DÉTECTION DE FRAUDES

Amina Ghoul
Yamina Boubekeur
Anthony Davidas Roch
Mbaimou Auxence Ngremmadji

Table des matières

1. Introduction

La détection des fraudes est l'une des problématiques les plus courantes et sensibles dans de nombreux secteurs, en particulier le secteur bancaire. Au cours des dernières années, les tentatives de fraude ont connu une forte hausse, ce qui rend la lutte contre ce phénomène très importante. Le système de prévention de la fraude actuel permet aux consommateurs d'économiser des millions de dollars par an. Les chercheurs de l'IEEE Computational Intelligence Society (IEEE-CIS) souhaitent améliorer ce chiffre tout en améliorant l'expérience client dans le but d'aider des centaines de milliers d'entreprises à réduire leurs pertes de fraude et à augmenter leurs revenus.

L'objectif de ce projet est de prédire la probabilité qu'une transaction soit frauduleuse en s'aidant de la base des données fournies par l'entreprise Vesta.

Pour ce faire, nous utiliserons les algorithmes d'apprentissage automatique suivants : Naïve Bayes, k Nearest Neighbors (k-NN), la Régression Logistique, Random Forest et Adaptative Boosting.

2. Exploration des données

Les données proviennent des transactions de commerce électronique de Vesta qui est le précurseur des solutions de paiement e-commerce garanties.

Il s'agit d'un problème de classification binaire, c'est-à-dire que notre variable cible 'isFraud' prend deux valeurs possibles. La question qui se pose est alors la suivante : l'utilisateur fait-il une transaction frauduleuse ou non ?

Les données d'entrainement et de test sont divisées en deux fichiers :

- identity : contient des informations sur les utilisateurs.
- transaction : contient les transferts d'argent et également d'autres produits et services de cadeaux.

Les deux fichiers sont reliés par l'attribut 'TransactionID'.

Nous avons 4 fichiers à notre disposition :

- sample_submission.csv - un exemple de fichier de soumission
- train_identity.csv - données d'entrainement d'identité
- test_identity.csv - données de test d'identité
- train_transaction.csv - données d'entrainement de transaction
- test_transaction.csv - données de test de transaction

- Taille du dataset :
  - Données d'entrainement : 590540 observations et 434 variables qualitatives et quantitatives.
  - Données de test : 506691 observations et 433 variables qualitatives et quantitatives.

Nous avons quelques informations sur certaines variables :

- Données qualitatives discrètes :
  - ProductCD : code produit pour chaque transaction
  - card1-card6 : Informations sur la carte de paiement
  - P_emaildomain, R_emaildomain, domaine de messagerie de l'acheteur et du destinataire
  - addr1 : région de facturation, addr2 : pays de facturation
  - M1-M9 : Correspondance, comme les noms sur la carte et l'adresse, etc.
  - id_12, id_15, id_16, id_23, id_27, id_28, id_29, id_30, id_31, id_33, id_34, id_35, id_36, id_37, id_38
  - DeviceType, DeviceInfo

Le reste des variables sont des variables numériques :

— TransactionDT : timedelta à partir d'une datetime de référence donnée (pas un horodatage réel)
— TransactionAMT : le montant de la transaction en USD
— dist : distance entre (sans limitation) l'adresse de facturation, l'adresse Nous avons quelques informations sur certaines variables
— TransactionDT : timedelta à partir d'une datetime de référence donnée (pas un horodatage réel)
— TransactionAMT : le montant de la transaction en USD
— C1-C14 : comptage, comme le nombre d'adresses associées à la carte de paiement, etc. La signification réelle est masquée
— D1-D15 : timedelta, comme les jours entre la transaction précédente, etc.
— Vxxx : Vesta a conçu de riches variables, notamment le classement, le comptage et d'autres relations d'entité
— id01-id11 : variables numériques pour l'identité
— IsFraud = 1 signifie qu'une transaction est frauduleuse et IsFraud = 0 désigne une transaction non-frauduleuse.

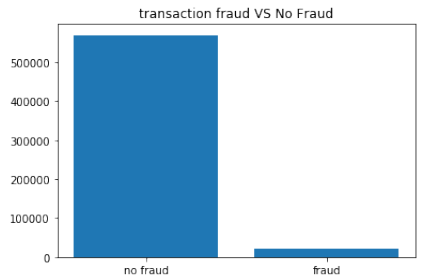
2.1 Visualisation des données

	TransactionID	isFraud	TransactionDT	TransactionAmt	ProductCD	card1	card2	card3	card4	card5	...	id_31	id_32	id_33	id_34	id_35	id_36	id_37	id_38	DeviceType	DeviceInfo
0	2987000	0	86400	68.5	W	13926	NaN	150.0	discover	142.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	2987001	0	86401	29.0	W	2755	404.0	150.0	mastercard	102.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	2987002	0	86469	59.0	W	4663	490.0	150.0	visa	166.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	2987003	0	86499	50.0	W	18132	567.0	150.0	mastercard	117.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	2987004	0	86506	50.0	H	4497	514.0	150.0	mastercard	102.0	...	samsung browser 6.2	32.0	2220x1080	match_status:2	T	F	T	T	mobile	SAMSUNG SM-G892A Build/NRD90M

Certaines variables ne sont pas très compréhensibles. De plus, on a énormément des valeurs inconnues (NaN). Cela est dû au fait que non pas toutes les transactions ont une identité correspondante. Voyons cela de plus près.

On va afficher les distributions de certaines variables.

- IsFraud (variable cible)

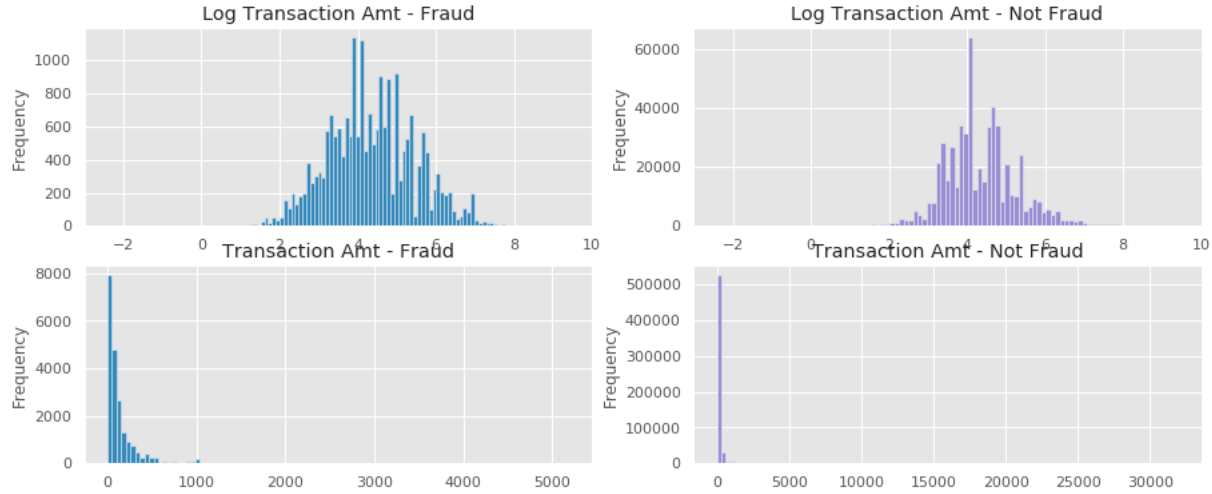


Nous pouvons voir clairement que la plupart des transactions sont non frauduleuses. Si nous utilisons cette base de données comme base pour nos analyses et nos modèles prédictifs, nous pourrons obtenir beaucoup d'erreurs et nos algorithmes seront probablement trop adaptés car ils "supposeront" que la plupart des transactions ne sont pas de la fraude.

Nous ne voulons pas que notre modèle suppose, mais qu'il apprenne si une transaction est frauduleuse.

- Transaction Amt

Cette variable décrit le montant de la transaction.



Nous avons représenté la distribution du log de la variable "Transaction Amt" pour les transactions frauduleuses et pour les transactions non frauduleuses. Ces distributions ont l'allure d'une loi normale et ont des variances comparables. Nous observons que la moyenne μ_1 du montant des transactions dans le groupe des transactions frauduleuses est différente de la moyenne μ_2 dans le cas des transactions non frauduleuses.

Nous avons effectué alors un test de student pour vérifier cette observation.

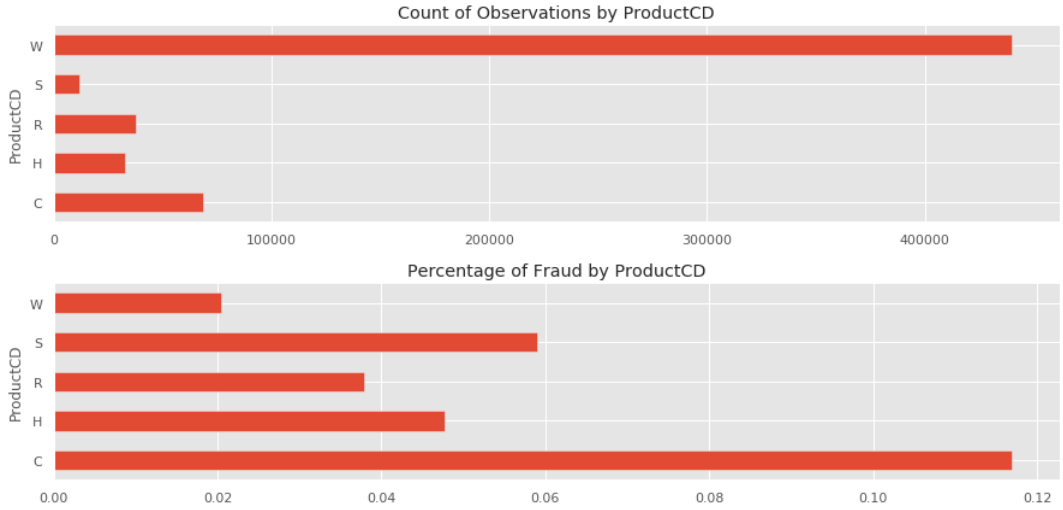
Nous testons alors : $H_0 : \mu_1 = \mu_2$ contre $H_1 : \mu_1 \neq \mu_2$

Nous obtenons la p-value suivante : $p - value = 3 \times 10^{-19} < 0.05$.

Donc on rejette H_0, c'est-à-dire que les moyennes des deux distributions ne sont effectivement pas égales.

- ProductCD

Cette variable décrit le code produit pour chaque transaction.



On a observé ce qui suit :

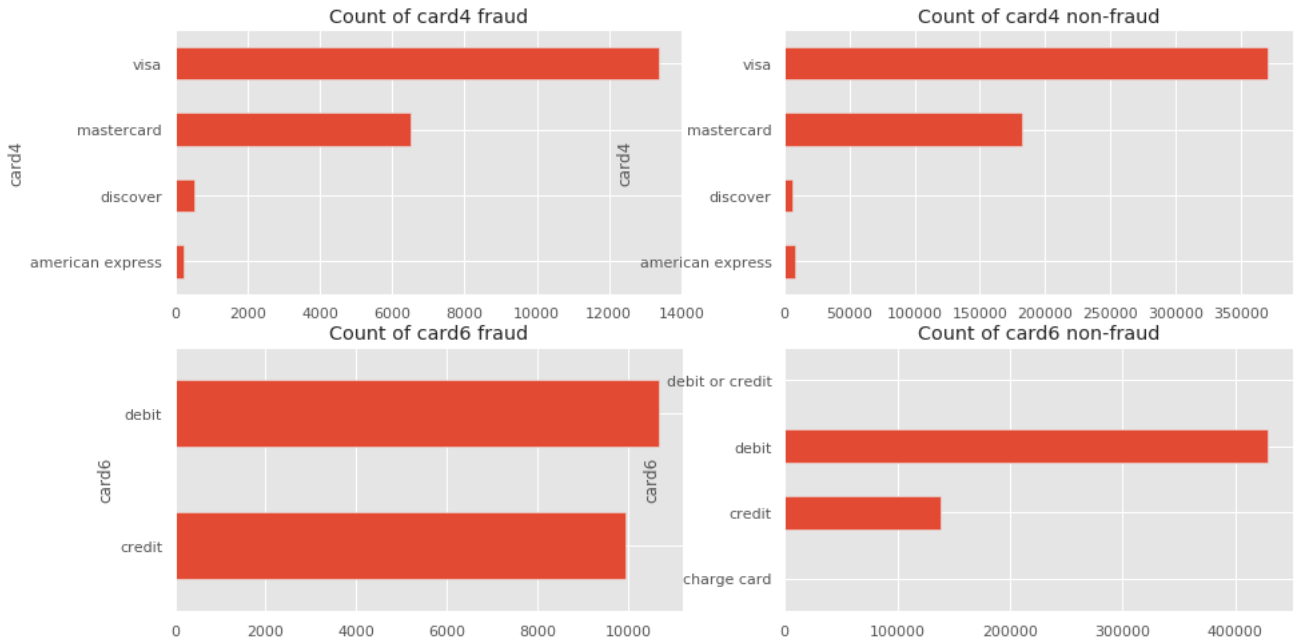
- W a le plus grand nombre d'observations, S a le moins.
- C a le plus grand pourcentage de fraude $> 11\%$, W a le moins avec environ 2%

- card 1 - card 6

Ces variables nous donnent des informations sur les cartes de paiement.

	card1	card2	card3	card4	card5	card6
0	13926	NaN	150.0	discover	142.0	credit
1	2755	404.0	150.0	mastercard	102.0	credit
2	4663	490.0	150.0	visa	166.0	debit
3	18132	567.0	150.0	mastercard	117.0	debit
4	4497	514.0	150.0	mastercard	102.0	credit

Les variables card4 et card6 sont des chaînes de caractères et ce sont les seules variables compréhensibles.

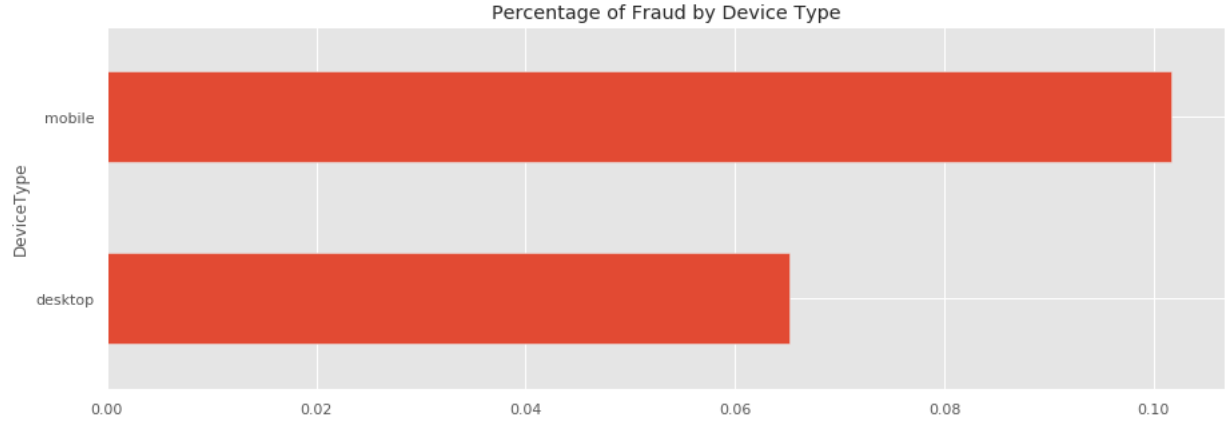


Nous avons observé que dans le cas des fraudes et des non-fraudes, les cartes visa sont les plus utilisées.

De plus, dans le cas des fraudes, le nombre de cartes de débit et de crédit sont à peu près les mêmes contrairement au cas des non-fraudes où le nombre de cartes de débit est bien supérieur au nombre de cartes de crédit.

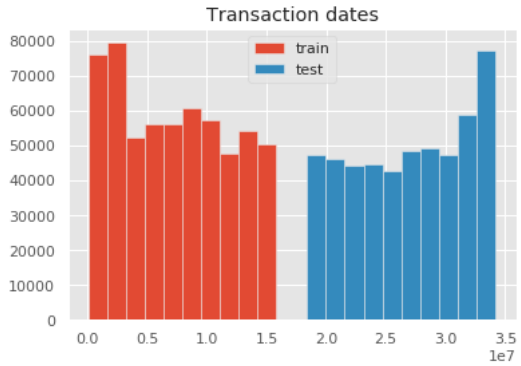
- DeviceType

Cette variable indique si la transaction a été réalisée sur mobile ou sur ordinateur.



Nous avons observé que le pourcentage de fraude est plus grand sur les mobiles que sur les ordinateurs.

- TransactionDT



Ci-dessus, on voit que les dates des données Train et Test ont une intersection vide.

2.2 Les valeurs manquantes

Comme mentionné précédemment, les données contiennent beaucoup de valeurs manquantes. On doit alors les gérer.

```
TransactionID        0
isFraud              0
TransactionDT        0
TransactionAmt       0
ProductCD            0
card1                0
card2             8933
card3             1565
card4             1577
card5             4259
dtype: int64
% of missing data =  45.074371905803936
```

On peut voir que 45% des données du train (entrainement) sont des valeurs manquantes.

	Missing Ratio
id_24	99.196159
id_25	99.130965
id_07	99.127070
id_08	99.127070
id_21	99.126393

Ce tableau indique par exemple que 99.20% des valeurs de id_24 sont des valeurs manquantes. Ce qui est énorme.

On a donc décidé de supprimer les variables contenant plus de 80% de valeurs manquantes (74 variables supprimées).

2.3 Les chaînes de caractères

La deuxième étape du pré-traitement des données, est la gestion des chaînes de caractères.

Pour cela, nous avons utilisé l'approche one-hot encoding.

Cette méthode fonctionne de la manière suivante : Pour chaque variable qualitative X (chaine de caractère) ayant k valeurs différentes ($i = 1, ..., k$), nous créons k colonnes contenant 0 et 1 notées X_i, $i = 1, ..., k$. La présence de 1 dans une nouvelle colonne X_i signifie que X prend la valeur i dans la variable initiale.

Par exemple, la variable 'card4' peut prendre les valeurs visa, american express, discover ou mastercard. Ainsi, avec l'encodage one-hot, quatres colonnes card4_visa,card4_americanexpress, card4_discover et card4_mastercard sont créees où si la transaction à été faite par la carte visa par exemple, la valeur de la variable card4_visa est de 1 et celle des autres variables est 0 comme représenté sur la photo ci-dessous :

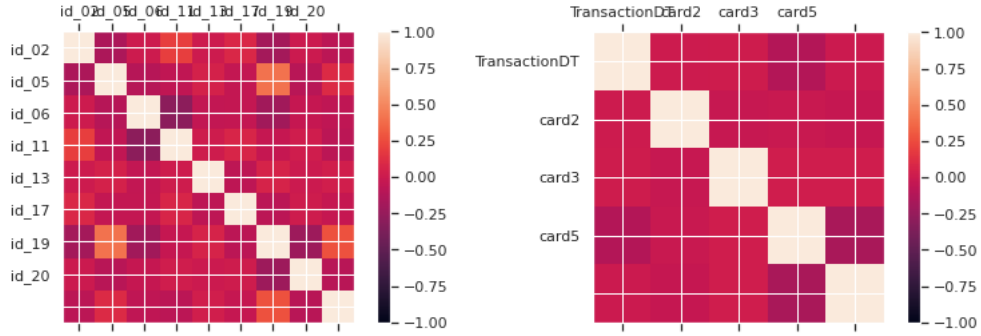
card4_american express	card4_discover	card4_mastercard	card4_visa	card4
0	1	0	0	discover
0	0	1	0	mastercard
0	0	0	1	visa
0	0	1	0	mastercard

En effet, nous n'utiliserons pas la méthode de label encoder consistant à remplacer les valeurs des variables qualitatives par des chiffres car cela impliquerait qu'il y aurait des relations de supériorité entre les valeurs qualitatives.

2.4 Les variables corrélées

La dernière étape de pré-traitement des données est la gestion des variables corrélées.

Voic les matrices de corrélations de certaines variables.



On observe très clairement que ces variables ne sont pas corrélées.

Pour faire la sélection des variables, on utilise la régularisation Lasso.

On cherche ainsi à expliquer de manière linéaire une variable Y, (ici Y correspond à la variable IsFraud) par p variables potentiellement explicatives x_i où $1 < i < n$; n est le nombre d'observations. On modélise la variable Y de la manière suivante :

$$Y_i = x_i^T \beta + \epsilon_i$$

où les $\epsilon_i \sim N(0, \sigma^2)$ sont i.i.d. $Y_i \in R$, $x_i \in R^p$, $\beta \in R^p$ est inconnu.

Les variables x_i n'étant pas toutes pertinentes, l'objectif est d'éliminer les variables inutiles et uniquement celles-ci. L'idée du Lasso est donc non pas de faire une régression linéaire classique mais une régression régularisée qui rend nuls certains coeficients de l'estimation de β. Cela consiste à estimer pour $\lambda \in R_+$:

$\hat{\beta}(\lambda) = \underset{\beta \in R^p}{argmin} \frac{1}{2}||Y - X\beta||_2^2 + \lambda||\beta||_1$ Avec $Y \in R^n$, $X \in R^{n \times p}$.

Le paramètre $\lambda \geq 0$ contrôle la puissance de la régularisation. Si on prend $\lambda = 0$, le Lasso correspond à une régression linéaire classique.

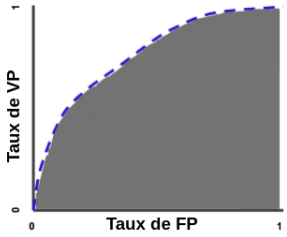
Pour $\lambda = 1$ et $\lambda = 0.1$, on a remarqué que tous les β sont nuls. Donc nous avons pas de variables selectionnées. En diminuant la valeur de lambda il est possible que certains β deviennent non nulles. Toutefois, ayant déjà considérablement diminué notre nombre de variables, nous avons décidé de travailler avec les variables restantes pour les parties suivantes.

3. Prédiction de la probabilité du nombre de fraudes

Une fois les données pré-traitées, on peut alors appliquer différents algorithmes de machine learning afin de prédire la probabilité qu'une transaction en ligne soit frauduleuse.

On va appliquer différents algorithmes : Naive Bayes, Knn, la Régression Logistique, Random Forest, Adaptive Boosting . Afin de pouvoir juger de l'efficacité des modèles proposés, on utilise l'AUC qui correspond à l'aire sous la courbe ROC.

Une courbe ROC (receiver operating characteristic) est un graphique représentant les performances d'un modèle de classification pour tous les seuils de classification. Cette courbe trace le taux de vrais positifs en fonction du taux de faux positifs.



3.1 Naive Bayes

Dans un premier temps, nous avons appliqué l'algorithme Naive Bayes. Son utilisation ne nécessite pas l'ajustement des hyperparamètres.

La classification naïve bayésienne est un type de classification bayésienne probabiliste simple basée sur le théorème de Bayes avec une hypothèse d'indépendance des variables explicatives.

- Avantages :

- Il est relativement simple à comprendre et à construire.
- Il est facile à former, même avec un petit jeu de données.
- Il est rapide.
- Il n'est pas sensible aux caractéristiques non pertinentes.

- Inconvénients :

— Il suppose une indépendance des variables, ce qui n'est pas toujours le cas.

C'est un algorithme qui permet de prédire la probabilité qu'un événement se produise en fonction des conditions que nous connaissons pour les événements en question.

L'AUC renvoyé par le modèle est de 0.58. Ce résultat est loin d'être bon.

3.2 Méthode des k plus proches voisins

Pour effectuer une prédiction, l'algorithme K-NN va se baser sur tout le jeu des données. En effet, pour une nouvelle observation, on souhaite prédire la classe où appartient cette dernière. L'algorithme va chercher les K instances du jeu de données les plus proches de notre observation selon une métrique bien définie.

Il faut donc déterminer la valeur de K optimale avant d'utiliser cet algorithme.

Pour ce faire, on utilise la méthode GridSearchCV qui prend en paramètre l'estimateur knn et une liste contenant des valeurs de K. Cela retourne la meilleure valeur de K, celle qui maximise le score obtenu en faisant une 5-fold cross validation.

Dans notre cas, nous avons trouvé une valeur de K optimale égale à 2.

En utilisant cette valeur de K, nous avons obtenu un AUC de 0.63.

- Avantages :

— Facile à comprendre .
— Adaptable à un problème de classification multi-classe.

- Inconvénients :

— Temps d'exécution énorme pour un jeu de données de grande taille.
— Tendance à surestimer lorsque les variables explications sont nombreuses.

3.3 Régression logistique

On a appliqué par le suite, la régression logistique. Elle décrit la modélisation d'une variable qualitative Y à 2 modalités : 1 ou 0. Dans notre cas Y correspond à IsFraud.

La régression logistique cherche à estimer la probabilité $P(Y|X)$: Y a deux modalités, donc : $\frac{P(Y=1|X)}{P(Y=0|X)} = \frac{P(Y=1)*P(X|Y=1)}{P(Y=0)*P(X|Y=0)}$

La régression logistique consiste à faire l'hypothèse que cette quantité peut être écrite à l'aide d'une fonction linéaire en X et par la suite à maximiser la log-vraisemblance. Toutefois, afin d'éviter un sur-apprentissage ou un problème de convergence aux algorithmes qui approximent les coefficients de régression, on peut pénaliser les grands coefficients. Cela peut se faire en ajoutant un terme de pénalisation à la fonction de log-vraisemblance.
La fonction que nous avons utilisé nous a permis d'utiliser 3 types de pénalisation : la régularisation L1, la régularisarion L2 et elasticnet (qui est une combinaison des deux premières). De plus, la régularisation L1 est aussi utile pour la selection de variable.

- Avantages :

- Bon algorithme pour les classifications binaires.
- Simple à utiliser et n'a pas besoin de beaucoup d'optimisaiton de paramètres.

- Inconvénients :

- Mauvais pour les classifications non linéaire.
- Généralement possible d'obtenir de meilleur résultat avec des algorithmes plus performants.

Sur nos données, nous avons ajouté une régularisation afin d'éviter un sur-apprentissage dû au grand nombre de données. Après cross validation sur différents paramètres, nous avons obtenu que la L2 régularisation est celle qui permet d'avoir le meilleur auc. Cela peut s'expliquer que par le néttoyage de données nous avions déjà suffisament réduit notre nombre de variable. Nous avons optimisé les paramètres en prenant en compte notre base de donnée qui a un problème de classe déséquilibrée. À l'issue de cela, nous avons obtenu un score auc allant jusqu'à 0.827 par cross validation sur nos données.

3.4 Random Forest

Random Forest ou Forêt aléatoire est un modèle composé de nombreux arbres de décision. Plutôt que de simplement faire la moyenne de la prédiction des arbres (qu'on pourrait appeler une «forêt»), ce modèle utilise deux concepts clés qui lui donnent le nom aléatoire :

- Échantillonnage aléatoire des points de données d'entraînement lors de la construction d'arbres.
- Sous-ensembles aléatoires de variables pris en compte lors de la division des noeuds.

- Avantages :

- L'algorithme Random Forest est considéré comme une méthode très précise et robuste en raison du nombre d'arbres de décision participant au processus.
- Cet algorithme classe les variables explicatives en fonction de leur lien avec les variables à expliquer.
- Il ne souffre pas du problème de sur-ajustement. La raison principale est qu'il prend la moyenne de toutes les prédictions, ce qui annule les biais.
- On peut obtenir l'importance relative des variables, ce qui aide à sélectionner les variables les plus contributives pour le classificateur.

- Inconvénients :

- Les forêts aléatoires sont lentes à générer des prédictions car elles ont plusieurs arbres de décision. Chaque fois qu'il fait une prédiction, tous les arbres de la forêt doivent faire une prédiction pour la même entrée donnée, puis effectuer un vote sur celle-ci. Tout ce processus prend du temps.
- Le modèle est difficile à interpréter par rapport à un arbre de décision, où vous pouvez facilement prendre une décision en suivant le chemin dans l'arbre.

- Paramètres utilisés :

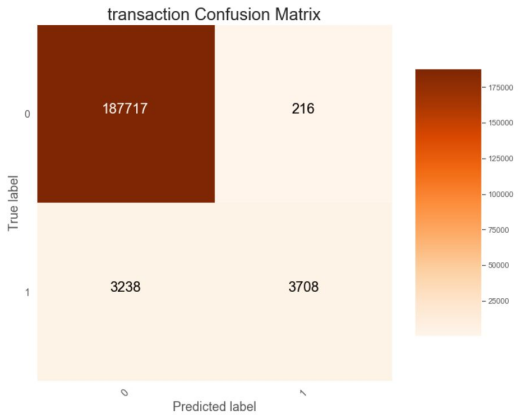
- n_estimators : Nombre d'arbres dans la forêt. La valeur choisie dans notre cas est 40.
- min_samples : nombre minimum de noeud requis pour la division. La valeur choisie est la valeur par défaut qui est de 2.
- max_features : nombre de variables à considérer lors de la recherche de la meilleure répartition. Dans notre cas max_features='sqrt' c'est-à-dire max_features est égale à la racine du nombre de variables utilisées dans notre modèle.
- n_jobs= -1 signifie qu'on utilise tous les processeurs.

Après avoir entrainé le modèle sur l'ensemble de variables et fait des prédictions sur les données de test, On a cherché combien de noeuds y a-t-il en moyenne pour chaque arbre et la profondeur maximale de chaque arbre.

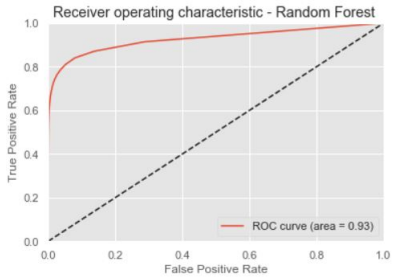
Chaque arbre de décision dans la forêt a de nombreux noeuds (33829) et est extrêmement profond (moyenne de la profondeur maximale 49). Cependant, même si chaque arbre de décision individuel peut s'adapter à un sous-ensemble particulier des données d'entrainement, l'idée est que la forêt aléatoire globale devrait avoir une variance réduite.

- Evaluation des performances :

La matrice de confusion est utile pour nous donner le nombre de faux positifs et de faux négatifs.



Cela montre les classifications prévues par le modèle sur les données de test ainsi que les labels réels. On peut voir que notre modèle a de nombreux faux positifs (une transaction non frauduleuse prédite mais en fait une transaction frauduleuse)



Le modèle atteint toujours des mesures parfaites sur les données d'entraînement, mais cette fois, les résultats des tests sont bien meilleurs. Le score de notre modèle sur la base du score roc_auc,est de 0,93.

La prochaine chose qu'on a fait est de régler nos hyperparamètres pour voir si on peut améliorer les performances du modèle.

Avec l'aide de l'article de William Koehrsen, «Hyperparameter Tuning the Random Forest in Python», on a utilisé la fonction RandomizedSearchCV de la librairie sklearn pour optimiser nos hyperparamètres. Koehrsen utilise une grille complète d'hyperparamètres dans son article, mais on a trouvé que cela pouvait prendre énormement de temps pour s'exécuter dans la pratique. on a décidé de nous concentrer sur 3 hyperparamètres : n_estimators, max_features et max_depth(profondeur maximale de l'arbre). L'agorithme a fait en tout 300 calculs, 3 fits et pour chaque fit 100 estimations. Temps de calcul a duré 9h et les hyperparamètres optimals retournés par l'algorithme sont les suivants : n_estimators = 485, max_features='sqrt',max_depth=164.

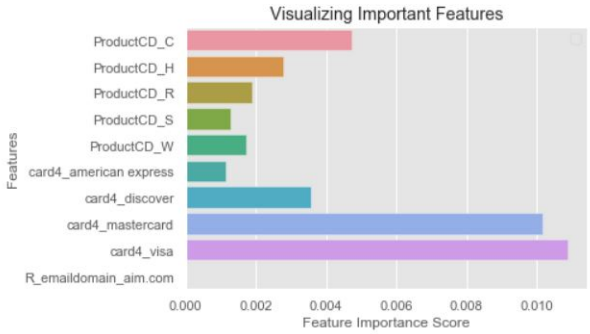
Avant d'appliquer ces nouveaux hyperparamètres dans le modèle pour voir si cela a amélioré nos performances, on cherche les variables considérées comme les plus importantes par notre modèle. Les valeurs sont calculées à l'aide de l'indice de Gini de l'impureté d'un noeud.

On définit l'impureté des noeuds fils, le plus souvent par le biais de l'indice de Gini. L'indice

de Gini d'un noeud t est le suivant :

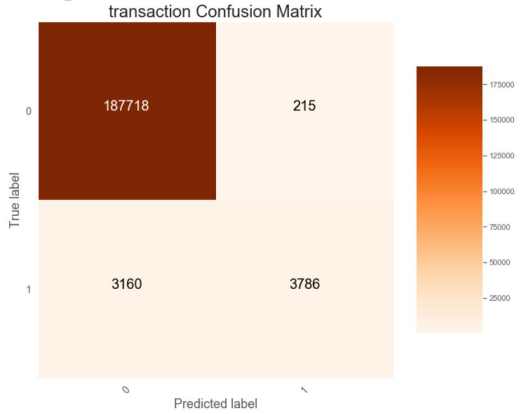
$$\phi(t) = \sum_{c=1}^{L} \hat{p}_t^c (1 - \hat{p}_t^c)$$

où $\hat{p}_t^c$ est la proportion d'observations de classe c dans le noeud t.



En tout le modèle possède 126 variables importantes (ci-dessus sont représentées que les 10 premières).

On a ensuite relancer notre algorithme que sur les variables jugées importantes par notre modèle et les hyperparamètres optimaux.



Le score de notre modèle optimal sur la base du score roc_auc,est de 0,944.

Il y a eu une légère amélioration des résultats. Notre score roc_auc est passé de 0,93 à 0,944. Et aussi, Le nombre de faux positifs faux négatifs ont tous les diminués.

3.5 Adaptative boosting

La méthode de boosting adaptative consiste à appeler de manière repétitive (M fois) un classifieur faible. Considérons les données d'apprentissages suivantes : $(X_i, y_i), i = 1, ..., n$ et $y_i = 0 \quad$ ou $\quad y_i = 1$ (classification binaire). On associe à chaque X_i un poids D_i. Initiallement,

on choisit des poids égaux. A chaque fois qu'on appelle le classifieur faible, on ajuste les poids : ceux des observations mal classifiés augmentent.

A chaque itération k ($k = 1, ..., M$), une fonction $\hat{h}_k : X_i \longrightarrow \{0, 1\}$ est défini. Il s'agit du classifieur faible. La mise à jour des poids D_i associés aux observations dépend de l'erreur de classification du classifieur faible. Après avoir obtenus des classifieurs faibles, la sortie finale de la méthode de boosting adaptative est le signe de la combinaison linaire des classifieurs faibles $\hat{h}_k$ où les coefficients de combinaison linéaire dépendent de l'erreur d'estimation du classifieur faible.

Nous avons choisi d'utiliser cet algorithme car il permet en premier lieu d'éviter le sur-apprentissage. De plus, il donne en général un score acceptable.

Comme classifieur faible, nous avons utilié l'arbre de décision avec un seul noeud . Les hyper-paramètres obtenus après optimisation sont les suivants : nestimator = 50 et learningrate = 1 .

Rappelons que nestimator désigne le nombre de fois où on entraine de manière repétitive l'arbre de décision (50 fois dans notre cas). learningrate un paramètre qui exprime la contribution de chaque modèle entrainé itérativement aux poids D_i.

- Avantages :

- La méthode de boosting adaptative permet dans la plus part des cas d'éviter le sur-apprentissage.
- Il s'agit d'un algorithme qui adapte à chaque fois les poids associés à chaque observation. Cela permet de mieux gérer les labels mal classifiés.
- C'est un algorithme facile à implémenter. De plus, on le combine avec d'autres algorithmes de machine learning.

- Inconvenients :

- La méthode de boosting adaptative est sensible aux données bruitées.
- C'est un algorithme qui est également sensible aux valeurs aberrantes.

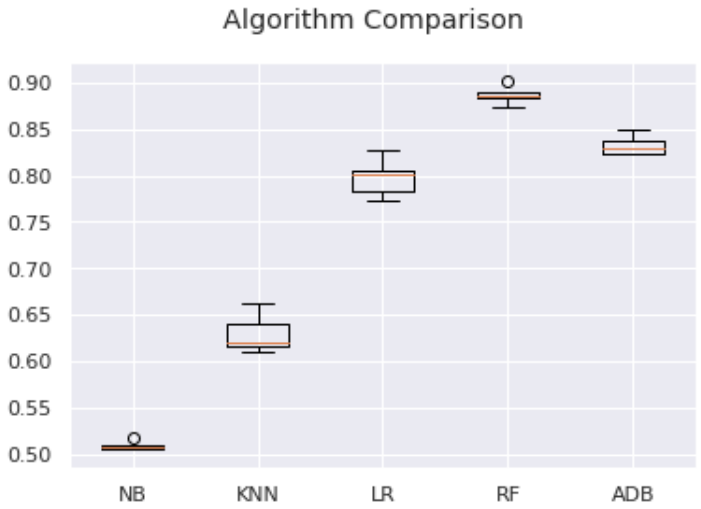
Dans le cadre de ce travail, l'AUC obtenu est 0.83.

4. Comparaison des algorithmes d'apprentissages

Nous avons évalué chaque algorithme sur les mêmes données.

La procédure de validation croisée de 5-folds est utilisée pour évaluer chaque algorithme, configuré de manière importante avec la même graine aléatoire pour garantir que les mêmes répartitions aux données d'entrainement sont effectuées et que chaque algorithme est évalué précisément de la même manière.

Ci-dessous est représenté le boxplot de la répartition des scores de précision à travers chaque pli de validation croisée pour chaque algorithme.



D'après ces résultats, on voit que les algorithmes Random Forest et Adaptive Boosting sont ceux qui renvoient les meilleurs prédictions.

5. Conclusion

En conclusion, ce projet nous a permis d'étudier un grand jeu de données réelles (non académiques).

Nous avons passé une grande partie du temps à pré-traiter ces données brutes. Ensuite, nous avons appliqué un certain nombre d'algorithme de machine learning afin de comparer les scores obtenus (auc score). Nous avons sélectionné Naïve Bayes, KNN, Régression Logistique, Random Forest et Adaptive Boosting.

Random Forest et Adaptive Boosting donnent des meilleurs résultats parmi les cinqs algorithmes proposés (AUC de 0,94 et 0,84 respectivement). En prenant en compte le temps d'exécution de ces deux algorithmes, adaptative boosting s'exécute en peu de temps comparé au Random Forest.

En guise de perspective, nous suggérons d'analyser davantage ces deux algorithmes afin d'augmenter les scores.

Bibliographie

[1] Cours - forêts aléatoires. http://cedric.cnam.fr/vertigo/Cours/ml2/coursForetsAleatoires.html.

[2] Data description (details and discussion). https://www.kaggle.com/c/ieee-fraud-detection/discussion/101203.

[3] L1 and l2 regularization methods. https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c.

[4] Regularization for logistic regression : L1, l2, gauss or laplace ? https://www.knime.com/blog/regularization-for-logistic-regression-l1-l2-gauss-or-laplace.

[5] Understanding random forests classifiers in python. https://www.datacamp.com/community/tutorials/random-forests-classifier-python.

[6] Will Koehrsen. Hyperparameter tuning the random forest in python. https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74.

[7] Robert E.Schapire Yoav Freund. A short introduction to boosting. *Journal of Japanese Society for Artificial intelligence*, September 1999.