UNIVERSITÉ DE NANTES

IAE NANTES
ÉCONOMIE & MANAGEMENT

Détection de fraude sous Python

Thibaud MEYNIER & Lucas AUROUET

Dossier d'évaluation pour le Master 2
Econométrie et statistiques appliquées
sous la direction de de Mr Lenouvel

Master 2 EKAP

IAE Nantes

16/01/2021

1 Cadre et Méthodologie

1.1 Cadre

L'objectif de ce dossier est, à l'aide de modèles de machine learning, de détecter des transactions financières par téléphone frauduleuses. Le but étant de délivrer une information utile, avec des modèles suffisamment sensibles pour prédire la fraude mais aussi pour éviter la détection de fraudes non avérées. Pour ce faire, nous disposons d'un échantillon d'entraînement de 500,000 observations, puis d'un échantillon de test de 250,000 observations. Nous utiliserons trois modèles différents ; un arbre de classification, un Forêt Aléatoire et l'algorithme de Gradient Boosting. Nous pourrons comparer leur performance en termes de détection de transactions frauduleuses et observer comment certains hyperparamètres peuvent faire varier la qualité des résultats. Nous pourrons également définir l'importance de chaque variable dans la classification des transactions et l'efficacité de notre meilleur modèle en termes financiers.

Nous allons tout d'abord dressser la liste des variables disponibles dans le jeu de données, et brièvement analyser les données disponnibles Nous pourrons éventuellement tenter de créer d'autres variables pour nous aider à correctement classifier les transactions.

— IsFraud : la transaction est-elle frauduleuse ? 1 si oui 0 si non.

— Amount : le montant de la transaction

— OldBalance(Orig, Dest) : la valeur du compte avant transaction (Émetteur, Destinataire)

— NewBalance(Orig, Dest) : la valeur du compte après transaction (Émetteur, Destinataire)

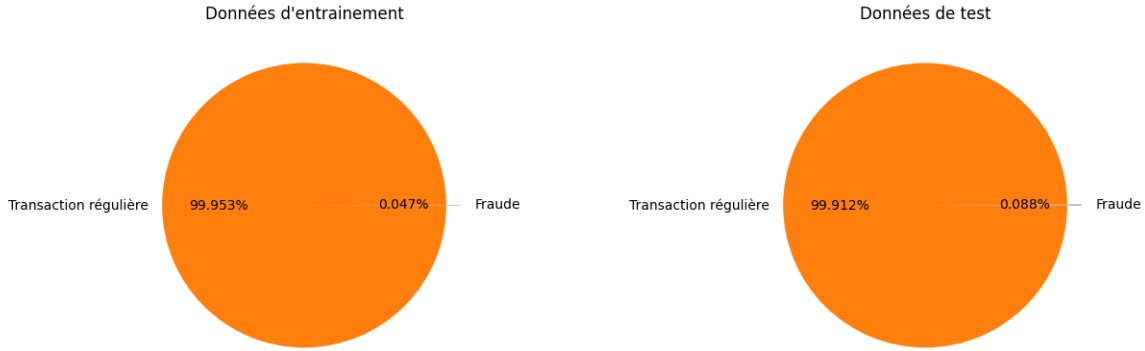


FIGURE 1 – Répartition de la variable **IsFraud**

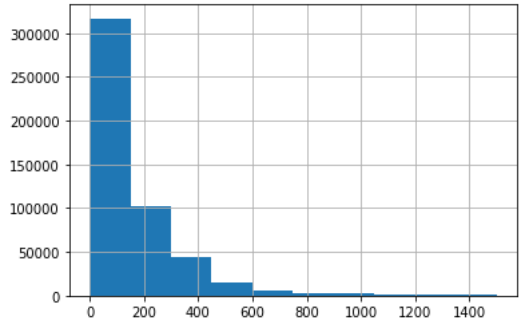
La répartition de la variable n'est pas du tout symétrique, nous avons moins de 0.1% de fraudes avérées dans les deux jeux de données. Entre les jeux de données d'entraînement et de test, cette fréquence n'est pas la même, 0.047% dans le jeu d'entraînement pour 0.088% dans le

jeu test. Les fraudes sont donc plus fréquentes dans le jeu test, c'est une information qu'il faut garder à l'esprit lors de la modélisation. En effet si le modèle s'entraîne avec l'idée que les fraudes sont très rares, il risque de mal les détecter dès lors que l'on passe sur un jeu de données où elles sont plus fréquentes. le modèle risque, entre autres, de classer injustement des transactions régulières en fraude. Les autres variables sont détaillées ci-dessous.

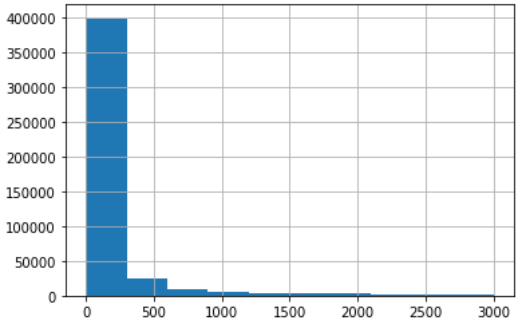
	Montant	OldBalanceOrig	OldBalanceDest	NewBalanceOrig	NewBalanceDest
moyenne	166393.7	911692.8	982773.9	931426.1	1162668.16
écart-type	272584	3016900.5	2336426	3054014.8	2510610.4
	Montant	OldBalanceOrig	OldBalanceDest	NewBalanceOrig	NewBalanceDest
moyenne	163976.1	820736.6	988382.9	842553.3	1098219.5
écart-type	267078.4	2849500.3	2285586	2886191.7	2356826.7

FIGURE 2 – Graphique des variables explicatives du jeu d'apprentissage

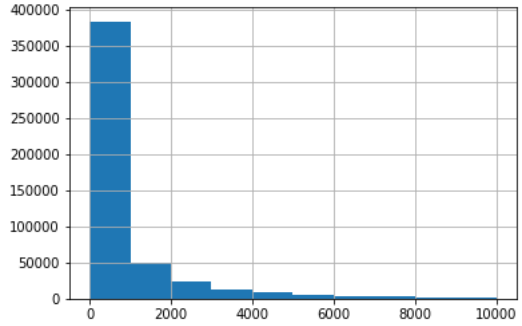
(a) Montant



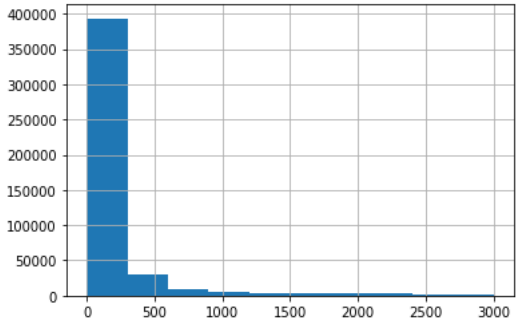
(b) Balance émetteur départ



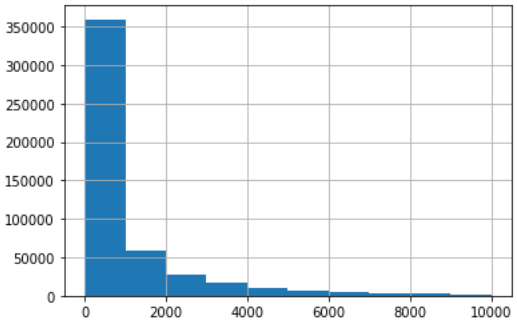
(c) Balance receveur départ



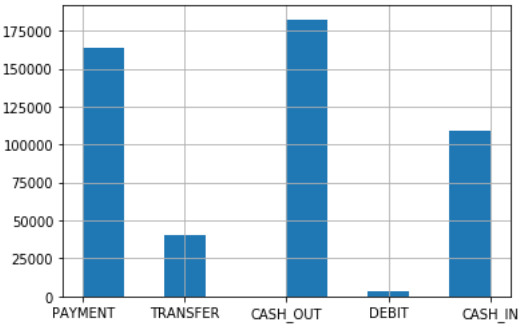
(d) Balance émetteur arrivée



(e) Balance receveur arrivée



(f) Type de transaction



le montant moyen des transactions est de 166393.7 sur le jeu de données train contre 163976.1

dans le jeu de données test. Les montants sont donc légèrement moins importants, en moyenne, sur le jeu de données test, mais les deux valeurs sont du même ordre de grandeur. Les autres variables sont également distribuées de la même façon dans les deux jeux. Cette propriété est souhaitable pour les mêmes raisons que nous avons citées plus haut avec la variable IsFraud. Si les deux jeux sont trop différents, les phénomènes retenus par le modèle lors de l'entraînement auront changés ou disparus dans le jeu test et il en résultera une qualité de prédiction amoindrie.

En ce qui concerne la distribution des variables présentée dans la figure 2, nous notons que les données monétaires [1], que ça soit le montant de la transactions, ou les différentes balances de comptes, sont distribuées asymétriquement. Enfin, pour le type de transaction, on note qu'il y a 4 types qui sont fréquemment utilisés parmis les 5 possibles.

1.2 Méthodologie

Pour tenter d'identifier les fraudes, nous avons eu recours à différents modèles de machine learning. Nous allons, avant d'analyser les résultats, présenter brièvement le fonctionnement des modèles en question.

— CART : Classification And Regression Tree, CART est un arbre de décision. On essaye de trouver la variable et la valeur de cette variable qui scinde les données en deux sous-groupes de sorte à maximiser la variance inter-groupes tout en minimisant la variance intra-groupe. Cette opération est répétée un certain nombre de fois et chaque sous-groupe est re-divisé ainsi de suite. On atteint idéalement un stage où chaque sous-groupe (feuille) est constitué uniquement de transactions régulières ou uniquement de fraudes.

— Forêt Aléatoire : les Forêts Aléatoires sont des ensembles d'arbres CART. Les résultats de chaque arbre sont agrégés pour fournir un seul et unique résultat. Cette méthode dite d'ensemble, tire profit de l'agrégation de plusieurs modèles simples construits de sorte à être aussi différents les uns des autres que possible. Ces modèles simples sont souvent limités, les CART en l'occurrence sont connus pour être très sensibles au jeu de données et donnent des résultats très différents dès lors que certaines observations changent, ils sont également soumis au sur-apprentissage si on ne limite pas leur croissance [2]. La forêt aléatoire permet de remédier à ces deux problèmes en ajoutant une partie stochastique dans la construction du modèle et en agrégeant plusieurs modèles non-restreints.

— Forêt Aléatoire (grid search) : Les modèles mentionnés ci-dessus possèdent nombre de paramètres à choisir, nous avons parlé de la croissance de l'arbre qui en est un exemple. Le

1. les données sont exprimées en milliers
2. La croissance d'un arbre est représentée par le nombre de divisions du jeu qui sont effectuées

nombre d'arbres dans la forêt ou bien le nombre de variables candidates pour diviser le jeu de données [3] sont aussi des paramètres qui doivent être déterminés à priori. En machine learning, une méthode très pratique qui nous aide à choisir ces paramètres est l'évaluation par validation croisée d'un ensemble de valeurs possibles pour ces paramètres (grid search). L'erreur de cross-validation est calculée pour chaque valeur de paramètre considérée et on choisi simplement la valeur qui minimise l'erreur de cross-validation.

— Forêt Aléatoire (gradient boosting) : dans une forêt "standard" les arbres sont construits simultanément et c'est le caractère aléatoire et ensembliste du modèle qui donne des résultats fiables. En revanche le modèle n'apprend pas. Le gradient boosting consiste à introduire une descente de gradient dans le processus de création d'une forêt aléatoire. Chaque arbre apprend ainsi des erreurs de l'arbre précédent et tente de les corriger, ils sont donc construits de manière itérative. Ici la descente de gradient cherchera à optimiser les poids des observations de sorte à ce que l'erreur de prédiction diminue d'arbre en arbre.

— Forêt Aléatoire (extreme gradient boosting) : très similaire au gradient boosting, XGB pour Xtreme Gradient Boosting repose sur une descente de gradient qui rend chaque arbre plus performant que le précèdent. En revanche cette technique est, selon son auteur, plus performante et permet de mieux contrôler le sur-apprentissage.

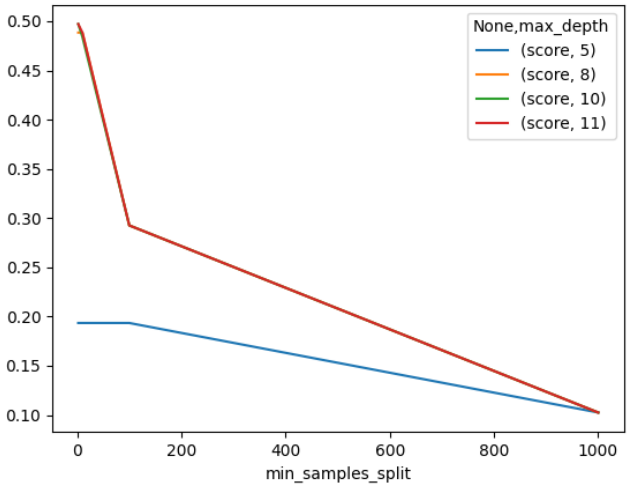
Nous allons tenter d'identifier les transactions frauduleuses à l'aide de ces modèles susmentionnés. En premier lieu nous utiliserons un CART seul, puis une Random Forest optimisée par grid search et enfin l'algorithme XGB de gradient boosting. Comme nous l'avons mentionné plus haut, les modèles contiennent des paramètres qui peuvent substantiellement impacter la qualité des résultats. Dans un arbre de décision, ces paramètres sont : la profondeur (nombre de fois où le jeu de données en divisé), et le nombre d'observations requis dans un groupe pour que ce groupe puisse être divisé. Ces deux paramètres vont donc jouer sur la qualité d'ajustement du modèle et il faut choisir des valeurs telles que la classification soit suffisamment précise tout en évitant le sur-apprentissage. L'identification de ces paramètres peut se faire par validation croisée (grid search). On pourrait également choisir ces paramètres en fonction de nos hypothèses à priori, néanmoins, il est très difficile d'anticiper quelles sont les valeurs optimales. Nous procédons donc à une recherche par cross validation (10-folds). Le score de chaque couple de valeurs est représenté dans le graphique ci-dessous. Nous évaluons min_samples_split (1000, 1000, 10, 2) et max_depth [4] (5, 8, 10, 11) conjointement. Le score utilisé ici, et dans les autres estimations est le

3. Pour obtenir des arbres différents, on n'essaye qu'une partie des variables disponibles ainsi pour le même noeud (endroit où les données sont divisées en deux), deux arbres choisiront des variables différentes pour scinder le jeu

4. Le nombre d'observations minimum pour qu'un noeud puisse être divise et le nombre de fois où l'on découpe le jeu de données respectivement.

Recall, défini comme $\frac{TruePositives}{FalseNegatives+TruePositives}$. Dit autrement, le Recall représente combien de transactions ont été identifiées comme fraudes par rapport à la totalité des fraudes effectivement avérées. Cette métrique nous donne une bonne idée de la capacité de notre modèle à détecter les fraudes. A titre de comparaison, la Precision mesure combien d'observations classées comme fraudes le sont réellement et permet d'avantage de vérifier que notre modèle n'identifie pas à tort des transactions régulières comme étant des fraudes. Pour notre problème, le Recall nous semble préférable, puisque nous cherchons à obtenir des modèles capables de détecter la moindre fraude, quitte à identifier des transactions régulières comme étant des fraudes. Nous verrons plus tard plus en détails cette démarche et en quoi ce choix est particulièrement justifié dans notre cas.

FIGURE 3 – Grid Search pour l'arbre de décision



Min_samples_split semble donner une erreur de cross-validation minimale pour une valeur de 2 et cette erreur décroît avec min_samples_split, ce qui semble suggérer qu'une valeur la plus faible possible donne les meilleurs résultats. Nous choisissons donc la valeur de 2 pour ce paramètre. De même, max_depth semble être optimal pour des valeurs plus élevées, 11 donne le meilleur score. Ces résultats indiquent que pour résoudre ce problème, des modèles plus complexes sont privilégiés.Nous pouvons désormais estimer un arbre avec ces valeurs optimales, et tester le modèle sur le jeu test pour vérifier sa validité. Les performances de l'ensemble des modèles est présentée plus bas.

Il est maintenant nécessaire de trouver les paramètres optimaux pour la forêt aléatoire. A savoir ; le nombre d'arbres (n_estimartors), la profondeur (max_depth), la proportion de variables qui seront aléatoirement choisies pour scinder les données à chaque noeud (max_features) ainsi que min_sample_split. Pour diminuer nos temps de calcul, nous avons été contraints de diviser

cette optimisation en deux parties. Le tremps de calcul requis pour une grid search par cross validation est égal à $\frac{V*P*A*T*CV*}{C}$ avec V le nombre de valeurs essayées par paramètre optimisé, P le nombre de paramètres, A le nombre d'arbres, T le temps moyen d'estimation d'un arbre, CV le nombre de plis de cross-validation et C le nombre de processeurs utilisés simultanément pour le calcul. Pour pouvoir optimiser les paramètres dans un délai raisonnable, nous optimisons d'abord le nombre d'arbres en fixant les autres paramètres. Puis nous utilisons la grid search pour optimiser les paramètres restants avec le nombre d'arbres optimal calculé dans un premier temps. Nous parcourons les valeurs 1, 5, 10, 50, 100, 250, 500, 1000, 2500 et récupérons la précision pour chaque valeur, les autres paramètres étant respectivement fixés à :

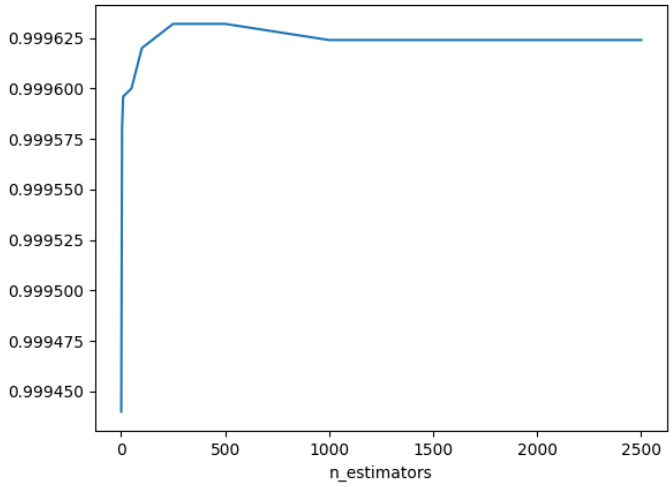
— min_samples_split : 2, comme suggéré par l'optimisation du CART plus haut.

— max_features : 7.

— max_depth : 11.

Nous obtenons un mesure de l'évolution du score (in-sample) en fonction du nombre d'arbres que nous représentons ci-dessous.

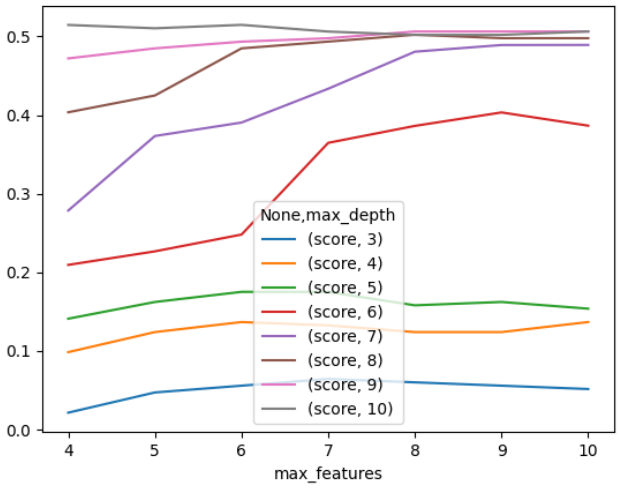
FIGURE 4 – Erreur de Cross Validation fonction du nombre d'arbres de la Foret



Il apparaît qu'après 300 arbres, il n'y a quasiment aucun gain en terme de précision. L'erreur augmente légèrement dès lors que le nombre d'arbres dépasse 500. Nous choisissons donc 300 arbres pour estimer nos Forets Aléatoires et nous conserverons cette valeur pour estimer le Gradient Boosting. Nous pouvons désormais évaluer différentes valeurs de max_features (4, 5, 6, 7, 8, 9, 10) et max_depth (3, 4, 5, 6, 7, 8, 9, 10). Nous procédons à uhne validation croisée 5-folds [5]. n_estimators et min_samples_split sont fixés conformément aux résultats présentés plus haut.

5. 10-folds prendrait un temps considérable.

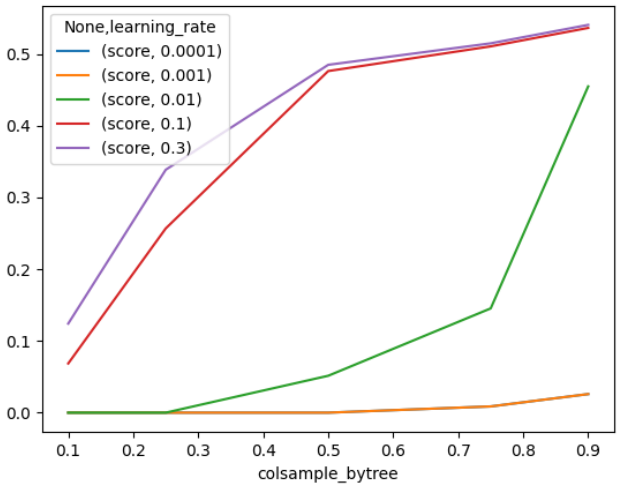
FIGURE 5 – Grid Search pour les paramètres max_features et max_depth



L'évolution du Recall en fonction des valeurs des paramètres est représenté ci-dessus. On voit que les valeurs optimales semblent être atteintes pour max_depth égal à 10. Max_features ne semble pas avoir un impact drastique sur le recall. Nous choisissons donc max_depth = 10 et max_features = 6.

L'algorithme de Gradient Boosting requiert lui aussi des paramètres spécifiques : max_depth, et n_estimators sont déjà connus. Nous optimisons donc leanring_rate, qui représente la vitesse de convergence de la fonction de perte vers son minimum (espérons le, global) et colsample_bytree qui est un paramètre analogue de max_features pour la fonction XGBClassifier du module xgb.

FIGURE 6 – Grid Search pour les paramètres du gradient boosting



Pour le modèle de Gradient Boosting, les paramètres optimaux d'après la méthode de grid

search sont de 0.3 pour le learning rate et 0.9 pour la proportion de variables choisies aléatoirement pour séparer le jeu de données en deux. On préfère souvent un learning rate inférieur à 0.1, en effet, plus le learning rate est faible plus la convergence vers le minimum est assurée. Le coût associé est un temps de calcul supérieur, mais un learning rate faible n'entraîne aucun biais supplémentaire. A l'inverse un learning rate trop élevé peut entraîner une estimation sous-optimale, dans le cas où la descente de gradient "passerait à coté" du minimum. La fonction de perte ne serait alors pas minimisée et il est facile de prouver que dans ce cadre, l'estimation n'est pas fiable. Cependant la recherche par cross-validation indique qu'un learning rate de 0.3 est optimal, nous conservons donc cette valeur.

2 Résultats

Les résultats présentés dans le tableau sont les résultats pour les modèles avec les paramètres optimisés. D'après nos résultats, le modèle le plus performant est le modèle de Gradient boosting, qui arrive à prédire correctement 60.63% des fraudes sur l'échantillon test. La Forêt Aléatoire est en deuxième position avec 59.27% de fraudes identifiées. L'arbre de décision semble être le modèle le moins performant avec 56.56% de fraudes identifiées. XGB permet également d'obtenir le moins de faux positifs, ce qui se traduirait en situation réelle par le moins de transaction bloquées pour soupçon de fraude invarée (0.0452% des transactions dans le cas du Gradient Boosting). Enfin, ce modèle permet d'éviter presque 25,275 millions d'euros de fraude, contre 24,819 millions pour le CART. Nous présentons ensuite les matrices de confusion pour chaque modèle.

	CART	Random Forest	Xtreme Gradient Boosting
Recall score OOS	0.57	0.59	0.61
Fraudes détéctées OOS (%)	56.56%	59.27%	60.63%
Faux positifs OOS (%)	4.01%	0.0452%	0.0452%
Fraude évitée (milliers)	24 819.7	25 352.5	25 751.1

CART

-	0	1
0	249770	9
1	96	125

Forêt Aléatoire

-	0	1
0	249778	1
1	90	131

Gradient Boosting

-	0	1
0	249778	1
1	87	134

TABLE 1 – Matrices de Confusion

Les matrices de confusion permettent de voir concrètement comment les observations sont classées par notre modèle vis-à-vis de leur classe effective. Avec sickit-learn, les vraies valeurs sont représentées sur l'axe horizontal et les prédictions sur l'axe vertical. Dans le cas du XGB,

134 observations qui appartiennent à la classe 1 sont classées comme appartenant à la classe 1, autrement dit 134 fraudes sont correctement identifiées par le modèle. 1 observation est classée comme fraude alors qu'elle n'en est pas une et 87 fraudes sont manquées, étant classées 0 (transactions régulière). Si l'on regarde la matrice de la Forêt Aléatoire, on voit que 131 fraudes ont été identifiées (True Positive), moins que pour le XGB, 1 transaction régulière est classée comme fraude (False Positive), autant qu'avec XGB et 90 fraudes sont manquées (False Negative), 3 de plus qu'avec XGB. On retrouve donc ici les résultats présentés plus haut ; XGB permet d'identifier le plus de fraudes (True Positive), tout en minimisant la gêne occasionnée en bloquant des transactions régulières (False Positive). Ce résultat est intéressant car notre modèle à été optimisé avec l'idée de maximiser le taux de True Positives (avec le choix du Recall comme métrique d'évaluation des modèles) au risque d'augmenter le taux de False Negatives, il parvient néanmoins à concilier les deux. Il est possible d'encore améliorer le modèle en ajustant cette classification comme nous le verrons dans la section suivante.

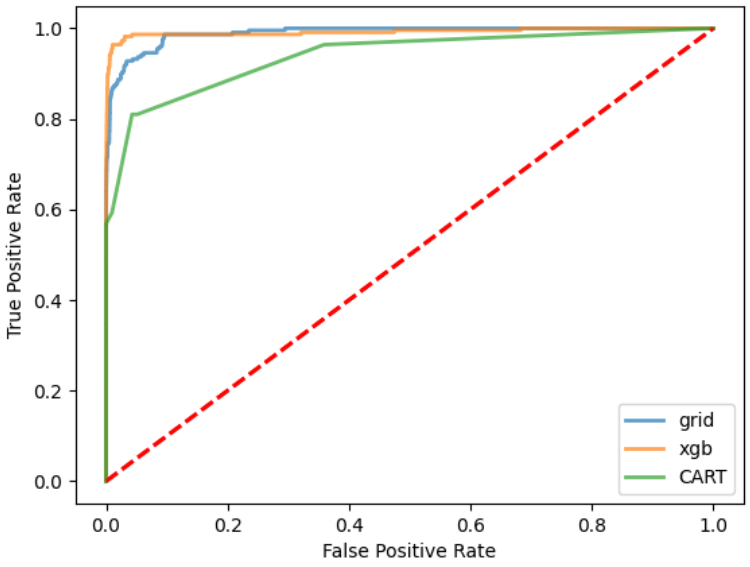


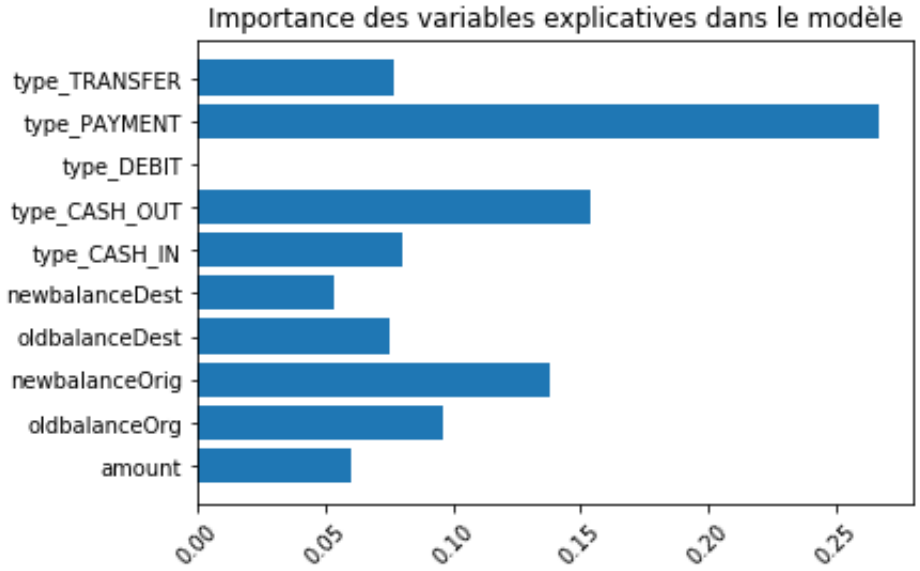
FIGURE 7 – ROC

Le graphique ci-dessus représente les ROC de nos trois modèles. Les courbes ROC permettent de rendre compte de la capacité de classification des modèles considérés en fonction de de plusieurs seuils de probabilité. Les modèles que nous utilisons ici prédisent l'appartenance d'une observation à une certaine classe en fonction de la probabilité qu'a cette observation d'appartenir à chaque classe. Si la probabilité que l'observation soit une fraude dépasse 80%, alors le modèle assigne l'observation à la classe fraude. 80% est un exemple qui illustre un seuil particulier, 50% est aussi envisageable et c'est la valeur qui est utilisée par défaut dans la méthode .predict()

de sickit-learn. En réalité, tous les seuils sont envisageables et la courbe ROC permet de rendre compte de l'efficacité de chacun d'entre eux. Un modèle avec un seuil très élevé, comme 0.8, ne classera dans la classe fraude que les observations qui sont très probablement des fraudes, au risque que certaines fraudes avérées ne soient pas détectées. Néanmoins, ce modèle évitera également de considérer frauduleuses des transactions régulières. Un modèle avec un seuil faible classera beaucoup plus facilement des transaction comme frauduleuses. Il est donc peu probable que des fraudes ne soient pas identifiées, mais en contrepartie on accepte de classer comme fraudes beaucoup plus de transactions régulières. En définitive, un seuil élevé minimise les Faux Positifs et un seuil faible maximise les Vrais Positifs. Idéalement, nous voudrions conjuguer ces deux effets, en pratique, et selon le problème, il est naturel de choisir un seuil différent de 0.5. Les courbes ROC permettent de voir comment le seuil agit sur ces deux effets. Elles permettent aussi de constater lequel des modèles est meilleur. L'aire en dessous de la courbe, ROC Area Under Curve (ROC AUC) représentant la capacité générale de notre modèle à bien classer les observations, indépendamment de la classe à laquelle elles appartiennent. Plus cette aire est grande, plus le modèle est capable d'assigner à chaque observation la classe qui lui correspond. On retrouve les mêmes résultats que dans le tableau vu plus haut ; le modèle XGB est le meilleur classifieur avec une ROC AUC supérieure à celle des autres modèles. XGB sera toujours plus apte à correctement classer les observations, indépendamment du seuil employé.

L'importance des variables dans la classification des transactions est présenté dans la figure 6. On note que le type de transaction est la variable qui apporte le plus d'informations. En particulier le type de transaction PAYEMENT semble être le type qui permet le mieux de déterminer si une transaction est frauduleuse ou non. Le type DEBIT n'a pas l'air significatif mais cela peut être dû au très faible nombre d'observations dans cette classe en particulier. La valeur du compte de l'émetteur après transaction semble être la seconde variable la plus importante dans la détection de fraudes.

FIGURE 8 – Importance des variables



3 Pour aller plus loin

Pour améliorer notre modèle, nous allons tenter deux approches. la première consiste à modifier le seuil de classification que nous venons d'évoquer. Même si notre modèle permet déjà de ne classer qu'une transaction régulière en fraude et 87 fraudes en transactions régulières, nous pouvons essayer de modifier le seuil par défaut pour peut être obtenir des meilleurs résultats. La deuxième solution consiste à rajouter une variable en espérant qu'elle puisse aider notre modèle à mieux appréhender les données. Cette variable est construite à partir de celles déjà présentes et prend la valeur de 1 si le numéro de compte du destinataire de la transaction n'apparaît qu'une seule fois et 0 si le compte apparaît plusieurs fois. Cette variable repose sur l'idée qu'un compte n'apparaissant qu'une seule fois laisse penser que le compte à été créé uniquement dans le but de recevoir une unique transaction frauduleuse, un compte apparaissant plusieurs fois à l'inverse laisse penser que le client effectue des transactions régulièrement, comme le ferait un client normal.

3.1 Seuil de Classification

Comme nous l'avons mentionné plusieurs fois auparavant, plusieurs paramètres peuvent impacter la capacité de notre modèle à détecter les fraudes. Utiliser le Recall comme métrique, par exemple, permet d'optimiser les modèles dans l'idée d'augmenter le ratio de True Positives, quitte à générer de la gêne en contrepartie. L'arbitrage True Positives / False Positives dépend de cette métrique, mais aussi du seuil de classification. Jusqu'ici, les prédictions ont été réalisées

avec la valeur par défaut, 0.5. Nous allons désormais essayer de choisir des seuils plus faibles (0.1, 0.25) et plus élevés (0.75, 0.9) dans l'espoir d'améliorer notre modèle. Nous devrions voir que les seuils faibles augmente les chances d'identifier des fraudes mais génère de la gêne supplémentaire, un seuil élevé devrait procurer l'effet inverse.

Dans le tableau suivant nous représentons le Recall et la Precision pour les seuils considérés.

seuil	0.1	0.25	0.5	0.75	0.9
Recall	63%	61%	60%	54%	50%
Precision	94%	95%	99%	99%	100%

TABLE 2 – Résultats des prédictions avec différents seuils

Comme prévu, les seuils élevés maximisent la Precision et les seuils faibles maximisent le Recall. On arrive à 100% de Precision avec un seuil à 0.9 et à 63% de Recall avec un seuil de 0.1. Pour les valeurs extrêmes ; entre 0 et 8 transactions régulières sont identifiées, à tort, comme des fraudes et entre 80 et 109 fraudes ne sont pas repérées par le modèle. En revanche avec un seuil de 0.5, nous sommes à 99% de Precision, proches du 100% optimal et à 60% de Recall, bien plus qu'avec les seuils 0.75 et 0.9. Nous préférerons donc un seuil de 0.5 quitte à mal classer une seule observation, le mieux que nous puissions faire est 0, atteignable avec un seuil de 0.9, mais cela résulterait en une baisse significative du nombre de fraudes détectées. Nous conservons donc 0.5 comme seuil de classification.

3.2 Variable Supplémentaire

Nous allons désormais rajouter à l'estimation la variable "unique" qui prend la valeur de 1 si le numéro de compte n'apparaît qu'une seule fois dans la base et de 0 si le compte apparaît au moins deux fois. nous espérons que cette variable puisse nous aider à mieux identifier les transactions frauduleuses. Nous présentons ci-dessous les résultats avec et sans cette variable supplémentaire.

	Avec "unique"	Sans "unique"
Recall	57%	60%
Precision	99%	99%

TABLE 3 – Comparaison des résultats avec et sans la variable **unique**

La variable n'améliore pas le modèle, une observation est toujours classée à tort comme étant une fraude, le taux de False Positives reste le même et par extension la Precision également. De plus, avec la nouvelle variable, le modèle ne capte que 125 fraudes contre 134 cas détectés sans la variable. Il semble donc judicieux de conserver le modèle initial et d'exclure la variable que nous voulions rajouter.

4 Conclusion

Notre but était de mettre au point une technique efficace de détection de fraude par virement téléphonique à l'aide de différentes méthodes de machine learning. Nous avons utilisé des méthodes de la famille des arbres décisionels, à savoir ; un arbre de décision, une Forêt Aléatoire et une Forêt Aléatoire optimisée par Gradient Boosting. Les paramètres des différents modèles ont été optimisés par cross validation et dans la majeure partie des cas, en grid search, sur un échantillon train de 500,000 observations. Nous avons décidé de nous baser sur le Recall comme mesure principale de l'efficacité de nos modèles, avec l'objectif d'augmenter le taux de True Positives ou dit autrement de détecter le plus de fraudes possible. Les résultats obtenus sont en accord avec la littérature, XGB est le meilleur modèle, tirant profit de la descente de gradient pour optimiser la construction des arbres. La Forêt Aléatoire est plus efficace que le CART grâce aux propriétés ensembliste et stochastique qui permettent de lutter contre les fluctuations d'échantillonage et le sur-apprentissage. XGB performe relativement bien sur l'échantillon test de 250,000 observations. La gêne occasionnée est minime avec un seul False Positive, soit 0.045% des transactions. Notre modèle identifie 134 fraudes sur 221 soit 61%. Nous avons tenté d'améliorer encore notre modèle en changeant le seuil de classification mais aucune des valeurs considérées ne permettent de réellement augmenter le taux de True Positives et le léger gain en terme de Precision n'est pas suffisant pour justifier la perte de Recall associée. Ajouter une variable binaire qui repère les numéros de compte n'apparaissant qu'une seule fois dans la base n'a pas non plus permis d'obtenir une meilleure classification. Le type de transaction et la valeur du compte de l'émetteur après transaction semblent être les deux variables qui permettent au mieux de distinguer les transactions frauduleuses des transactions régulières et la performance des modèles peut facilement se traduire en termes financiers.