Détection de la fraude financière à l'aide de l'apprentissage automatique

Mohamed BEN HAMDOUNE, Lucas ISCOVICI

Paris, France

Université de Paris Descartes

Abstract

La fraude représente une perte de chiffre d'affaires de plusieurs milliards de dollars et augmente chaque année. L'enquête sur la criminalité économique mondiale menée par PwC en 2018 a révélé que la moitié (49%) des 7 200 entreprises interrogées avait été victime d'une fraude quelconque. Il s'agit d'une augmentation par rapport à l'étude PwC 2016 dans laquelle un peu plus du tiers des organisations interrogées (36%) avaient été victimes d'un crime économique.
Les méthodes traditionnelles d'analyse des données sont utilisées depuis longtemps pour détecter les fraudes et c'est dans ce contexte que nous effectuerons plusieurs manière d'aborder le problème afin de voir les avantages et désavantages de chaque modèle d'apprentissage et celui donnant les meilleurs résultats de prédiction.

Keywords: Classes déséquilibrées, SMOTE, Validation croisée, Méthodes ensemblistes, Voting/Stacking

Contents

1. Introduction

Les techniques utilisées pour détecter les fraudes nécessitent des enquêtes complexes et fastidieuses puis elles traitent de différents domaines de la connaissance tels que la finance, l'économie.
Voici des exemples de techniques d'analyse de données statistiques:

- Techniques de pré-traitement des données pour la détection, la validation, la correction des erreurs et le remplissage des données manquantes ou incorrectes.
- Calcul de divers paramètres statistiques tels que les moyennes, les quantiles, les mesures de performance, les distributions de probabilité, etc.
- Analyse chronologique de données dépendantes du temps (Séries temporelles).

Dans notre cas, nous utiliserons des modèles d'apprentissage supervisé et nous n'utiliserons pas des méthodes de clustering.
Ce type de modèles ne permet de détecter que des fraudes semblables à celles qui ont eu lieu précédemment et ont été classées par un humain.
En ce qui concerne la détection des fraudes par paiement par carte de crédit, le problème du classement implique la création de modèles suffisamment intelligents pour classer correctement les transactions en transactions légitimes ou frauduleuses, en fonction des détails de la transaction tels que le montant, le commerçant, l'emplacement, l'heure et autres.

Nous commencerons par une analyse exploratoire du jeu de données avec la particularité que nos données ne sont pas équilibrées.

2. Analyse exploratoire des données

Notre ensemble de données contient les transactions effectuées par carte de crédit en septembre 2013 par les titulaires de carte européens. Cet ensemble de données présente les transactions qui ont eu lieu en deux jours, soit 492 fraudes sur 284 807 transactions. L'ensemble de données est très déséquilibré, la classe positive (fraudes) représentant 0,172% de toutes les transactions.
Il est important que les sociétés émettrices de cartes de crédit soient en mesure de reconnaître les transactions frauduleuses par carte de crédit afin que les clients ne soient pas facturés pour des articles qu'ils n'ont pas achetés.

Le jeu de données possède des valeurs continues issues des 28 variables (à savoir V1 à V28). Les variables *Time* et *Amount* ne sont pas des données transformées alors que les variables issues de l'ACP sont normalisées. Il n'y a pas de valeur manquante dans l'ensemble de données.

Notre jeux de données a subi une ACP, donc les variables devraient être linéairement décorélées, nous avons en résultat d'une matrice de corrélation en utilisant Pearson:

- Faible corrélation entre *V7* et *Amount*
- Faible corrélation entre *V20* et *Amount*

Notre jeux de données ne présente pas de problème de multicolinéarité donc nous aurons pas de soucis avec la régression logistique et l'analyse discriminante (linéaire et quadratique) pour la suite.
Ensuite, nous avons remarqué une corrélation assez forte entre la variable *V21*, *V22* suivant Spearman.

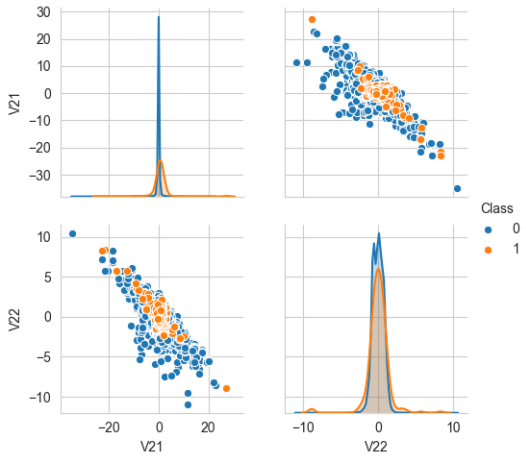
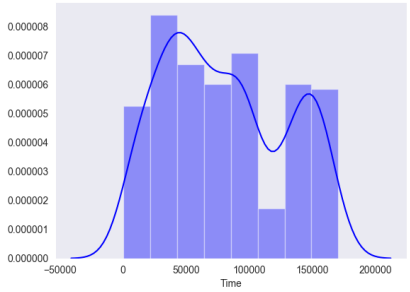
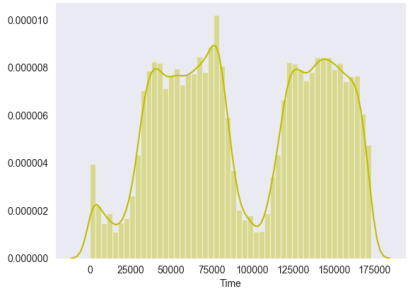


Figure 1: Distribution entre V21 et V22

Nous utilisons un *pairplot* pour comprendre le meilleur ensemble de variables pour expliquer une relation entre deux variables ou pour former les clusters les plus séparés. Cela nous aide également à former des modèles de classement simples en traçant des lignes simples ou à séparer de manière linéaire dans notre jeu de données.
Dans notre cas, les variables approximent une distribution gaussienne et vont en effet dans le même sens.
D'autres variables ont des coefficient Spearman autour de 0.3 mais nous avons décider de ne pas en tenir compte même si cela peut être significatif lorsque nous sommes face à un très grand jeu de données.
Nous allons maintenant nous intéresser plus en détails sur les transactions effectuer au cours des 2 jours.



(a) Distribution des transactions frauduleuses



(b) Distribution des transactions non frauduleuses

Figure 2: Transaction au cours du temps

En regardant le temps par rapport au nombre de transactions, nous pouvons visualiser que les transactions normales suivent un schéma cyclique, contrairement aux transactions frauduleuses. Cela pourrait nous aider pour le modèle de prédiction. Nous nous intéressons maintenant aux montant frauduleux afin de voir si elles se distinguent des autres transactions.

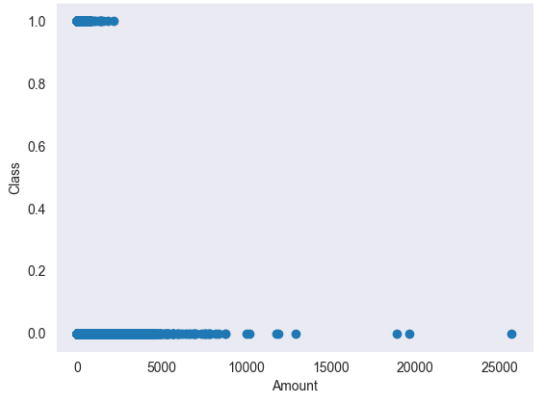


Figure 3: Le montant des transactions suivant leurs classes

La plupart des transactions sont inférieures à 2000 dollars pour les transactions frauduleuses et normales. Nous pouvons observer que les transactions normales sont légèrement plus dispersées que les transactions frauduleuses en termes de montant et que les transactions non frauduleuses sont généralement inférieures à 1000$.

3. Traitements d'un jeux de données non équilibré

L'idée est de comparer si les techniques de prétraitement fonctionnent mieux lorsqu'il existe une classe majoritairement écrasante susceptible de perturber l'efficacité de notre modèle prédictif.

3.1. Échantillonage des données

Le traitement des jeux de données déséquilibrés comprend diverses stratégies, telles que l'amélioration des algorithmes de classification ou l'équilibrage des classes dans les données d'apprentissage (essentiellement une étape de pré traitement des données) avant de fournir les données en entrée du modèle d'apprentissage. Il peut y avoir deux principaux types d'échantillonnage:

- Nous pouvons ajouter des copies d'instances de la classe de minorité appelée sur-échantillonnage (*over-sampling*).
- Nous pouvons supprimer des instances de la classe majoritaire, appelée sous-échantillonnage (*under-sampling*) .

Dans notre cas, nous n'utilisons pas de sous-échantillonnage aléatoire puisque cela peut poser des problèmes d'informations utiles sur les données elles-mêmes, qui pourraient s'avérer nécessaires pour créer des classifier basés sur des règles tels que Random Forest ou le Gradient Boosting.

Ensuite, l'échantillon choisi par sous-échantillonnage aléatoire peut être un échantillon biaisé. Et ce ne sera pas une représentation précise dans notre cas. Le sur-échantillonnage aléatoire peut être une idée envisageable mais il faut savoir doser l'échantillonnage en question afin de ne pas sur-alimenter la classe minoritaire.

Notre idée retenu est de générer des échantillons synthétiques et la plus populaire de ces algorithmes est appelé SMOTE [2].
SMOTE est une méthode de sur-échantillonnage qui crée un exemple synthétique plutôt que de sur-échantillonnage par remplacement. La sur-évaluation de la classe minoritaire est réalisée en prenant chaque échantillon de la classe minoritaire et en introduisant des exemples synthétiques.

Les points positifs est qu'on limite la suralimentation provoquée par le sur-échantillonnage aléatoire car des exemples synthétiques sont générés plutôt

que la réplication d'instances puis il n'y a pas de perte d'information. En revanche, lors de la génération d'exemples synthétiques, SMOTE ne prend pas en compte les exemples voisins pouvant provenir de la classe opposé si celle ci est trop éloigné et SMOTE n'est pas très efficace pour les données en grande dimension.

3.2. Validation croisée

La validation croisée implique la division aléatoire de l'ensemble des observations en k groupes de taille approximativement égale. Chaque partie est traité comme un ensemble de validation mais le sur-échantillonnage de la classe minorité peut entraîner des problèmes de sur-ajustement (*over-fitting*) avant la validation croisée, cependant, il convient de garder à l'esprit que si le sur-échantillonnage avec SMOTE améliore les limites de décision (il permet par la suite à un modèle de mieux généraliser le problème), il n'a toutefois rien à voir avec la validation croisée. Tout comme la sélection de variables doit s'effectuer lors de la boucle de la validation croisée, le sur-échantillonage doit lui même être effectuer aussi.

3.3. Métriques

Notre jeux de données étant très déséquilibré, nous sommes face à un *accuracy paradox* lorsque nous lancerons nos modèles de classification. Nous pouvons commettre deux types d'erreurs dans notre cas:

- Faux Positif : Prédire une transaction frauduleuse alors qu'il n'y en a pas.
- Faux Négatif : Ne pas prédire une transaction frauduleuse alors qu'il y en a eu une.

Les courbes de précision-rappel doivent être utilisées en cas de déséquilibre de classe puisqu'il s'agit d'une représentation du tracé de la précision (axe des ordonnées) et du rappel (axe des abscisses) et ainsi, les courbes de précision-rappel résument le compromis entre le taux positif réel et la valeur en sortie positive pour nos différents modèles.
Pour évaluer cette métrique, nous utiliserons AUC-PR (Air sous la courbe précision-rappel) car une zone haute sous la courbe représente à la fois un rappel élevé et une précision élevée, et donc une précision élevée correspond à un faible taux de faux positifs et un rappel élevé correspond à un faible taux de faux négatifs.

4. Pré-traitement

4.1. Sélection de variables

Pour prendre en compte les meilleurs variables, nous utilisons la librairie LightGBM [4].
LightGBM développe l'arborescence en profondeur, tandis que les autres algorithme élève en largeur les arbres, ce qui signifie que LightGBM élève l'arborescence au niveau de la feuille, tandis que les autres algorithmes se développent en vertical. La particularité de cette méthode et qu'il n'est pas conseillé d'utiliser LGBM sur de petits jeux de données car il persiste des problèmes d'over-fitting. Il n'existe aucun seuil sur le nombre de lignes, mais dans la pratique, il est recommandé d'utiliser cette méthode sur des jeux de données contenant plus de 10 000 lignes.

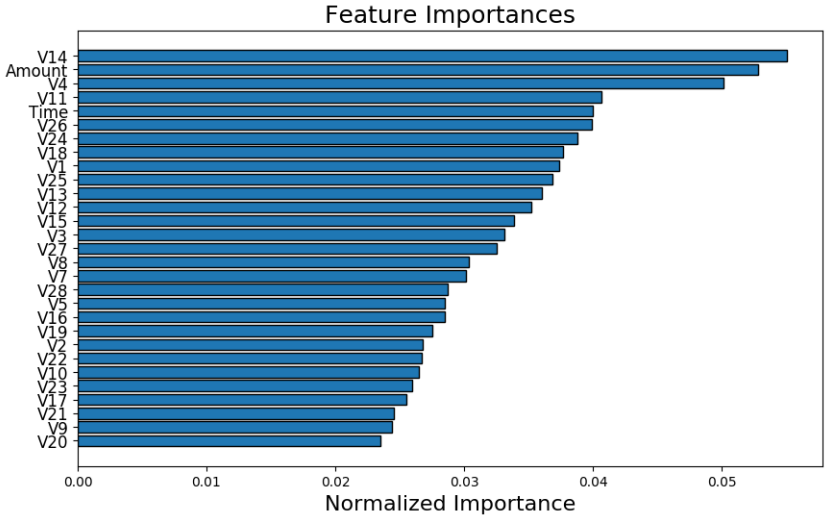


Figure 4: Degré d'importances de chaque variables

27 variables cumulant à eux même 95% de valeurs importante.
Dans notre cas il n'y pas de variables écrasante en terme d'importance, *V14*, *Amount* et *V4* sont dans les 3 premiers mais nous avons que seulement *V7* et *V20* étaient faiblement corréler avec *Amount*. Nous prendrons en compte chaque variable sauf *Amount* lors de l'apprentissage de chaque modèles car même en effectuant un *RobustScaler* sur cette colonne, l'échelle de cette variable et les outliners désavantage les modèles.

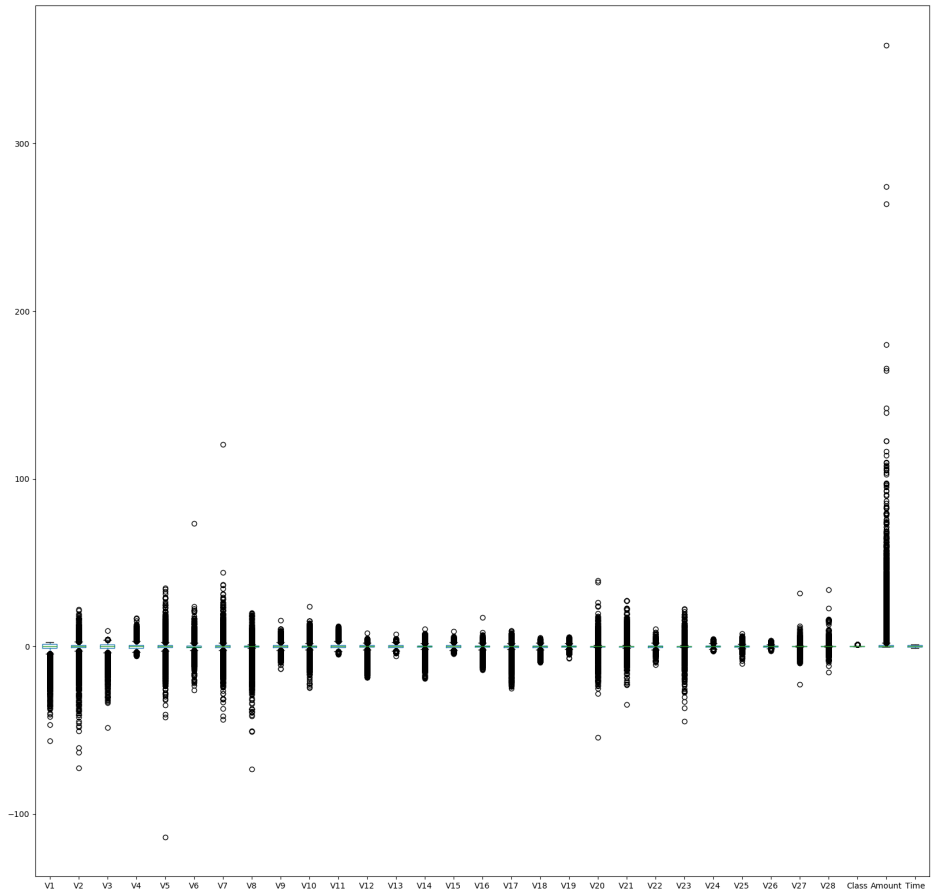


Figure 5: Boxplot des variables

Ensuite comme on a pu le voir dans l'analyse exploratoire des données, les montant des fraudes se mélangent aux autres montant sans réel importance.

5. Modèle d'apprentissage supervisé

5.1. Classification naïve bayésienne

Le classifieur naïf bayésien est l'une des méthodes les plus simples en apprentissage supervisé et il est basé sur le théorème de Bayes. Cet algorithme s'appelle «naïf» car il suppose naïvement que chaque caractéristique est indépendante des autres, ce qui est faux dans la vie réelle.
Ensuite le théorème de Bayes nous aide à trouver la probabilité d'une hypothèse compte tenu de nos connaissances antérieures.

Dans notre cas, nous utilisons l'implémentation *GaussianNB* de scikit-learn [5] (Il est utilisé dans la classification et suppose que les entités suivent une distribution normale.)

5.2. Régression Logistique

Le modèle de régression logistique prend des entrées à valeur réelle et permet de prédire la probabilité que l'entrée appartienne à une classe. Ainsi si la probabilité est supérieur à 0.5, nous pouvons prendre le résultat comme une prédiction pour la classe transaction frauduleuse, et sinon la prédiction est pour l'autre classe (transaction non frauduleuse).
La régression logistique est un algorithme de classification linéaire simple et puissant et cette méthode est destinée aux problèmes de classification à deux classes. Il peut être étendu pour la classification multi-classes, mais est rarement utilisé à cette fin.
Il faut savoir que cette méthode est sensible aux bruits, c'est à dire des valeurs aberrantes présents dans notre jeux de données ainsi que les variables fortement corrélées.
Nous utilisons l'implémentation dans scikit-learn, *LogisticRegression* avec plusieurs paramètres que l'on fait varier tout en utilisant le même solveur *liblinear*:

- *penalty* : La régularisation suivant les deux fonctions coût à minimiser (*l1* et *l2*).
- C : [0.1, 0.001, 0.001, 1, 10, 20, 100]
- *max_ iter* : 100 et 200

Le paramètre de compromis de la régression logistique qui détermine la force de la régularisation est appelé C et la maximum d'itérations est augmenté passant de 100 à 500 et nous avons aussi ajouter le fait de tenir en compte ou non le poids des classes.
C'est au total 56 modèles par fold.

5.3. Analyse Linéaire et Quadratique Dicriminante

L'analyse discriminante linéaire (ADL) recherche la projection qui minimise la variance intra-classe de cet ensemble de données projeté ainsi que de maximiser la variance inter-classe. Et de plus, l'analyse discriminante linéaire a pour objectif de projeter les points situées dans un espace de dimension

supérieure sur un espace de dimension inférieure.
Ensuite l'analyse discriminante linéaire suppose que chaque variable d'entrée a la même variance alors que l'analyse discriminante quadratique (QDA), suppose que chaque classe utilise sa propre estimation de la variance.
L'analyse discriminante quadratique possède une frontière de discrimination en forme de parabole et l'ADL est tout simplement une droite. Finalement, il est presque toujours judicieux de normaliser ses données avant d'utiliser l'ADL afin qu'elles aient une moyenne de 0 et un écart type de 1. En ce qui concerne les hyperparamètres lors de l'utilisation des fonctions implémenter sur scikit-learn, nous n'avons modifier aucun paramètre.

5.4. Machine à vecteurs de support

La SVM consiste à projeter d'abord les données sur un espace de dimensions plus grandes, où nous pouvons encapsuler les données normales dans une hypersphère, même si aucune hypersphère ne peut capturer toutes les données de l'espace d'origine.
Nous avons effectuer plusieurs test suivant un noyau *rbf* (Radial basis function kernel), *linéaire* et *polynomiale*.
La SVM étant très coûteuse nous avons réaliser les calculs sur GPU en utilisant la librairie *thundersvm* [6]. Après plusieurs test avec plusieurs paramètres, nous avons décider de garder le noyau polynomiale et de lancer un *GridSearch* en fesant varier le degré (2,3,4,5) et gamma (1/nombres de variables, 15, 25).
Ce sont 24 modèles de SVM par fold qui sont entrainés.

5.5. Méthode des k plus proches voisins

KNN est un algorithme non paramétrique (c'est-à-dire qu'il ne fait aucune hypothèse sous-jacente concernant la distribution des données). Cette méthode est coûteuse en calcul, car l'algorithme stocke toutes les données d'apprentissage donc une grande mémoire élevée est élevée. Ensuite dans le cas des prédiction pour une utilisation future peut être lente (avec un grand N). De plus, cette méthode est sensible aux valeurs aberrantes et à l'échelle des variables.
Dans notre cas nous n'avons pas modifier le nombre k voisins proches qui est de 5 par défaut, mais nous avons la métrique en fixant la puissance de la métrique de Minkowski à 1 et 2 (ce qui revient à la distance de Manhattan et Euclidienne).
C'est un total de 4 modèles par fold.

5.6. Arbre de décision

Un arbre de décision est un arbre dans lequel chaque nœud représente une caractéristique, chaque branche représente une décision et chaque feuille représente un résultat (catégoriel ou continue). L'idée est de créer un tel arbre pour l'ensemble des données et de traiter un résultat unique à chaque feuille (ou de minimiser l'erreur dans chaque feuille).
Nous utilisons l'algorithme *CART* et notre métrique est l'indice de Gini.
Un score de Gini donne une idée de la qualité d'une scission par la mixité des classes dans les deux groupes créés par la scission. Une séparation parfaite donne un score de Gini de 0, alors que le pire des cas se divise en 50/50 pour les 2 classes.

5.7. Méthodes ensemblistes

Il existe trois termes principaux décrivant les modèles ensemblistes de différents modèles en un modèle plus efficace:

- "*Bagging*" pour diminuer la variance du modèle.
- "*Boosting*" réduire le biais du modèle.
- "*Stacking*" d'augmenter la force prédictive du classifieur.

5.7.1. Random Forest

Les forêts aléatoires [1] sont des modèles puissant et précis puis ils sont performant sur de nombreux problèmes non-linéaire. En revanche, ils ne sont pas facilement interprétables et le nombre d'arbres choisis est importants car il peut y avoir des problèmes d'over-fitting.
L'objectif des forêts aléatoires est de prendre un ensemble d'arbres de décision à forte variance et faible biais et de les transformer en un modèle à faible variance et faible biais. En regroupant les différentes sorties des arbres de décision individuels, les forêts aléatoires réduisent la variance pouvant entraîner des erreurs dans les arbres de décision. Par le vote à la majorité, nous pouvons trouver la production moyenne donnée par la plupart des arbres individuels.

Le but est de réduire la variance de sorte que le modèle ait moins de chances de produire des résultats plus éloignés des valeurs réelles. Ensuite la normalisation des variables n'est pas obligatoire et chaque arbre de décision peut

être entraîner de manière parallèle.
Nous nous sommes basés sur un package non maintenu sous CUDA depuis 5 ans et l'avons repris afin de le mettre à niveau sous Python 3.6 et aussi l'utilisation de la célèbre librairie *Numba* (plus rapide et disponible pour plusieurs versions de Python que *Parakeet*) ainsi qu'une interface scikit-learn afin d'être utilisable lors d'un GridSearch et pour la compatibilité avec les autres modèles. Le code source est disponible ici.
Nous essayons plusieurs paramètres (10, 20 et 30 arbres), ce qui donne 6 modèles par fold.

5.7.2. Gradient Tree Boosting

Le principe du boosting est que les différents classifieurs sont pondérés de manière à ce qu'à chaque prédiction, les classifieurs ayant prédit correctement auront un poids plus fort que ceux dont la prédiction est incorrecte. Dans le cas du Gradient Tree Boosting, nous contruisons de manière itératif une fôret d'arbes en utilisant l'abre ayant approximé la même erreur que l'ensemble du classifier.
Ensuite on retranche cette arbre de décision et tout en optimisant son poids parmi le vecteur contenant le poids de chaque arbre. La fonction coût utilisée est une descente de gradient.
Nous utilisons la librairie *xgboost* [3] et plusieurs combinaisons:

- "n_estimators" : Le nombre d'arbres (1000, 2000, 3000).
- "learning_rate" : Le pas d'apprentissage de la descente du gradient (0.1 et 0.2).
- "max_depth" : Profondeur maximale d'un arbre (5 et 10).

C'est 24 modèles qui sont entraînées à chaque *fold*, la librairie utilise une implémentation sous CUDA.

6. Résultats

Voici les résultats du *GridSearchCV* (mis à 2 découpage puisque nous effectuons déja 5 découpage principaux en incluant un nouveau SMOTE pour chaque fold comme cela est détaillé dans la fonction *runModel* dans le notebook).

Modèle	Moyenne	Ecart-type
SVCPolynomial	50.55	7.9
LogisticRegression	48.89	2.43
RandomForestClassifier	84.51	1.96
XGBClassifier	82.63	1.85
DecisionTreeClassifier	48.37	3.79
LinearDiscriminantAnalysis	46.78	3.69
QuadraticDiscriminantAnalysis	47.31	1.48
GaussianNB	46.10	2.19
KNeighborsClassifier	66.74	2.28

Les deux grands vainqueurs sont *RandomForestClassifier* et *XGBoost-Classifier* en ayant un faible écart-type sur les prédictions. En revanche la SVM souffre d'un résultat qui peut être améliorable en précision-rappel et diminuer aussi son écart-type car nous avons choisi un nombre maximal d'itération faible afin de diminuer drastiquement la compléxité temporel du *GridSearch*.
KNN nous donne des résultats assez intéressant mais cette méthode sur le long terme peut poser problème en terme de complexité lorsque nous voudrons réaliser de future prédiction. En ce qui concerne les autres modèles le résultat de l'air sous la courbe du précision-rappel étant inférieur à 0.5, nous pouvons considérer cela comme des modèles avec une capacité de prédiction aléatoire n'ayant pas capturer la complexité du problème.

Le classificateur de vote (*VotingClassifier*) permet de combiner différents classifieurs sur les étiquettes de classe prédites pour un exemple avec un système de vote majoritaire.
Avec un vote *hard*, il suffit d'une majorité de classificateurs pour déterminer le résultat. Il ne prend pas en compte les probabilités de chaque classes pour chaques classificateurs, mais juste la classe prédites.
En revanche avec *soft*, nous calculons un poids (pourcentage) pour chaque classificateur. Une probabilité de classe prédite de chaque modèle pour chaque exemples est collectée et multipliée par le poids du classifieur, puis moyennée. La SVM permettant de récupérer des probabilités de prédicition sous scikit-learn, nous nous sommes orienté avec un vote *soft* et nous obtenons *VoteClassifier* une moyenne de 83.47% (+/- 1.44) et avec un StackingClassifier le résultat est de 84.86% (+/-1.77).

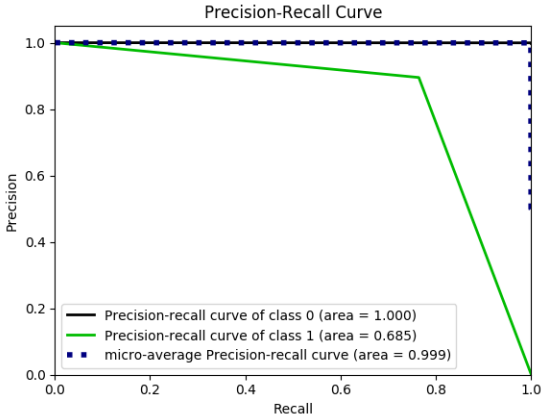
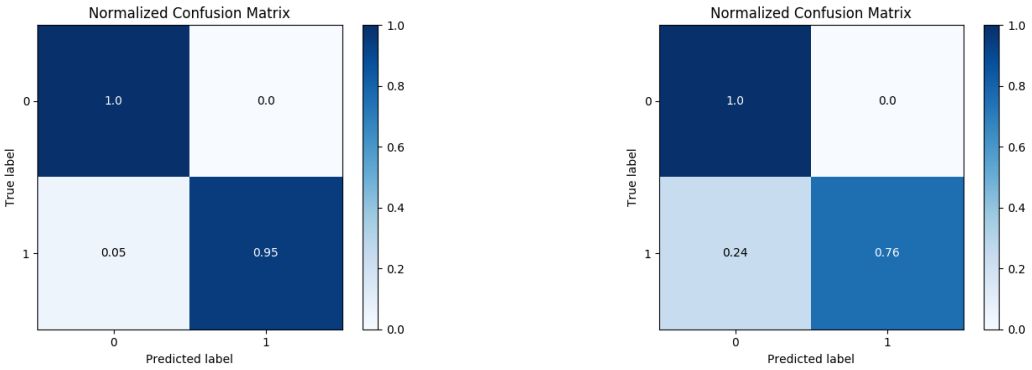


Figure 6: AUC-PR sur le StackingClassifier

Nous n'avons pas un gain très significatif dans notre cas, la courbe de AUC-PR est affiché ci-dessus et nous constatons qu'il y a un problème avec la détection de fraude sans grande surprise.



(a) Résultat sur le jeux d'entraînement

(b) Résultat sur le jeux de test

Figure 7: Matrice de confusion

Le modèle sur les données d'apprentissages et de test ne commet pas de faux négatif mais il possède une difficulté avec les faux positifs.

7. Conclusion

Pour conclure, nous nous sommes beaucoup intéressés aux méthodes ensemblistes dans notre approches de ce problème afin de faire preuve d'originalité. Les méthodes d'ensembles à base d'arbre de décision sont intéressantes car l'architecture des modèles réduit l'over-fitting, mais des pratiques d'échantillonnement médiocres peuvent néanmoins conduire à de fausses conclusions sur la qualité d'un modèle.
Le but principal de la validation du modèle consiste à estimer la manière dont le modèle généralisera à de nouvelles données. Si la décision de mettre un modèle en production dépend de la manière dont il fonctionne sur un ensemble de validation, il est essentiel que le sur-échantillonnage soit effectué correctement.
En effet, en sur-échantillonnant uniquement les données d'apprentissage, aucune des informations contenues dans les données de validation n'est utilisée pour créer des observations synthétiques donc, ces résultats devraient être généralisables.
Ensuite le suréchantillonnage est un moyen bien connu d'améliorer potentiellement les modèles formés sur des données déséquilibrées mais il est important de se rappeler qu'un suréchantillonnage incorrect peut conduire à penser qu'un modèle généralisera mieux qu'il ne le fait réellement.
Nos résultat peuvent sembler faible par rapport à ce qui se propose sur internet mais il convient d'utiliser correctement la validation croisée avec une bonne méthode d'échantillonnement ainsi qu'une métrique approprié lorsque nos données sont déséquiblibrées.

References

[1] Breiman, L., Oct. 2001. Random forests. Mach. Learn. 45 (1), 5–32.
URL https://doi.org/10.1023/A:1010933404324

[2] Chawla, N. V., Bowyer, K. W., Hall, L. O., Kegelmeyer, W. P., Jun. 2002. Smote: Synthetic minority over-sampling technique. J. Artif. Int. Res. 16 (1), 321–357.
URL http://dl.acm.org/citation.cfm?id=1622407.1622416

[3] Chen, T., Guestrin, C., 2016. Xgboost: A scalable tree boosting system. CoRR abs/1603.02754.
URL http://arxiv.org/abs/1603.02754

[4] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, T.-Y., 2017. Lightgbm: A highly efficient gradient boosting decision tree. In: Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (Eds.), Advances in Neural Information Processing Systems 30. Curran Associates, Inc., pp. 3146–3154.
URL http://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boos
pdf

[5] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., 2011. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12, 2825–2830.

[6] Wen, Z., Shi, J., Li, Q., He, B., Chen, J., 2018. ThunderSVM: A fast SVM library on GPUs and CPUs. Journal of Machine Learning Research 19, 1–5.