est la plus faible créant ainsi un *cluster* comprenant ces deux objets.
- On calcule ensuite la distance entre cette classe et les autres individus selon la distance. On regroupe enfin les individus ou classes dont le regroupement minimise une distance. On continue ainsi jusqu'à ce que tous les individus soient dans une classe.

Ces regroupements produisent un arbre que l'on appelle dendrogramme dont la racine correspond à la classe regroupant l'ensemble des individus. On peut alors couper l'arbre à un certain niveau selon le nombre de *cluster* choisis. Nous réalisons plusieurs variantes de la CAH. Nous présentons les predictions de l'approche CAH avec un *linkage ward* et la distance euclidienne et un *linkage complete* et la distance de Manhattan.

	0	**1**
Cluster 0	15007	8357
Cluster 1	4665	1971

TABLE 16 – Comparaison *clusters* avec l'approche CAH (*linkage ward*) et les prédictions réelles.

	Precision	*Recall*	*Accuracy*
0	0.76	0.64	
1	0.19	0.30	
			0.57

TABLE 17 – Analyse des *clusters* avec l'approche CAH (*linkage ward*).

Bien que les résultats soient légérements meilleurs comme décrit dans les Table 16 et 17, on remarque que cette méthode a toujours du mal à distinguer les individus en défaut de paiement. On varie la méthode et la distance, on utilise un *linkage complete* et la distance de Manhattan. Nous obtenons de légers meilleurs résultats comme en témoigne les Table 18 et 19.

	0	**1**
Cluster 0	16380	6984
Cluster 1	3922	2714

TABLE 18 – Comparaison *clusters* avec l'approche CAH (*linkage complete* et distance de Manhattan) et les prédictions réelles.

	Precision	*Recall*	*Accuracy*
0	0.81	0.70	
1	0.28	0.41	
			0.64

TABLE 19 – Analyse des *clusters* avec l'approche CAH (*linkage complete* et distance de Manhattan)

4.3 Clustering spectral

Le clustering spectral [7] est très utilisé quand on travaille avec des graphes. En effet le clustering spectral est un algorithme de classification non supervisé qui permet de définir des *clusters* de nœuds sur des graphes. L'algorithme est basé sur la décomposition spectrale du Laplacien du graphe $(\mathbf{D} - \mathbf{A})$ où $\mathbf{D}$ est la matrice des degrés et $\mathbf{A}$ la matrice d'adjacence. On remarque que l'algorithme ne repose pas sur le graphe en lui même, mais uniquement sur une matrice d'adjacence (ou similarité) issue de ce graphe. Il est par conséquent possible d'utiliser cet algorithme pour des données non issues d'un graphe à partir du moment où l'on peut calculer une matrice de similarité à partir de ces données. Il est également possible d'utiliser des noyaux pour définir cette similarité. Généralement on utilise le noyau gaussien (noyau RBF) pour mesurer l'intensité du lien entre deux individus x_i et x_j défini comme :

$$A_{ij} = exp(-||x_i - x_j||^2/\sigma^2) \tag{5}$$

De nombreux autres noyaux existent et sont aussi utilisés (linéaire, polynomial..). Dans le cas où l'on utiliserait la distance euclidienne, voici le pseudo code d'un exemple de clustering spectral :

1. Calculer la matrice carrée des distances euclidiennes de terme général $||x_i - x_j||^2$.
2. Appliquer la fonction du noyau gaussien précédente avec une valeur d'hyperparamètre sigma à fixer.
3. Calculer la matrice Laplacienne.
4. Calculer la décomposition spectrale de la matrice Laplacienne.
5. Utiliser les vecteurs propres associés aux plus faibles valeurs propres comme nouvelle représentation vectorielle de nos individus.
6. Utiliser un kmeans avec cette nouvelle représentation.

Pour notre test, nous utilisons un noyau gaussien. Pour cette méthode nous avons été contraints de diviser nos données car la mémoire était surchargée. Nous avons donc gardé uniquement 66% de nos données. Pour une comparaison juste nous forçons le fait qu'il y ait la même proportion de label 0 et de label 1 pour la variable Y dans notre échantillon que dans l'ensemble des données de départ. Nous obtenons les résultats mentionnés en Table 20.

	Precision	*Recall*	*Accuracy*
0	0.82	0.51	
1	0.26	0.60	
			0.53

TABLE 20 – Analyse des *clusters* avec la méthode de clustering spectral.

4.4 DBSCAN

Pour aller plus loin nous essayons une méthode basée sur la densité appelée DBSCAN. L'idée est de former autour de chaque point une boule ouverte de rayon r. Si un certain nombre de points sont à l'interieur de la boule, on crée un *cluster*. On se rend compte que les points isolé n'auront pas de *cluster*, on les représente donc comme des points "bruit". Cette méthode requiert donc deux paramètres :

— r : rayon (maximal) de voisinage.
— MinPts : nombre minimum de points dans le voisinage (défini par r) d'un point pour qu'il soit dense.

Néanmoins, après avoir réalisé un GridSearch sur les données nous n'obtenons pas de résultats intéréssants [1].

4.5 Concensus Clustering

Pour cette section nous avons créé une fonction `condorcet matrix` prenant en entrée differentes partitions (dans notre exemple 7 méhtodes), et retournant en sortie une matrice carrée indiquant pour chaque paire d'individus, le nombre de partitions les ayant mis dans le même cluster. Notre fonction `consensus clustering` va permettre de réaliser un clustering spectral défini 4.3. En effet la matrice d'*affinity* pour construire le graphe sera la matrice de condorcet. Le résultat de cette méthode est retranscrit en Table 21.

	Precision	*Recall*	*Accuracy*
0	0.74	0.48	
1	0.18	0.40	
			0.46

TABLE 21 – Analyse des clusters avec la méthode de consensus clustering.

La mesure NMI (*Normalized Mutual Information*) a été calculée pour toutes nos méthodes, néanmoins elle est généralement de l'ordrde de 10^{-3} ce qui est un résultat très faible pour être significatif.

5 Apprentissage dans un contexte déséquilibré

Dans ce projet nous avons mis en place un certain nombre de méthodes supervisées et non supervisées pour prédire des clients en défaut de paiement. Le contexte déséquilibré de la problématique a rajouté une difficulté. Pour la partie non-supervisée la détection des individus n'a pas été très frucutante bien que nous ayons réalisé toutes les methodes demandées et en essayant de proposer d'autres approches. Les methodes Kmeans avec initialisation Kmeans++, CAH avec la distance de Manahthan... semblent donner des résultats légérement plus intéréssants, et pour certaines, battent les méthodes supervisées sur le *recall* de la classe minoritaire. Une manière d'ameliorer nos résultats

1. Les résultats sont réalisables dans notre notebook.

serait d'apprendre une metrique sur nos données [8] [1]. Pour la partie surpervisée, sans surprise la méthode Random Forest plus robuste donne des résultats globaux corrects mais peine à détecter correctement les défauts de paiement. Pour aller plus loin nous avons aussi fait des tests en effectuant un sur-echantillonage sur nos données grâce à la librairie *imbalanced learning*. Nous effectuons différentes techniques de réchantillonage :

- **RandomOverSampler** est une stratégie consistant à générer de nouveaux échantillons en échantillonnant aléatoirement en remplaçant les échantillons.
- **SMOTE**, acronyme pour Synthetic Minority Oversampling TEchnique [3], est une méthode de suréchantillonnage des observations minoritaires. Le SMOTE va générer de nouveaux individus qui ressemblent aux autres, sans être identiques aux individus de la classe minoritaire. L'algorithme est le suivant :

1 Sélectionner aléatoirement une observation minoritaire "initiale".
2 Identifier les k plus proches voisins parmi les individus minoritaires.
3 Choisir aléatoirement l'un des k plus proches voisins.
4 Générer aléatoirement un coefficient $0 < \alpha < 1$.
5 Créer un nouvel individu entre l'observation initiale et le plus proche voisin choisi, selon la valeur du coefficient.

- **SVMSMOTE** Variante de l'algorithme SMOTE qui utilise une méthode SVM pour détecter l'échantillon à utiliser pour générer de nouveaux échantillons de la classe minoritaire.
- **BorderlineSMOTE** Autre variante de l'algorithme SMOTE.

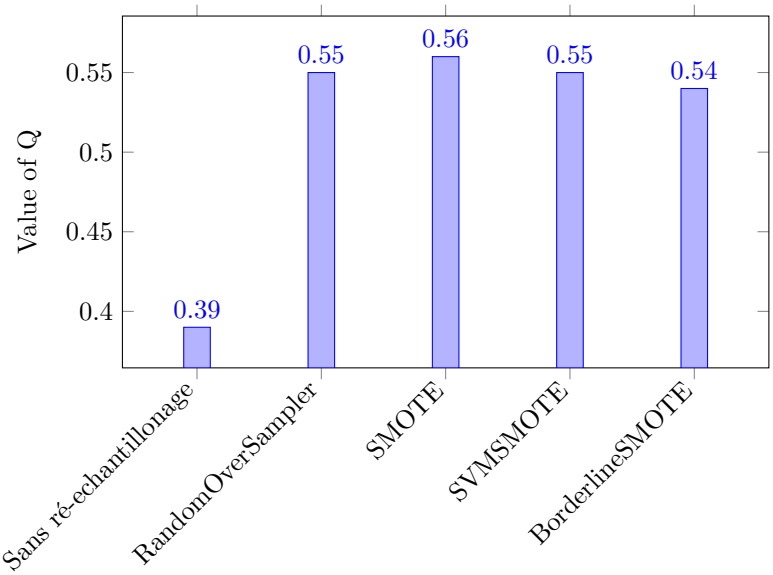

FIGURE 3 – Evolution du rappel pour la classe minoritaire.

La Figure 3 revient sur les résultats de nos différentes méthodes de rééchantillonage. On observe bien qu'après avoir fait un rééchantillongage, le *recall* de la classe minoritaire est plus élevé.

6 Conclusion

Ce projet démontre dans un premier temps l'intérêt d'utiliser des méthodes ensemblistes pour la tâche de classification mais également la complexité pour optimiser ces modèles. Il montre également les limites de ces méthodes dans un contexte déséquilibré, auquel nous proposons en dernière partie une solution possible pour renforcer les performances des méthodes. La partie sur l'apprentissage non supervisé quant à elle exprime la difficulté de visualiser des données en grande dimension, mais montre des résultats prometteurs notamment sur le *recall* de la classe déséquilibré. En revanche, la méthode de consensus clustering n'apporte pas de réelles améliorations comparé aux méthodes plus classiques.

Références

[1] Aurélien Bellet, Amaury Habrard, and Marc Sebban. A survey on metric learning for feature vectors and structured data. *arXiv preprint arXiv :1306.6709*, 2013.

[2] Leo Breiman. Bagging predictors. *Machine Learning*, 1996.

[3] 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.

[4] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 1995.

[5] Adele Cutler, D. Richard Cutler, and John R. Stevens. *Random Forests*, pages 157–175. Springer US, 2012.

[6] Leland McInnes, John Healy, and James Melville. Umap : Uniform manifold approximation and projection for dimension reduction. In *arXiv preprint*, 2018.

[7] Ulrike Von Luxburg. A tutorial on spectral clustering. In *Journal of Statistics and Computing*, 2007.

[8] Eric Xing, Michael Jordan, Stuart J Russell, and Andrew Ng. Distance metric learning with application to clustering with side-information. *Advances in neural information processing systems*, 15, 2002.

A Annexe 1

A.1 Évolution de l'accuracy en fonction des hyperparamètres - Random Forest

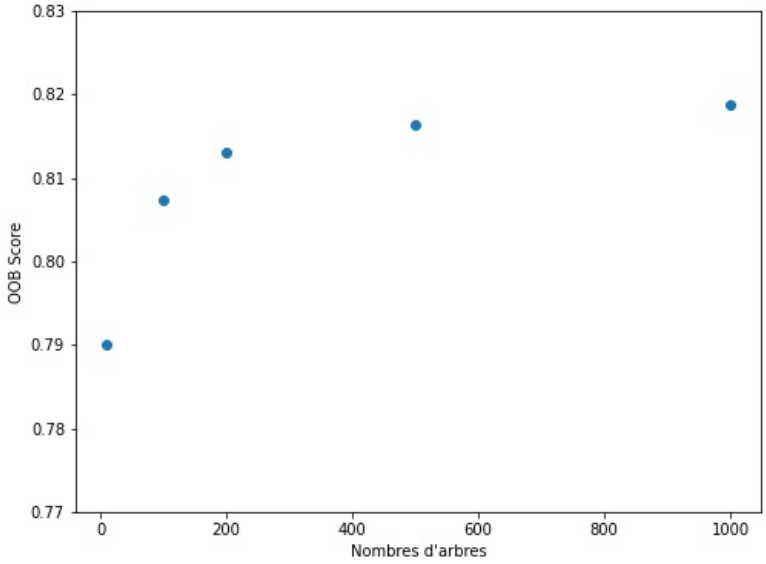

FIGURE 4 – Évolution de l'erreur OOB en fonction du nombre d'arbres.

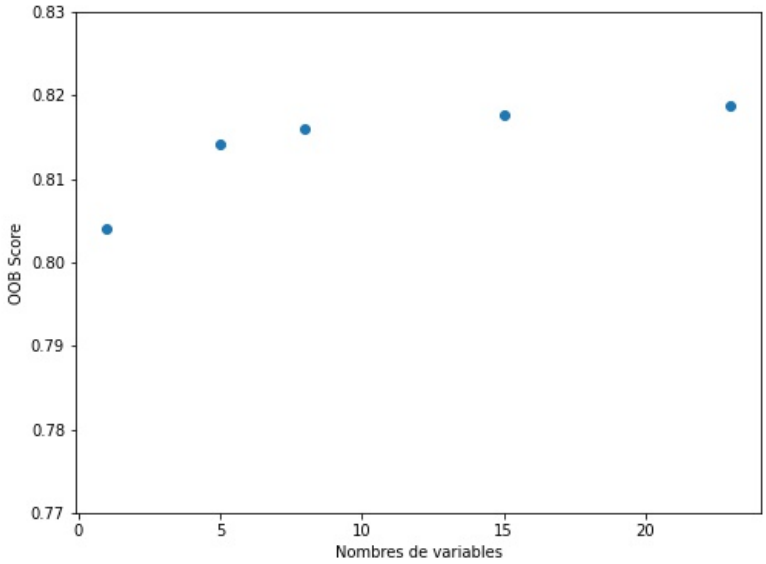

FIGURE 5 – Évolution de l'erreur OOB en fonction du nombre de variable.

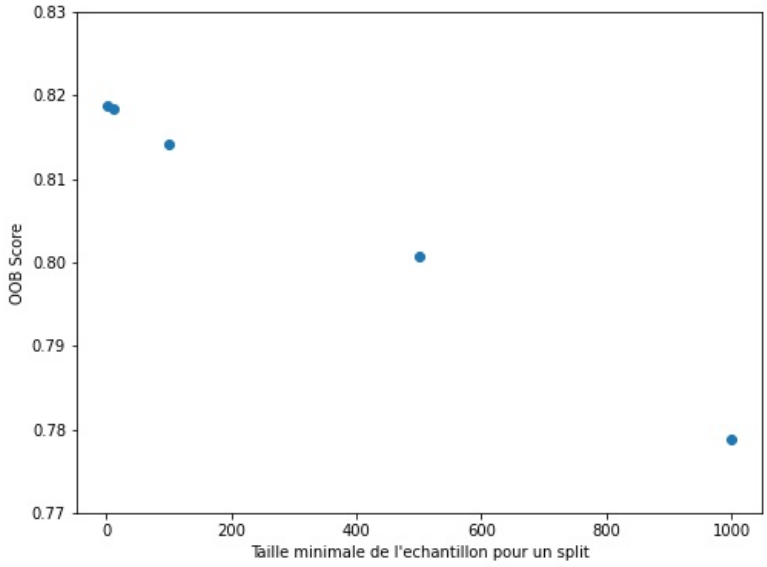

FIGURE 6 – Évolution de l'erreur OOB en fonction de la taille minimale de l'echantillon pour un *split*.

A.2 Évolution de l'accuracy en fonction des hyperparamètres - AdaBoost

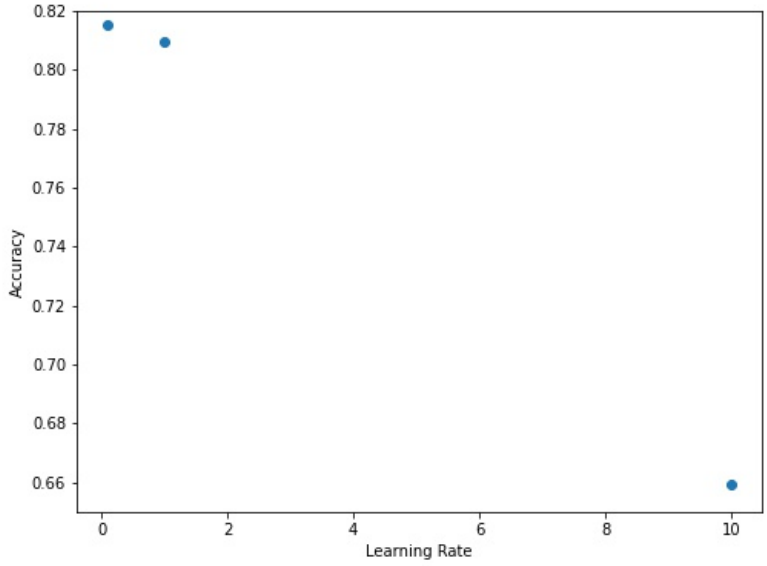

FIGURE 7 – Évolution de l'*accuracy* en fonction du *learning rate*.

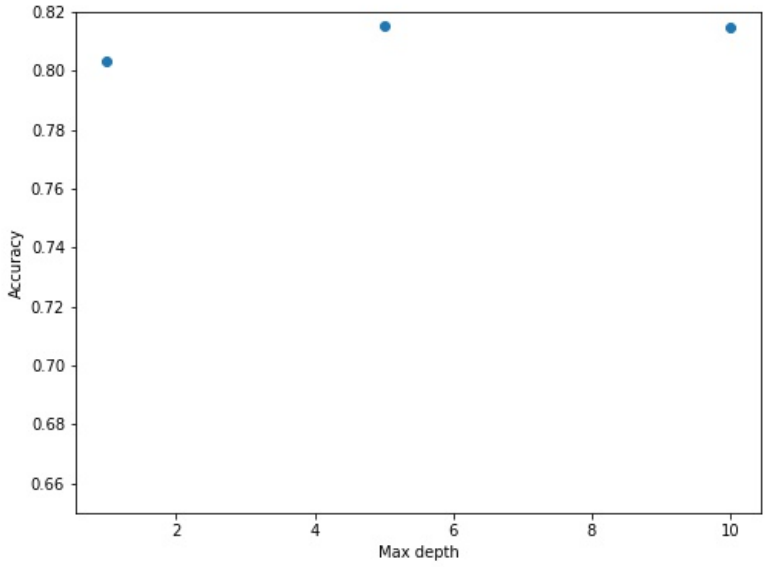

FIGURE 8 – Évolution de l'*accuracy* en fonction de la profondeur maximum des arbres.

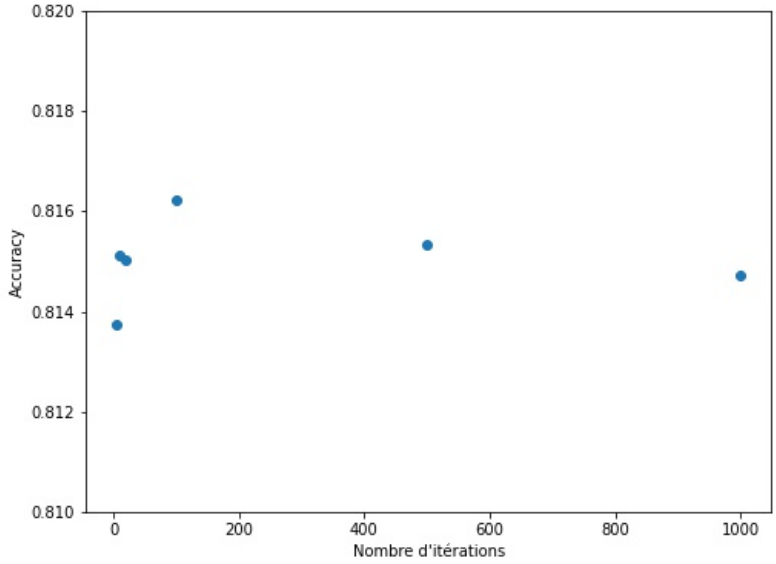

FIGURE 9 – Évolution de l'*accuracy* en fonction du nombre d'itérations.

A.3 Évolution de l'accuracy en fonction des hyperparamètres - Gradient Boosting

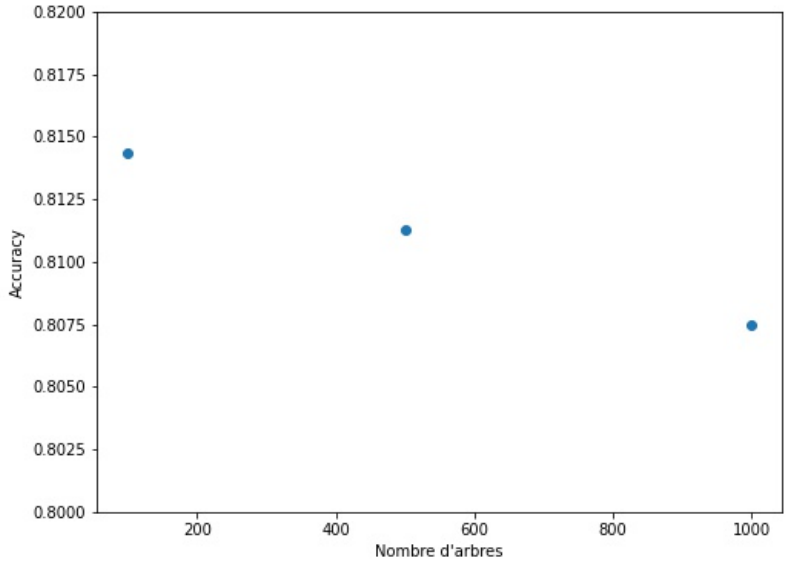

FIGURE 10 – Évolution de l'*accuracy* en fonction du nombre d'itérations.

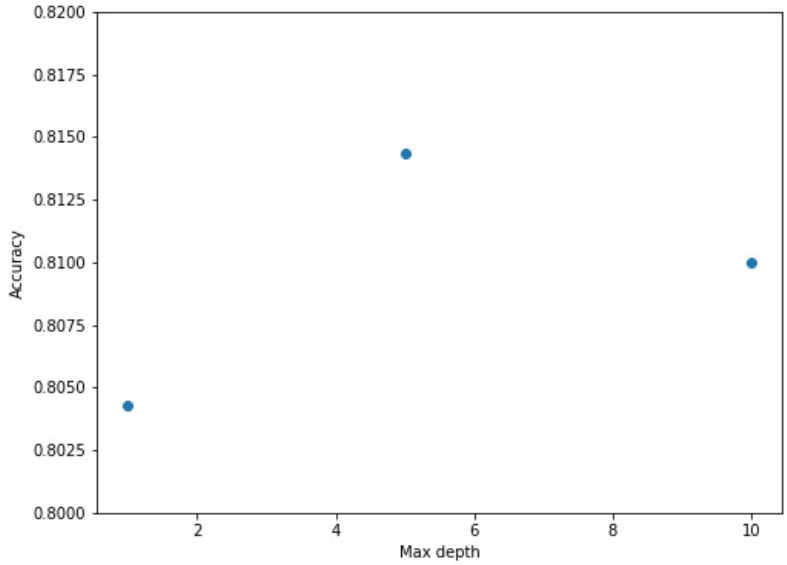

FIGURE 11 – Évolution de l'*accuracy* en fonction de la profondeur maximale des arbres.

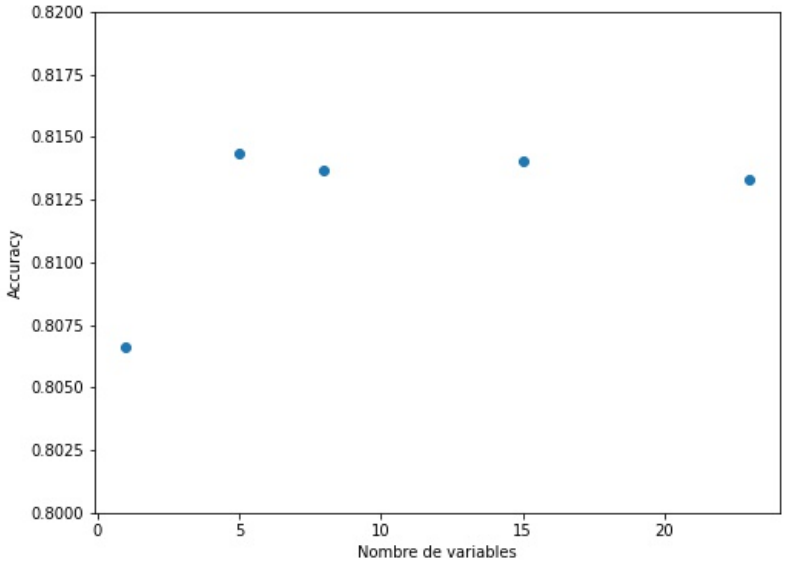

FIGURE 12 – Évolution de l'*accuracy* en fonction du nombre de variables choisies pour *split* un noeud.

B Annexe 2

B.1 Projection des données avec UMAP

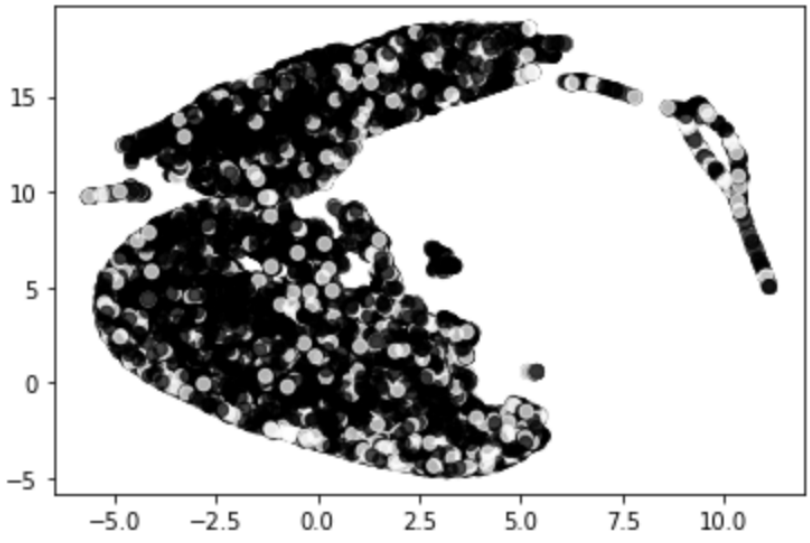

FIGURE 13 – Projections des individus avec UMAP.

B.2 Projection des données avec ACP

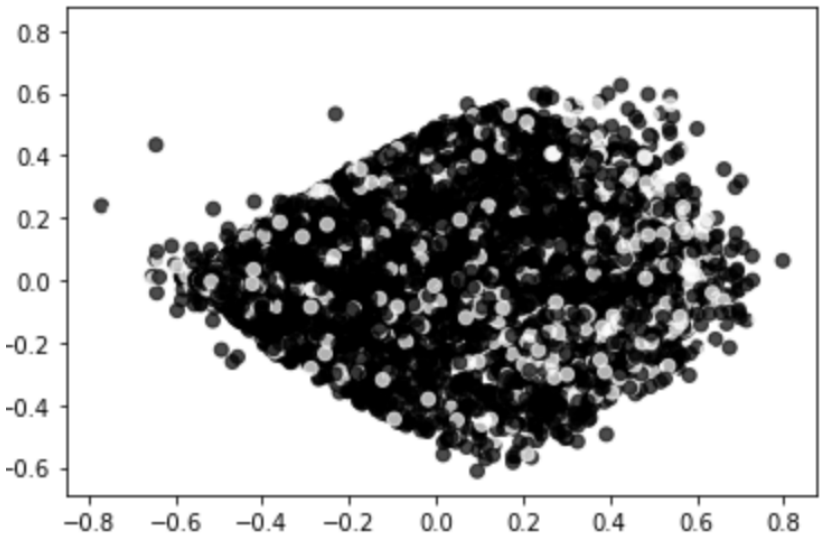

FIGURE 14 – Projections des individus avec ACP.

C Annexe 3

C.1 Représentation des centroïdes par la méthode des kmeans

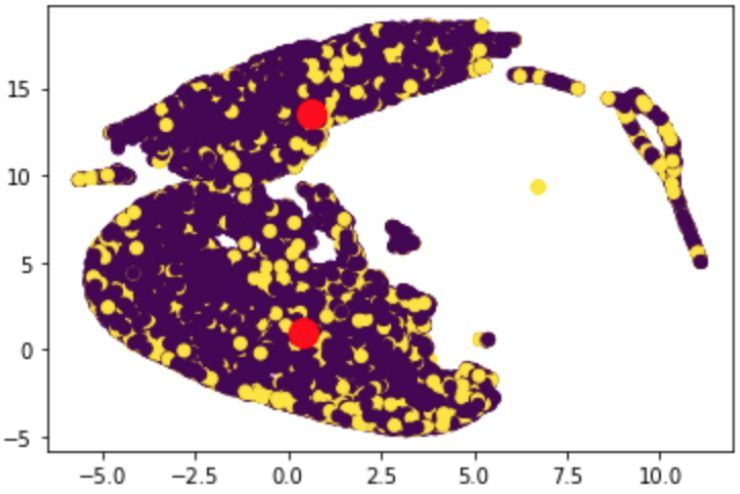

FIGURE 15 – Représentation des centroides des kmeans++ sur nos données (en jaune les individus en défaut de paiement).