Université Lumière Lyon 2
Institut de la communication

Université Lumière Lyon 2

Ensemble Methods Project

M2 Computer Science MALIA

BABEY Cassien

Table des matières

Introduction

Dans le cadre du projet « Ensemble Methods Project » de l'UE « Ensemble Methods in Machine Learning » du Master 2 MALIA, nous allons tenter de réaliser une expérimentation tournée vers la détection de fraudes. Pour ce faire, un ensemble de quatre data sets est mis à notre disposition. Ces 4 data sets sont disponibles sur le GitHub suivant : https://github.com/marrvolo/SCDA sous la forme de 4 sets de données représentant un train/test split déjà réalisé. Très peu d'informations sont disponibles quant à la composition des variables de ces data sets, pouvant limiter nos interprétations et hypothèses.

L'objectif principal de notre projet sera dans un premier temps de définir la meilleure métrique pour nos sets de données puis de développer un ensemble de méthodes d'apprentissage automatique afin de comparer les performances de ces différentes méthodes et définir la meilleure méthode a utilisé pour chacun des data sets étudiés dans ce projet.

Méthodologie

Nous commençons par introduire les notations essentielles pour notre étude. Soit $\boldsymbol{X}$ la matrice des caractéristiques, $\boldsymbol{y}$ les étiquettes correspondantes, $\boldsymbol{H}$ la fonction d'hypothèse, $\boldsymbol{S}$ les sets d'entrainement et $\boldsymbol{w}$ un vecteur poids.

Choix de la mesure de performance

Notre objectif est d'optimiser une mesure de performance adaptée à notre contexte de classe déséquilibrée. En effet, dans le cas de la détection de fraude, les cas de fraude sont souvent rares par rapport aux transactions normales, créant un déséquilibre dans la distribution des classes. Nous avons ici choisi le score F1 comme mesure de performance principale. Le F1 score est l'harmonique de la précision et du rappel (*F1_score = 2 * (précision * rappel) / (précision + rappel)*). Cette métrique est particulièrement efficace dans les situations où les classes sont déséquilibrées, car elle prend en compte à la fois les faux

positifs et les faux négatifs. L'utilisation du F1 score comme mesure de performance nous assure que notre modèle est capable de détecter de manière fiable les fraudes tout en minimisant les erreurs de classification.

Choix des modèles de prédiction

Notre projet vise à comparer efficacement diverses méthodes de machine learning dans la détection de fraude. Pour ce faire, chaque modèle est soumis à une validation croisée et à une optimisation via Grid Search. Nous évaluons la performance à l'aide du score F1 tout en prenant en compte le temps de traitement des modèles. Les modèles retenus sont ensuite testés sur l'ensemble des données sur 10 itérations pour assurer leur stabilité et fiabilité. Cette approche devrait nous garantir un ratio intéressant entre performance et temps de traitement.

Nous avons donc choisi ces modèles de machine learning pour la détection de fraudes :

Régression Logistique (L1 et L2) : La régression logistique est un modèle linéaire utilisé pour la classification binaire. Pour la régularisation L1 (Lasso), la fonction de coût est :

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(h(\theta^{(i)})) + (1-y^{(i)})\log(1-h(\theta^{(i)}))\right] + \lambda\sum_{j=1}^{n}|\theta_j|$$

Où L1 pousse certains coefficients vers zéro pour essayer de simplifier le modèle.

Pour la régularisation L2 (Ridge), la fonction coût à optimiser est :

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(h(\theta^{(i)})) + (1-y^{(i)})\log(1-h(\theta^{(i)}))\right] + \lambda\sum_{j=1}^{n}(\theta_j)^2$$

Où L2 tend à réduire les valeurs de tous les coefficients de manière proportionnelle pour augmenter la stabilité du modèle.

Bagging Classifier avec Régression Logistique : Le Bagging Classifier améliore la régression logistique en agrégeant les prédictions de plusieurs régressions logistiques. La combinaison des prédictions devrait permettre au modèle d'augmenter son score de performance. Nous utiliserons le meilleur modèle de régression logistique obtenu avant pour pouvoir comparer leur performance.

Random Forest Classifier : Un ensemble d'arbres de décision pour augmenter le score de performance. Le pseudo-code :

Algorithm 4: Random Forests

Inputs : Training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$, number of trees T, a sample size m' and a number of features p'

Output: Hypothesis H_K

1: **for** $t = 1$ to T **do**
2: create a bootstrap sample S_t of size $m' \leq m$ using S.
3: Build a decision tree h_t where at each split, a random subsample of p' features are used to split the node.
4: **end for**
5: set $H_T = \frac{1}{T}\sum_{t=1}^{T} h_t$ =0

Adaboost : Un modèle basé sur la méthode du boosting se focalisant sur les erreurs passées et l'assignation de poids pour améliorer le modèle final. Voici le pseudo-code du modèle :

Algorithm 5: Adaboost

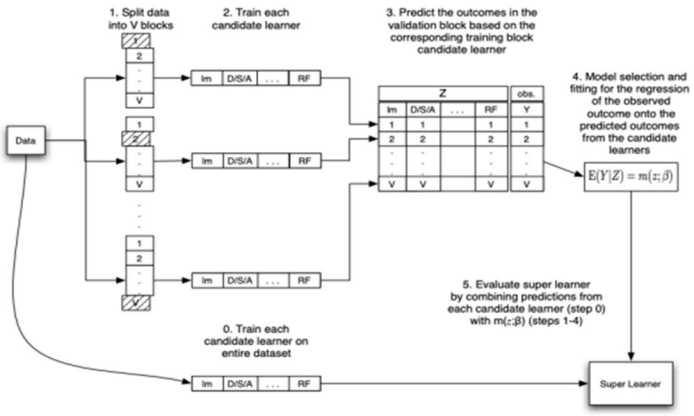
Inputs : Training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$, number of model T

Output: Hypothesis H_T

1: **for** $i = 1$ to m **do**
2: the weight of the i-th example $w_i^{(1)}$ is equal to $1/m$
3: **end for**
4: **for** $t = 1$ to T **do**
5: learn a base classifier h_t using S with the weights $\mathbf{w}^{(t)}$.
6: compute the error $\varepsilon_t = \sum_{i=1}^{m} w_i^{(t)} \mathbb{1}_{\{h_t(\mathbf{x}_i) y_i < 0\}}$ of h_t
7: set $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$
8: set $Z_t = 2\sqrt{\varepsilon_t(1-\varepsilon_t)}$
9: **for** $i = 1$ to m **do**
10: set $w_i^{(t+1)} = w_i^{(t)} \frac{\exp(-\alpha_t y_i h_t(\mathbf{x}_i))}{Z_t}$
11: **end for**
12: **end for**
13: set $H_T = \frac{1}{T}\sum_{t=1}^{T} \alpha_t h_t$ =0

Support Vector Machine (SVM) Classifier : L'objectif du modèle est de trouver un hyperplan optimal pour séparer nos classes. Nous testerons différents SVC.

Stacking de SVC avec un Random Forest en sortie : Le stacking combine les prédictions de plusieurs SVM et d'un Random Forest. Les SVM capturent des relations complexes, tandis que le Random Forest gère les structures non linéaires. La combinaison de modèles différents devrait permettre d'augmenter le score de performance. Une cross-validation est intégrée dans la version SKLearn du modèle. L'objectif ici est de voir si le stacking est une méthode vraiment intéressante pour augmenter le score de performance d'un modèle dans une tâche de classification. Nous nous sommes basés sur les travaux de Chand et al., 2016 pour la construction de ce modèle



XGBoost : Algorithme de boosting de gradient basé sur des arbres de décision pour des prédictions améliorées. Voici le pseudo-code du modèle :

Algorithm 7: Gradient Boosting [Friedman, 2000]

Inputs : Training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$, number of models T, a loss ℓ

Output: a model $H_T = H_0 + \sum_{t=1}^{T} \alpha_t h_{a^t}(\mathbf{x})$

1: Initialization $H_0 \quad H^0(\mathbf{x}_i) = \arg\min_{\rho \in \mathbb{R}} \sum_{i=1}^{m} \ell(y_i, \rho) \; \forall i = 1, \ldots, m$

2: **for** $t = 1$ to T **do**

3: Compute the gradient: $\tilde{y}_i = -\dfrac{\partial \ell(y_i, H_{t-1}(\mathbf{x}_i))}{\partial H_{t-1}(\mathbf{x}_i)}, \; \forall i = 1, ..., m$

4: Learn a model that approximates this gradient $a_t = \arg\min_{a \in \mathbb{R}^d} \sum_{i=1}^{m} (\tilde{y}_i - h_a(\mathbf{x}_i))^2$ to learn the new hypothesis

5: Find the optimal weight h_{a^t} : $\alpha^t = \arg\min_{a \in \mathbb{R}^+} \sum_{i=1}^{m} \ell(y_i, H_{t-1}(\mathbf{x}_i) + \alpha h_{a^t}(\mathbf{x}_i))$

6: Update $H_t(\mathbf{x}_i) = H_{t-1}(\mathbf{x}_i) + \alpha^t h_{a^t}(\mathbf{x}_i)$

7: **end for**=0

La définition de nos différents modèles étant faite, nous pouvons maintenant nous lancer dans la phase d'expérimentation de nos différents modèles.

Expérimentation

Dans cette section, nous plongeons dans l'expérimentation rigoureuse de nos modèles de prédiction. Notre objectif est de déterminer quelle méthode offre le meilleur score F1 dans la détection de fraudes, basée sur une évaluation approfondie de chaque modèle et de leurs performances respectives. Les détails de notre approche expérimentale, y compris les ensembles de données, les méthodes d'optimisation des hyperparamètres et les protocoles de validation, sont présentés pour garantir la clarté et la reproductibilité de nos résultats. Nous utiliserons ici principalement du pseudo-code pour expliciter les différentes étapes mais vous pouvez retrouver la partie « code » dans le fichier ipynb joint à ce projet.

Protocoles

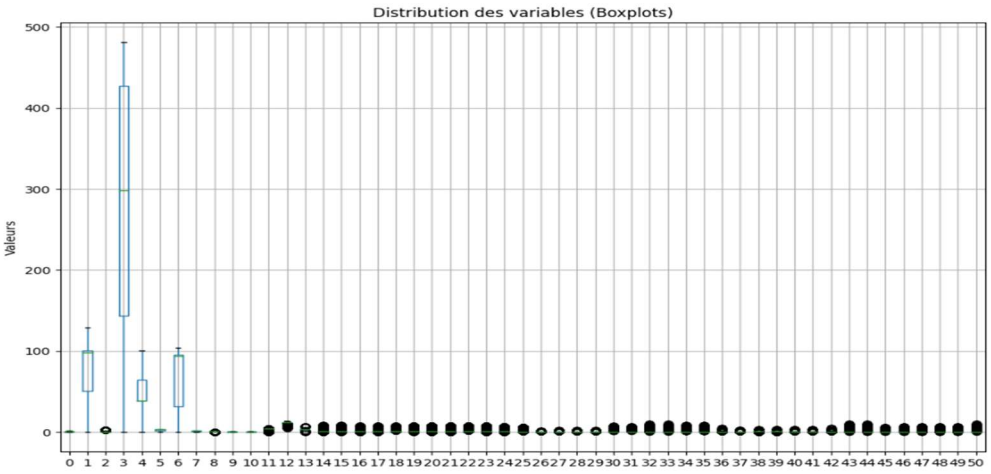
Préparation des données

Comme nous l'avons indiqué dans notre introduction, les sets de données utilisés dans notre projet sont au nombre de 4. Chacun de ces data sets a déjà subit un prétraitement et ne contient donc aucune valeur manquante. De plus une division entre données d'entrainement et données de test a déjà été réalisée. Enfin, un prétraitement des y a été réalisé pour ne garder que la variable cible. Les informations sur la distribution des différents data sets sont retranscrites dans le tableau 1 ci-dessous :

	dataset	shape	X_test_shape	X_train_shape	y_test_shape	y_train_shape	Train_dist_label
0	kaggle_source_cate_0	(54744, 52)	(13686, 51)	(41058, 51)	(13686, 2)	(41058, 2)	10.041892
1	kaggle_source_cate_1	(54744, 52)	(13686, 51)	(41058, 51)	(13686, 2)	(41058, 2)	9.956647
2	kaggle_source_cate_2	(54744, 52)	(13686, 51)	(41058, 51)	(13686, 2)	(41058, 2)	9.832432
3	kaggle_source_cate_3	(54744, 52)	(13686, 51)	(41058, 51)	(13686, 2)	(41058, 2)	9.815383

Tableau 1 : *Distribution des données des différents data sets.*

Avec le peu d'informations disponibles sur les variables composant nos différents data sets, nous avons réalisé une visualisation des données en boxplot pour observer la distribution de ces variables. Nous ne monterons ici que la distribution du premier set d'entrainement (Graph 1).



Graph 1 : *Distribution en Boxplot des variables du set d'entrainement 1*

Nous pouvons observer quatre variables numériques puis un ensemble de variables comprises dans des intervalles très faible. Nous pouvons supposer qu'il s'agit de variables catégorielles ou ordinales qui ont potentiellement subi un OneHotEncoding. Dans tous les cas, une normalisation des données semble nécessaire due à la présence des variables numériques.

Cross-validation des modèles et GridSearch des hyperparamètres

Nous avons opté pour une approche méthodique en employant la validation croisée (cross-validation) et la recherche systématique d'hyperparamètres (GridSearch). La validation croisée en k-folds divise nos données en plusieurs sous-ensembles (k), entraînant et évaluant le modèle k fois, garantissant ainsi une évaluation robuste de sa performance. Pour ce faire, nous utiliserons la fonction Stratified K-Fold de SKLearn qui nous permet de créer nos k-folds de manière manuelle en ayant un contrôle plus important sur la division et l'étape de normalisation contrairement aux autres méthodes plus compactes et « black box » de SKLearn. En parallèle, la méthode GridSearch explore méthodiquement un espace prédéfini d'hyperparamètres pour chaque modèle, évaluant chaque combinaison grâce à la validation croisée. Cette double technique garantit que nos modèles seront le plus optimisés et robustes possibles pour notre tâche. Les résultats obtenus grâce à cette méthodologie seront essentiels pour éclairer nos choix finaux. Cette méthodologie est appliquée à l'ensemble de nos

méthodes pour nos quatre sets d'entrainement à l'exception des méthodes de Bagging et de Stacking car nous aurons déjà obtenus les hyperparamètres optimaux pour les modèles les composants et partirons donc du postulat que ce n'est pas nécessaire. Notre méthodologie générale est représenté par le pseudo-code ci-dessous :

Algorithm 1: GridSearchCross-Validate Methodology

```
Result: F1 scores, best parameters, and training times for each dataset and model
Import Xtrain and ytrain;
Create 5 similar k-fold using StratifiedKFold for each dataset;
Normalize features of all datasets during cross-validation;
for each dataset do
    for each model do
        for each parameter configuration do
            Perform GridSearchCV with 5-fold cross-validation;
            Calculate average F1 score using cross-validated results;
            Store average F1 score;
        end
        Select best parameters based on the highest average F1 score;
        Store best parameters;
    end
end
Create a dataframe containing datasets names, F1 scores, the best parameters, training
 times.
```

Après avoir déterminé les meilleurs hyperparamètres via la validation croisée, nous passons à l'étape suivante : l'entraînement de nos modèles. Dans cette phase, chaque modèle est entraîné sur l'intégralité de l'ensemble de données d'entraînement, puis évalué à l'aide du score F1 sur nos données de test. Pour garantir la stabilité et la fiabilité de nos résultats, nous répétons ce processus d'entraînement et d'évaluation sur 10 itérations. La moyenne des scores F1 obtenus à partir de ces itérations nous fournit un indicateur robuste de la performance de chaque modèle. La méthodologie pour cette partie est explicitée via le pseudo-code suivant :

Algorithm 2: Model Training and Evaluation Workflow

```
Result: Average F1 score and process time for each model after training and evaluation
for each model do
    Initialize an empty list to store F1 scores;
    for iteration in range(1, 11) do
        Split the dataset into training and test sets;
        Train the model on the entire training dataset using the best hyperparameters;
        Evaluate the model on the test dataset and calculate F1 score;
        Store the F1 score in the list;
    end
    Calculate the average F1 score and std from the list of scores;
    Store the average F1 score and std for the model;
end
Create a summary table with model names and their corresponding average F1 scores and
 process times;
Display the summary table;
```

Résultats

Comparaison générale des modèles

Les scores F1 moyens des différents modèles après 10 itérations sont retranscrits dans le tableau 2 ci-dessous :

	Dataset	AdaBoost Classifier	Bagging (Logistic Regression)	Logistic Regression	Random Forest Classifier	SVC	Stacking Classifier	XGBoost Classifier
0	Dataset_0	0.64 ± 0.00	0.56 ± 0.00	0.56 ± 0.00	0.75 ± 0.00	0.23 ± 0.00	0.44 ± 0.00	0.79 ± 0.00
1	Dataset_1	0.63 ± 0.00	0.55 ± 0.00	0.55 ± 0.00	0.74 ± 0.00	0.22 ± 0.00	0.44 ± 0.00	0.78 ± 0.00
2	Dataset_2	0.60 ± 0.00	0.52 ± 0.00	0.52 ± 0.00	0.74 ± 0.00	0.20 ± 0.00	0.44 ± 0.00	0.77 ± 0.00
3	Dataset_3	0.61 ± 0.00	0.53 ± 0.00	0.54 ± 0.00	0.74 ± 0.00	0.21 ± 0.00	0.45 ± 0.00	0.79 ± 0.00
4	Mean	0.62 ± 0.00	0.54 ± 0.00	0.54 ± 0.00	0.74 ± 0.00	0.22 ± 0.00	0.44 ± 0.00	0.78 ± 0.00
5	Rank	3	4	5	2	7	6	1

Tableau 2 : *Moyenne des F1 scores sur les 10 itérations pour chaque modèle*

On observe que le modèle « XGBoost Classifier » semble offrir la meilleure performance en termes de F1 score sur l'ensemble des 4 datasets ($M = 0.78 \pm 0.00$). Le « Random Forest Classifier » offre également un score de performance très similaire ($M = 0.74 \pm 0.00$). Le modèle « Adaboost » se place à la 3e place avec un score de performance un peu plus faible suivi par nos modèles de « Régression Linéaire Pénalisée » et de « Bagging » qui propose des performances similaires. Enfin, nos modèles de « Stacking » et de « SVC » offrent les scores de performances les plus faibles sur nos sets de données ($M < 0.5$). Enfin, on observe que nos modèles sont très stables même après 10 itérations ($std = 0.00$ pour tous les modèles).

Model Dataset	AdaBoost Classifier	Bagging (Logistic Regression)	Logistic Regression	Random Forest Classifier	SVC	Stacking Classifier	XGBoost Classifier
Dataset_0	27.60 ± 0.11	25.79 ± 0.10	4.03 ± 0.18	35.57 ± 0.10	25.16 ± 0.18	127.82 ± 0.79	9.20 ± 0.39
Dataset_1	27.67 ± 0.16	25.17 ± 0.08	3.95 ± 0.32	35.52 ± 0.26	24.69 ± 0.08	126.03 ± 0.55	9.24 ± 0.32
Dataset_2	27.63 ± 0.15	27.12 ± 0.06	4.28 ± 0.34	35.33 ± 0.29	24.65 ± 0.17	125.23 ± 0.90	9.05 ± 0.62
Dataset_3	27.64 ± 0.12	25.71 ± 0.07	4.34 ± 0.21	35.67 ± 0.42	24.24 ± 0.10	123.99 ± 0.44	9.40 ± 1.12

Tableau 3 : *Moyenne des durées d'entrainement sur les 10 itérations pour chaque modèle en secondes*

Model Dataset	AdaBoost Classifier	Bagging (Logistic Regression)	Logistic Regression	Random Forest Classifier	SVC	Stacking Classifier	XGBoost Classifier
Dataset_0	0.38 ± 0.00	0.04 ± 0.01	0.00 ± 0.00	0.39 ± 0.01	6.38 ± 0.16	6.55 ± 0.18	0.02 ± 0.00
Dataset_1	0.38 ± 0.01	0.05 ± 0.01	0.00 ± 0.00	0.39 ± 0.00	6.35 ± 0.29	6.46 ± 0.09	0.02 ± 0.00
Dataset_2	0.38 ± 0.00	0.04 ± 0.02	0.00 ± 0.00	0.39 ± 0.01	6.17 ± 0.19	6.41 ± 0.20	0.02 ± 0.00
Dataset_3	0.37 ± 0.01	0.04 ± 0.02	0.01 ± 0.00	0.39 ± 0.01	6.09 ± 0.06	6.39 ± 0.25	0.02 ± 0.00

Tableau 4 : *Moyenne des durées de prédiction sur les 10 itérations pour chaque modèle en secondes*

Toujours dans une optique de mesure de la performance globale des modèles, nous avons également enregistrer les durées d'entrainement (*Tableau 3)* et de prédiction (*Tableau 4)* de chaque modèle pour nous aiguiller vers une sélection plus éclairée du meilleur modèle pour notre projet. On observe que le modèle de « Régression Logistique Pénalisée » est le modèle le plus rapide à entrainer (environ 4 secondes par data set) suivi par le modèle « XGBoost Classifier » (environ 9 secondes). Le modèle de « Stacking » est le plus long a entrainé avec une durée moyenne de 125.5 secondes. Les autres modèles présentent une durée d'environ 30 secondes pour entrainer un data set de notre projet. Au niveau des durées de prédiction, à part les modèles « SVC » et « Stacking » qui présentent des durées de prédictions d'environ 6 secondes, tous les autres modèles sont en dessous de la seconde.

Comparaison des modèles de Régression Logistique et de Bagging

On observe que les deux modèles montrent des scores de performances similaires sur nos quatre data sets (*M = 0.54 ± 0.00).* On peut donc supposer que la méthode de Bagging n'a pas ajouté de valeur significative en termes d'amélioration de performance d'un modèle de Régression Logistique seul. De plus, la durée d'entrainement est plus rapide pour la Régression Logistique que pour la méthode de Bagging (*Mlog ± 4 secondes vs Mbag ± 26 secondes*). On peut donc conclure que la méthode de Bagging n'apporte pas d'amélioration en termes de performance sur notre métrique ou en durée d'entrainement pour nos données spécifiques.

Comparaison des modèles de SVC et de Stacking

On observe qu'au niveau des performances, le modèle « SVC » offre les performances les plus faibles de tous les modèles (*M=0.22 ± 0.00*) suivi par le « Stacking » (*M=0.44 ± 0.00*). Cependant, on peut remarquer une nette augmentation des performances du SVC quand il est inclus dans une méthode de Stacking. Cette nette augmentation de la performance, peut être expliquer par l'utilisation d'un « Random Forest Classifier » en sortie de « Stacking » qui permet supposément de mieux capturer certains aspects de nos données qu'un « SVC » seul. Il faut toutefois noter que la durée d'entraînement est très importante dans le cas du « Stacking » comparé aux autres méthodes étudiées.

On peut donc conclure que malgré une amélioration de la performance avec l'utilisation de la méthode de « Stacking », cette amélioration ne permet pas à notre « SVC » d'avoir des performances acceptables. Les coûts supplémentaires en termes de complexité et

de temps d'entraînement de cette méthode nous poussent à la considérer comme potentiellement non optimale dans le cas de nos sets de données.

Conclusion

La détection de fraude exige des méthodes d'apprentissage automatique robustes, efficaces et rapides. Dans ce projet, nous avons employé une série de modèles avancés pour offrir une étude comparative qualitative des différentes méthodes de classification. Notre travail s'est concentré sur l'optimisation et l'évaluation de ces modèles en utilisant le score F1 comme mesure de performance principale, compte tenu du déséquilibre important dans nos jeux de données.

Nos résultats montrent que le modèle XGBoost Classifier a dominé en termes de performances, avec le score F1 moyen le plus élevé sur l'ensemble des data sets étudiés. Son équilibre entre la précision et la robustesse en fait un candidat idéal pour la détection de la fraude. Cependant, le facteur temps est aussi essentiel dans le choix d'un modèle, notamment dans les applications en temps réel. Bien que le modèle XGBoost se soit distingué en matière de performance, c'est la Régression Logistique qui a été la plus rapide à entraîner. Même si les performances de ce modèle sont plus limitées que le XGBoost, il peut arriver dans certaines situations que l'on préfère privilégier la vitesse d'entrainement aux performances. Dans notre cas, le XGBoost propose tout de même le ratio performance/vitesse d'entraînement et corrobore l'idée générale que ce modèle est souvent parmi les meilleurs dans de nombreuses tâches, que ce soit en régression ou en classification.

D'autre part, certaines méthodes, que nous espérions prometteuse, comme le Bagging ou le Stacking, n'ont pas montré d'amélioration réellement significative pour nos données. L'implémentation de ces deux méthodes pour comparer les performances nous a tout de même permis de mettre en lumière que ces méthodes, bien que souvent considérées comme meilleures que certaines méthodes unitaires, n'offrent pas toujours des améliorations conséquentes, que ce soit en termes de performance, ou en termes de vitesse d'entraînement.

Dans notre projet, nous avons même pu observer que ces modèles peuvent même être contre-performant, ajoutant du poids à l'importance de réaliser ce type d'expérimentation.

Il est aussi important de signaler que notre étude a certaines limitations. Tout d'abord, les informations disponibles pour nos données ont été très limitées, pouvant limiter nos hypothèses et certains pré-traitements de celles-ci. Sinon, nous avons principalement basé notre évaluation sur les performances du score F1 via GridSearchCV sans une analyse approfondie de surapprentissage ou sous-apprentissage. De plus, bien que nous ayons exploré divers modèles, une optimisation plus exhaustive des hyperparamètres ainsi que l'utilisation de techniques d'équilibrage des classes comme l'Oversampling aurait pu être bénéfique. Cependant, le but du projet étant une comparaison de méthodes et non la mise en place du meilleur modèle livrable possible, nous avons donc choisi de faire fit de cette analyse approfondie qui aurait requis plus de ressources de calcul que nous n'en disposions pour réaliser ce projet dans les temps.

En conclusion, notre étude démontre l'efficacité de l'application méthodique des techniques d'apprentissage automatique à la détection de la fraude. Chaque modèle présente ses avantages et ses inconvénients, et le choix final dépendra des exigences spécifiques du potentiel client. Il apparaît donc essentiel de toujours tester, évaluer et, si nécessaire, adapter plusieurs modèles pour garantir les meilleurs résultats possibles et pouvoir répondre au mieux à la problématique posée.

Fin.