Table des matières

[Introduction 2](#_Toc93235575)

[I. Contexte et présentation des données 2](#_Toc93235576)

[II. Organisation du code et stratégie pour gérer la volumétrie 4](#_Toc93235577)

[Partie 1 : Exploration et nettoyage des données 4](#_Toc93235578)

[Partie 2 : Création des modèles 4](#_Toc93235579)

[Calcul du chiffre d’affaires 5](#_Toc93235580)

[Déploiement du modèle sur les données de test 5](#_Toc93235581)

[Partie 3 : Déploiement et comparaison des modèles 6](#_Toc93235582)

[III. Méthodes utilisées 6](#_Toc93235583)

[A) Méthodes d’apprentissage supervisées 6](#_Toc93235584)

[Régression logistique 6](#_Toc93235585)

[Méthodes basées sur les arbres de décision 7](#_Toc93235586)

[B) Méthodes d’apprentissage non supervisées 8](#_Toc93235587)

[IV. Comparaison des performances de chaque modèle 8](#_Toc93235588)

[V. Conclusion 9](#_Toc93235589)

[VI. Perspectives d’évolution 9](#_Toc93235590)

# Introduction

Dans le cadre de notre cours de fouille de données massives, nous avons eu l’occasion de travailler sur des données réelles de fraudes issues d’une enseigne de la grande distribution ainsi que de certains organismes bancaires (FNCI et Banque de France).

L’objectif de notre projet est de proposer la meilleure solution possible pour détecter des fraudes de chèques bancaires dans un contexte déséquilibré de données massives. Pour cela, nous avons choisit le langage de programmation python qui est l’un des plus utilisés en science des données.

Tout au long de ce projet, nous avons essayer de faire très attention aux temps de calcul, à la reproductibilité et à la compréhension métier des résultats obtenus.

Afin de présenter au mieux notre travail, nous allons tout d’abord présenter le contexte et les données utilisées. Ensuite, nous détaillerons l’organisation de notre code ainsi que notre stratégie pour travailler efficacement avec une telle volumétrie de données. Enfin, nous expliquerons chaque méthode utilisée avant de comparer leurs performances. Pour conclure, nous discuterons des meilleures méthodes et de leurs perspectives d’évolution.

# I. Contexte et présentation des données

Notre fichier contient la liste de l’ensemble des transactions de chèques bancaires qui ont eu lieu dans une enseigne de la grande distribution entre le 1er février et le 30 novembre 2018. Ces données proviennent de deux organismes bancaires différents (FNCI et Banque de France). Nous avons ainsi remarqué que la ligne n°1956361 est une duplication du nom des variables qui est sûrement dû à la fusion des deux fichiers issus des deux organismes. Il faut donc en tenir compte pour l’importation des données.

Notre fichier contient 4 646 773 observations et 23 variables et sa taille est de 1 514 848 142 octets.

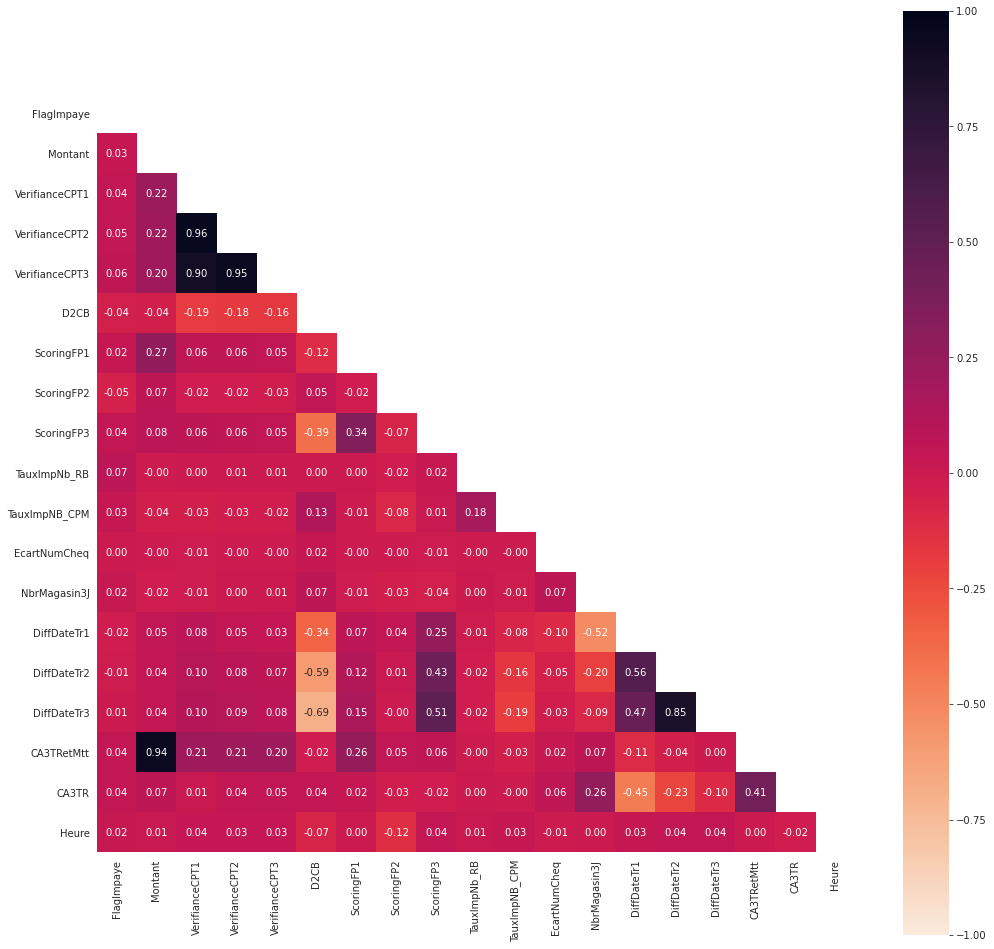
L’importation de ce fichier à l’aide la librairie pandas étant longue et coûteuse, notre stratégie est de l’importer une première fois afin de réaliser tous les traitements nécessaires (suppression de variables, changement de type etc..). Après le nettoyage de nos données, nous avons transformé ce dataframe propre en format dask qui permet une meilleure gestion des fichiers de grande dimension.

Afin de gagner un maximum de temps et d’espace mémoire, nous avons exporter ces dataframes dask propres aux format parquet à l’aide la librairie fastparquet. A partir de là, nous pouvons réimporter ces fichiers directement au format dask très rapidement.

Nous avons donc exporté les fichiers suivants :

* Le dataframe propre d’apprentissage,
* Le dataframe propre de test,
* Le dataframe propre d’apprentissage avec sur-échantillonnage,
* Le dataframe propre d’apprentissage avec sous-échantillonnage.

#### Matrice de corrélation :



#### Les changements apportés à notre jeu de données

* Variable "Heure" → Suppression parce qu’elle fait doublon avec la variable "DateTransaction".
* Variable "DateTransaction" → Séparation en deux variables "Date" et "Heure".
* Variables "ZIBZIN" et "IDAvisAutorisationCheque" → Suppression parce que ce sont des variables d'identifiants, elles n'apportent donc pas d'information pertinente.
* Variable "CodeDecision" → En observant le dataframe, nous avons remarquer que la variable qualitative "CodeDecision" possède une valeur à 4 alors que dans la description des données, cette variable ne peut prendre que les valeurs 0, 1, 2 ou 3. Nous faisons donc le choix de retirer l’observation du dataframe.
* Variables "VerifianceCPT1", "VerifianceCPT2" et "VerifianceCPT3" → Nous supprimons ces trois variables parce qu’elles sont constantes à 0 dans le dataframe de test.
* Variable "CA3TRetMtt" → Suppression dans 97,5 % des cas, il s’agit de la somme de "Montant" et de "CA3TR".

**Transformation des données : Centrage-réduction**

La méthode de transformation fit\_transform() est utilisé sur les données d'entraînement afin que nous puissions mettre à l'échelle les données d'entraînement et également apprendre les paramètres de mise à l'échelle de ces données. En utilisant la méthode transform(), nous pouvons utiliser la même moyenne et la même variance que celles calculées à partir de nos données d'entraînement pour transformer nos données de test. Ainsi, les paramètres appris par notre modèle à l'aide des données d'apprentissage nous aideront à transformer nos données de test.

Dans notre cas, nous utilisons la méthode fit\_transform() sur nos données d’entraînement initiales, puis nous utilisons la méthode transform() pour appliquer ce même centrage-réduction sur les données de test mais également sur les données d’apprentissage qui ont été sur-échantillonnées. En effet, le sur-échantillonnage de nos données aurait de grandes chances de faire varier les paramètres de centrage-réduction par rapport aux données d’entraînement initiales.

# II. Organisation du code et stratégie pour gérer la volumétrie

Afin de rendre gérer au mieux la volumétrie de ce projet et de faciliter la reproductibilité, nous avons décidé d’organiser notre code en trois parties de la manière suivante :

## A) Exploration et nettoyage des données

Le premier notebook jupyter contient les statistiques descriptives nécessaires à la compréhension de notre jeu de données. C’est ici que nous l’avons nettoyé afin qu’il corresponde au mieux à notre utilisation (suppression de variables, recodage des variables qualitatives, sur-échantillonnage, sous-échantillonnage, etc.). C’est également ici que nous avons exporté nos jeux de données propres au format dask et parquet afin qu’il soit plus facilement ré-importable par la suite (gain de temps et de lignes de codes).

## B) Création des modèles

Le second notebook jupyter contient la création des modèles de machine learning que nous avons tenté d’appliquer à notre jeu de données. Ici, nous importons directement les jeux de données propres construits à l’étape précédente. Nous n’avons donc plus qu’à réaliser quelques quelques traitements rapides tels que séparer la variable cible des variables explicatives et centrer-réduire les données d’apprentissage.

Pour construire les modèles, nous utilisons la librairie dask\_ml qui nous permet d’entraîner nos modèles de manière parallèle, ce qui nous fait gagner énormément de temps. Toujours dans un souci de gain de temps et de reproductibilité des calculs, nous utilisons la librairie pickle afin d’exporter nos modèles entraînés au format modele.sav.

### Calcul du chiffre d’affaires

Afin de déployer nos modèles entraînés sur nos données de test, nous avons créé une fonction déploiement() qui elle même fait appel à une autre fonction Calcul\_CA().

- Calcul\_CA(Montant, yReel, yPred) est une fonction qui répond à la question bonus de notre projet :

- Lorsque nous prédisons une fraude et que ça en est bien une (Vrai positif), le chiffre d’affaires est de 0.

- Lorsque nous prédisons une bonne transaction et que ça en est bien une (Vrai négatif), le chiffre d’affaires est égal au montant de la transaction. Tout s’est bien passé.

- Lorsque nous prédisons une fraude et alors que ça n’en est pas (Faux positif), le chiffre d’affaires est de 80 % du montant de la transaction. En effet, dans ce cas l’enseigne perd la confiance de son client qui ne reviendra sûrement pas. Les faux positifs sont donc très importants à minimiser.

- Lorsque nous prédisons une bonne transaction alors qu’il s’agit d’une fraude, le chiffre d’affaires est égal à 1 – exp(1/montant). Cela fait donc perdre du chiffre d’affaires à l’entreprise mais cela reste très faible comme perte.

Grâce à cette analyse, nous comprenons beaucoup mieux les enjeux de la détection de fraude et cela nous donne une information cruciale à utiliser dans le paramétrage de nos algorithmes : L’enseigne a besoin de détecter un maximum de fraude. Pourtant, il est plus avantageux pour elle de laisser passer certaines fraudes plutôt que de prédire une fraude alors que ça n’en est pas une. Afin de maximiser le chiffre d’affaires, il faut donc minimiser le taux de faux négatif et le taux de faux positif, mais il est quand même beaucoup plus important de minimiser le taux de faux positif quitte même à ce que le taux de faux négatif augmente légèrement.

### Déploiement du modèle sur les données de test

Afin de faciliter le déploiement des modèles sur les données de test et de réduire le nombre de lignes de code, nous avons créé la fonction deploiement(modele, XTest, yTest, scale, features).

Celle-ci prend en entrée :

- modele : le nom/chemin du modèle à déployer, les données

- XTest: les données explicatives de test

- yTest : la variable cible à prédire, cela nous permet de calculées les performances du modèle.

- scale : un booléen qui permet à l’utilisateur de dire si les données du modèle sont centrer-réduire ou pas.

- features : un booléen qui permet d’afficher ou non un graphique d’importance des variables (nécessité de connaître le modèle car tous ne peuvent pas afficher ce graphique).

Cette fonction fonctionne de la manière suivante :

Si l’option scale = True, nous centrons et réduisons les données XTest à l’aide du scaler déjà entraîné plus tôt. Cela permet d’appliquer le modèle sur le même type de données à partir duquel il à été construit.

Ensuite, nous importons le modèle déjà entraîné et nous l’appliquons à nos données de test pour calculer les valeurs prédites et leur score.

A partir de là et avec l’aide des données réelles de l’échantillon de test yTest, nous calculons et affichons les performances du modèle qui sont : la matrice de confusion, le rappel, la précision, l’air sous la courbe ROC (auc) et le chiffre d’affaires généré. Ces données de performance sont retournées par la fonction afin qu’elles puissent être stockées et comparées entre elles.

## C) Déploiement et comparaison des modèles

Le troisième et dernier notebook jupyter contient uniquement le déploiement de tous les modèles. Il est assez rapide d’exécution puisqu’il fait à appel aux modèles précédemment entraînés. Suite à cela, nous pouvons comparer les différents modèles à l’aide de graphiques comparatifs basés sur différents indicateurs de performance tels que le chiffre d’affaires généré, l’aire sous la courbe ROC (auc), le rappel, la précision et l’accuracy.

# III. Méthodes utilisées

## A) Méthodes d’apprentissage supervisées

### Régression logistique

Nous avons décidé d’utiliser la régression logistique car elle possède de nombreux hyperparamètres qui nous permettent d’essayer plusieurs modèles en même temps. En effet, entre les solveurs, les pénalisations, les degrés des pénalités et les stratégies d’échantillonnage, nous avons pu tester de nombreuses situations.

Régression logistique par défaut :

Nous avons dans un premier temps commencé par une régression logistique par défaut. De ce fait, le solveur utilisé est le ‘liblinear’ qui n’est pas conseillé pour les grands jeux de données. Les résultats s’avèrent en effet médiocre.

Régression logistique solveur SAGA :

La documentation de scikit-learn préconise d’utiliser le solveur SAGA pour les grands jeux de données. Les résultats sont sans appel, l’AUC augmente et le nombre de fraude détectées est multiplié par 3. En revanche nous augmentons également le nombre de faux positifs et nous avons pu voir précédemment que cela n’était pas une bonne chose.

Régression logistique avec différentes pénalisations :

Nous avons ensuite pensé à essayer sur les différentes pénalisations. Les résultats n’ont que très peu bougé.

Régression logistique avec sur-échantillonnage SMOTE :

La méthode d’échantillonnage SMOTE apporte un avantage sur la reconnaissance de fraudeurs mais augmente considérablement le nombre de bons payeurs reconnus comme fraudeurs, et nous avons vu que cela n’était pas une bonne chose. Même si l’AUC est plus élevée, dénoncer une plus grande quantité des personnes non-fraudeuses comme fraudeuses n’est pas ce qui est recherché.

Pour conclure, il semblerait que la régression logistique ne soit pas l’algorithme le plus adapté pour ce genre d’analyse.

### Méthodes basées sur les arbres de décision

Après de nombreuses recherches, nous avons remarqué que les algorithmes basés sur des arbres de décision étaient préconisés pour de la détection de fraudes bancaires.

#### Arbre de décision

Nous avons donc commencé par entraîner un arbre de décision simple qui est très sensible au sur-apprentissage. Celui-ci donne des résultats mitigés et bien moins bon que les méthodes de régression logistique vu précédemment.

#### Forêt aléatoire

La forêt aléatoire est une méthode plus robuste et donc moins sujette au sur-apprentissage que qu’un arbre de décision seul. Plus coûteuses en calcul, dans notre cas nous avons utilisé 10 arbres de décision. L’algorithme apprend alors sur 10 arbres de décision différents et retourne 10 prédictions. La prédiction retenue est celle qui a été le plus souvent prédite. Nous remarquons que les résultats sont excellents, le nombre de faux positifs est très réduit.

#### Adaboost avec Arbre de décision

Adaboost est une méthode d’apprentissage basé sur les apprenants faibles. C’est-à-dire qu’il va se servir de ses erreurs pour progresser petit à petit. Il est également moins sujet au sur-apprentissage que d'autres algorithmes. Dans notre cas, il est basé sur un arbre de décision de profondeur 1 et on a fixé le nombre maximal d’estimateurs à 10. Les résultats de cette méthode sont encore meilleurs que la forêt aléatoire précédente. L’aire sous la courbe ROC est légèrement plus faible les performances globales sont bien meilleures. En effet, il ne prédit aucun faux positif et c’est une excellente nouvelle ! Le nombre de faux négatifs est légèrement plus haut mais cela nous permet de maximiser le chiffre d’affaires. Nous rappelons qu’il est beaucoup plus avantageux de minimiser au maximum le nombre de bons clients identifiés comme fraudeurs quitte même à augmenter légèrement le nombre fraudeurs non détectés.

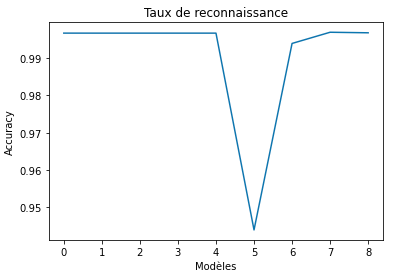
## B) Méthodes d’apprentissage non supervisées

# IV. Comparaison des performances de chaque modèle

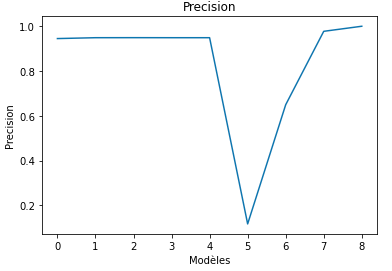
Voici un récapitulatif de tout les modèles d’apprentissage supervisé que nous avons testé :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Modèle** | **Données** | **CA** | **AUC** | **Rappel** | **Précision** | **Accuracy** |
| modele1.sav | LogisticRegression(random\_state=0) | Initiales + Centrage-réduction | 45836958.17 | 0.8327 | 0.6657 | 0.9449 | 0.9967 |
| modele2.sav | LogisticRegression(random\_state=0, solver='saga') | Initiales + Centrage-réduction | 45841059.47 | 0.8309 | 0.6622 | 0.9487 | 0.9967 |
| modele3.sav | LogisticRegression(C=0.1, penalty='l1', random\_state=0, solver='saga') | Initiales + Centrage-réduction | 45844719.47 | 0.8309 | 0.6622 | 0.9489 | 0.9967 |
| modele4.sav | LogisticRegression(C=0.1, random\_state=0, solver='saga') | Initiales + Centrage-réduction | 45841059.47 | 0.8309 | 0.6622 | 0.9487 | 0.9967 |
| modele5.sav | LogisticRegression(C=0.01, l1\_ratio=0.5, penalty='elasticnet', random\_state=0, solver='saga') | Initiales + Centrage-réduction | 45844667.27 | 0.8309 | 0.6622 | 0.9487 | 0.9967 |
| modele6.sav | LogisticRegression(random\_state=0, solver='saga') | SMOTE + Centrage-réduction | 44935012.82 | 0.8822 | 0.8194 | 0.1168 | 0.9439 |
| modele7.sav | DecisionTreeClassifier(random\_state=0) | Initiales | 45838330.04 | 0.8381 | 0.6795 | 0.6492 | 0.9939 |
| modele8.sav | RandomForestClassifier(n\_estimators=10, random\_state=0) | Initiales | 45884533.32 | 0.8345 | 0.6692 | 0.9773 | 0.9969 |
| modele9.sav | AdaBoostClassifier(  base\_estimator=  DecisionTreeClassifier(  max\_depth=1, random\_state=0), n\_estimators=10, random\_state=0) | Initiales | 45889118.39 | 0.8183 | 0.6366 | 1.0 | 0.9968 |
|  |  |  |  |  |  |  |  |

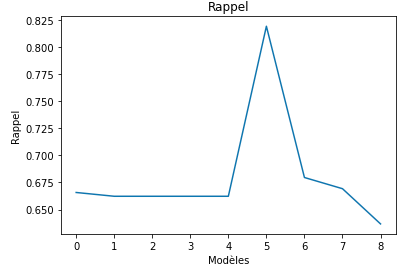
#### Taux de reconnaissance



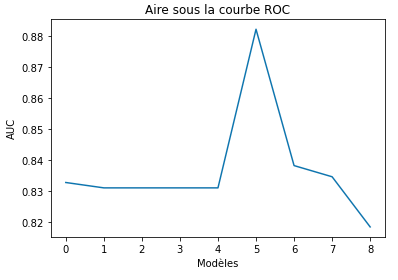
#### Précision



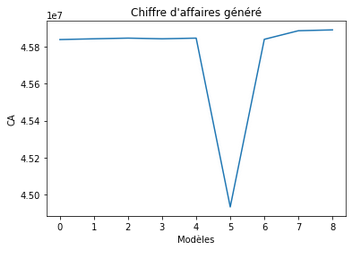
#### Rappel



#### Aire sous la courbe ROC



#### Chiffre d’affaires généré



Pour réaliser ce projet de détection de fraudes bancaires, nous avons choisi de comparer nos modèles à l’aide de quatre éléments différents : le chiffre d’affaires généré, l’aire sous la courbe ROC (auc), le rappel, la précision et l’accuracy. Ce choix est arbitraire, il existe bien d’autres métrique d’évaluation.

Selon la nature de notre problème à résoudre et les besoins métiers, nous allons privilégier certaines métriques au lieu d’autres. Dans notre cas, des experts métiers ont pu estimer le chiffre d’affaires qui seraient généré dans chacune des situations (Vrai positif / Faux positif / Vrai négatif / Faux négatif). Cela nous donne donc un indicateur de performance personnalisé bien plus robuste que les autres.

Il s’agit donc d’une question de point de vue :

- Si l’on regarde le chiffre d’affaires généré ou la précision, le meilleur modèle est le modèle 9.

- Si l’on regarde l’aire sous la courbe ROC ou le taux de rappel, le meilleur modèle est le modèle 6.

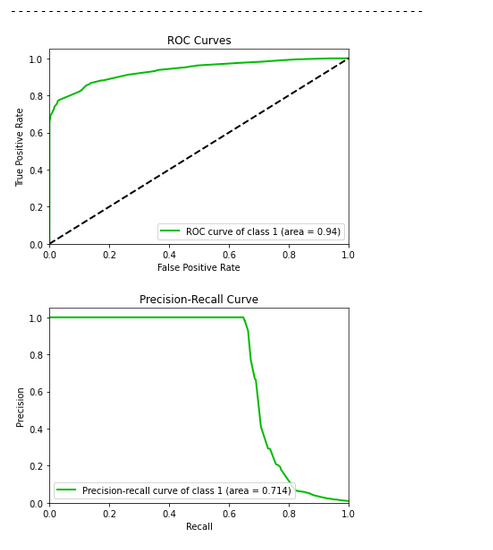
- Si l’on regarde l’accuracy, le meilleur modèle est le modèle 8.

Dans notre cas de détection de fraudes bancaires, la priorité est de maximiser le chiffre d’affaires généré tout en minimisant au maximum le nombre de bons clients considérés comme fraudeurs (précision). Nous pouvons donc conclure qu’ici, **le meilleur modèle est le modèle 9**.

#### Détail du meilleur algorithme : le modèle 9

Une image contenant table

Description générée automatiquement



# V. Conclusion

Pour conclure, nous pouvons dire que nous avons beaucoup apprécié ce projet. En effet, cela nous a permis de nous familiariser avec de nombreuses méthodes et concepts tels que : la gestion des données très volumineuses, l’optimisation des temps de calcul, la reproductibilité, la sauvegarde et le déploiement de modèles déjà entraîner etc...

De plus, ce projet lié à des données réelles de fraude nous a permis de comprendre beaucoup de choses notamment liées l’évaluation des modèles qui dépend énormément du contexte. En effet, ici le but est de détecter le maximum de fraude mais pas à tout prix. En effet, les indicateurs de performance classique (AUC, accuracy, etc.) sont importants et peuvent nous donner de bons résultats. Cependant, dans notre contexte, nous avons développé un indicateur de performance personnalisé (le chiffre d’affaires) et celui-ci est bien meilleur que les indicateurs classiques dans ce cas-là. En effet, au-delà de la reconnaissance des fraudes, il est préférable pour l’enseigne de laisser passer certains fraudeurs plutôt que de considérer des bons clients comme fraudeurs. Cela a un meilleur impact sur le chiffre d’affaires. Le contexte est ici très important car sans cela, nous aurions peut-être choisi un modèle moins performant.

Le meilleur modèle que nous avons réussi à produire est un **modèle Adaboost avec Arbre de décision**. En effet, c’est celui qui maximise le plus le chiffre d’affaires généré et la précision. De plus, nous avons également remarqué que ce modèle minimise le plus possible le nombre de faux positifs qui tombe même à 0 sur nos données de test. Après avoir cru à du sur-apprentissage, nous nous sommes finalement dit que cela correspondait plutôt bien à la réalité.

# VI. Perspectives d’évolution

Une des perspectives d’évolution de notre projet consisterait à détecter automatiquement si les modèles déployés s’utilisent sur des données centrées-réduites et si l’on peut afficher un graphique d’importance des variables. Actuellement, la fonction de déploiement nécessite que l’utilisateur connaissent le modèle construit afin de mettre les bons paramètres dans fonctions de déploiement.

Une autre perspective d’évolution de notre projet serait d’utiliser GridSearchCV pour optimiser au maximum les paramètres de nos algorithmes.

Et enfin, une dernière perspective d’évolution serait de créer une interface graphique utilisable par un utilisateur lambda.