L2-Mini Projet
Xporters - groupe Caravan
Membres : Ouassim Meftah, Jeyanthan Markandu, Julien Rolland, Mohamed Oumezzaouche, Pierre Cornemillot
URL du challenge: https://codalab.lri.fr/competitions/652
Repo GitHub du projet : https://github.com/pierre6398/caravan

XPorters

Contexte et description du problème :

Le challenge qui nous a été proposé est une prédiction du trafic sur une autoroute, c'est-à-dire le nombre de voitures sur cette autoroute à une date et une heure donnée. Pour cela, on nous a fourni un "starting kit" avec des données et un squelette de programme permettant d'effectuer une régression. Le set de données (séparé en 3 ensembles : entraînement, test et validation) contient des mesures de la densité volumique de voitures à différentes dates, et dans différentes conditions météorologiques. Il pourrait être intéressant en ouverture à la fin du projet de s'intéresser à d'autres métriques pour comparer les résultats. Avec celle-ci, le notebook et le modèle originel permettent déjà d'obtenir un score de 0,9467. Notre objectif est de l'améliorer en cherchant un meilleur modèle de régression et en manipulant les données. Notre groupe divisera donc son travail en trois tâches principales (preprocessing, modélisation et visualisation) qui seront détaillées plus bas. Le résultat initial étant déjà très correct (plus que pour une classification), les différents modèles de prédiction possibles ne seront pas facile à différencier (comme le montre la figure 1 page suivante).

La métrique :

Afin de pouvoir mesurer quantitativement les performances de notre modèle, une métrique nous a été fournis, qui nous permet d'obtenir un score entre 0 et 1.

Voici la formule pour cette métrique, dites des "r carrés" :

$$score = 1 - \frac{moy\left((predict - solut)^2\right)}{var\left(solut\right)}$$

On voit donc que pour optimiser la métrique, il faut diminuer l'écart entre la prédiction et la solution réelle (ce qui est tout à fait logique). La métrique utilise ici la moyenne des carrés des erreurs de prédiction (le score est donc plus sévère que pour une métrique sans les carrés).
On voit aussi que plus les solutions sont diverses, moins la métrique est "punitive" pour les erreurs de prédiction.

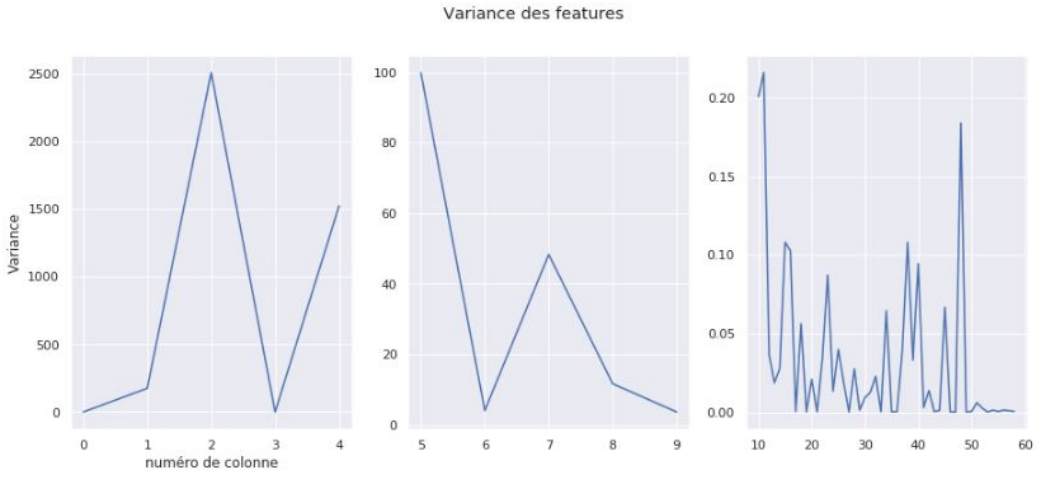
Approche choisie :

Après étude des données, il nous est apparu que nombre d'entre elles contenaient des informations inutiles et nous avons donc prévu d'effectuer une réduction de dimension grâce à des études de variances, et grâce à des regroupements de données avec Sklearn.cluster et Seaborn.
Ensuite, nous chercherons le meilleur modèle et les meilleurs hyper-paramètres grâce à différents modes de visualisation des scores et des variances de modèles.

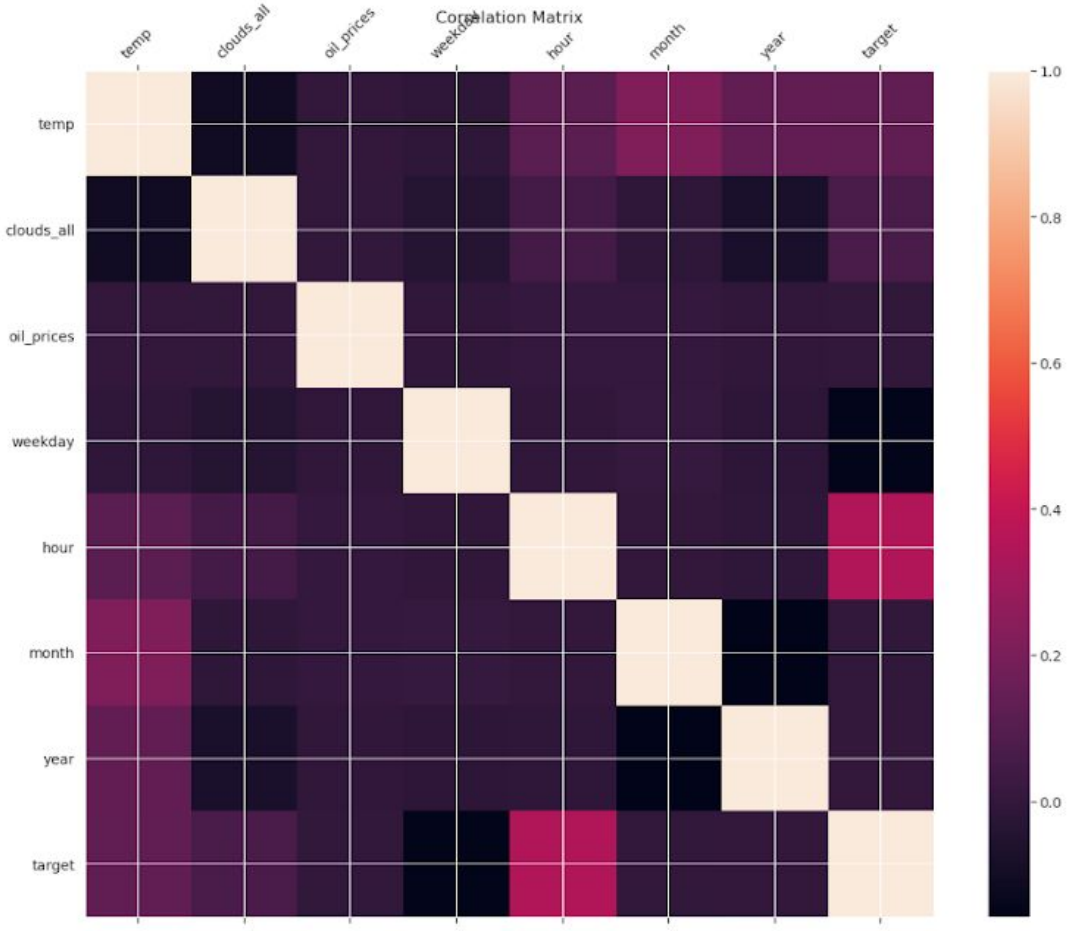
Algorithmes étudiés

Preprocessing

Pour le preprocessing, nous avons décidé de prendre un combo de deux méthodes de scikit learn : VarianceThreshold et Robust_Scale. La première méthode consiste à supprimer les colonnes dont la variance est trop élevée, et est utilisée avec threshold = 0.25 selon le graphique de variance des features juste en dessous. Ensuite on utilise la méthode Robust_scale qui gère les outliers selon des quartiles donnés en paramètre. Cette méthode supprime les données en dehors des quartiles. Avec ce combo, on obtient un training_score de 0.9866.

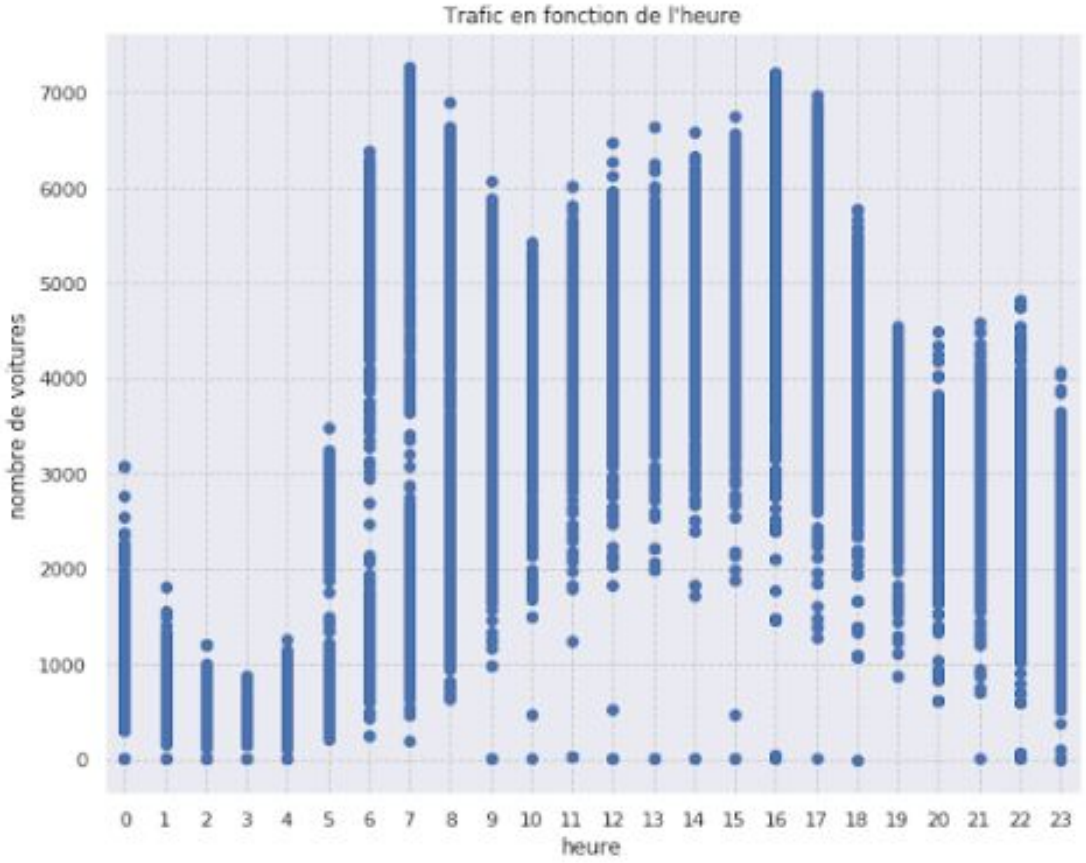


Voici maintenant une étude de l'importance des features via une matrice de corrélation, en ayant déjà éliminé la plupart des features inutiles.
On constate que l'heure et la température sont d'assez loin les deux facteurs les plus importants.




```
Most important features according to the correlation with target
target        1.000000
hour          0.350545
temp          0.131803
clouds_all    0.064201
year          0.002271
month        -0.000533
oil_prices   -0.000706
weekday      -0.150780
Name: target, dtype: float64
```

Un de nos axes de progression pour la suite consistera donc à ajouter des features qui sembleraient plus approprié afin d'améliorer encore l'efficacité du modèle. Par exemple, on peut repérer les heures de pointes en affichant le trafic en fonction de l'heure pour créer une colonne "heure de pointe" booléenne. Voici ce que l'on obtient lorsqu'on affiche un tel graphique :

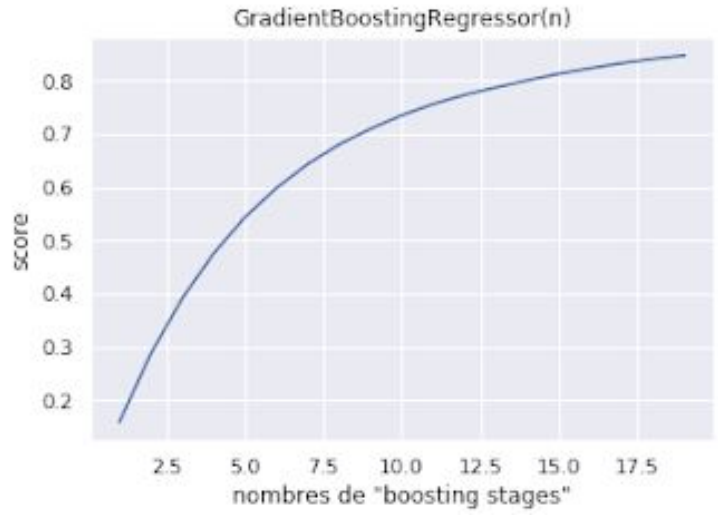


Modélisation :

Après avoir essayé différent algorithmes, nous sommes arrivés à la conclusion que GradientBoostingRegressor était notre meilleur algorithme car il s'exécute rapidement et nous a donné le meilleur résultat.
Pour éviter de tester chaque modèle un par un dans différents fichiers README.ipynb, nous avions créé une fonction test_score qui effectue la cross-validation avec son incertitude et calcule aussi le temps d'exécution pour un modèle donné.
Cette méthode nous a permis de gagner beaucoup de temps car certains modèles pouvaient prendre énormément de temps.

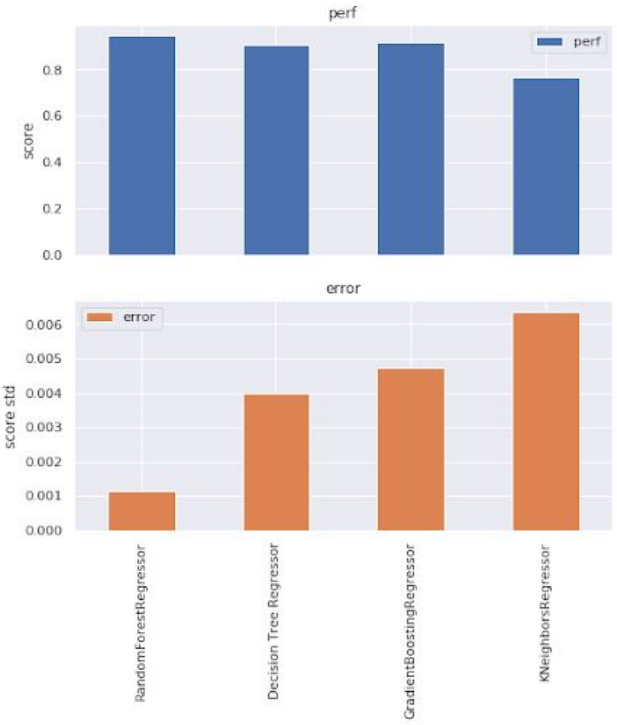
Dans un premier temps nous avions modifier la class model fournie et avons essayé cet algorithme sans hyperparamètres.
Nous avons obtenu un score de 0,92 qui est très proches du score obtenu par le modèle fourni. Par la suite nous avons utilisé la méthode GridSearchCV qui permet de tester différentes valeurs sur nos hyperparamètres. Cette méthode nous a permis d'obtenir un score de 0,96 qui est nettement supérieur au modèle de base.

Figure 1 : Score en fonction des hyper-paramètres :



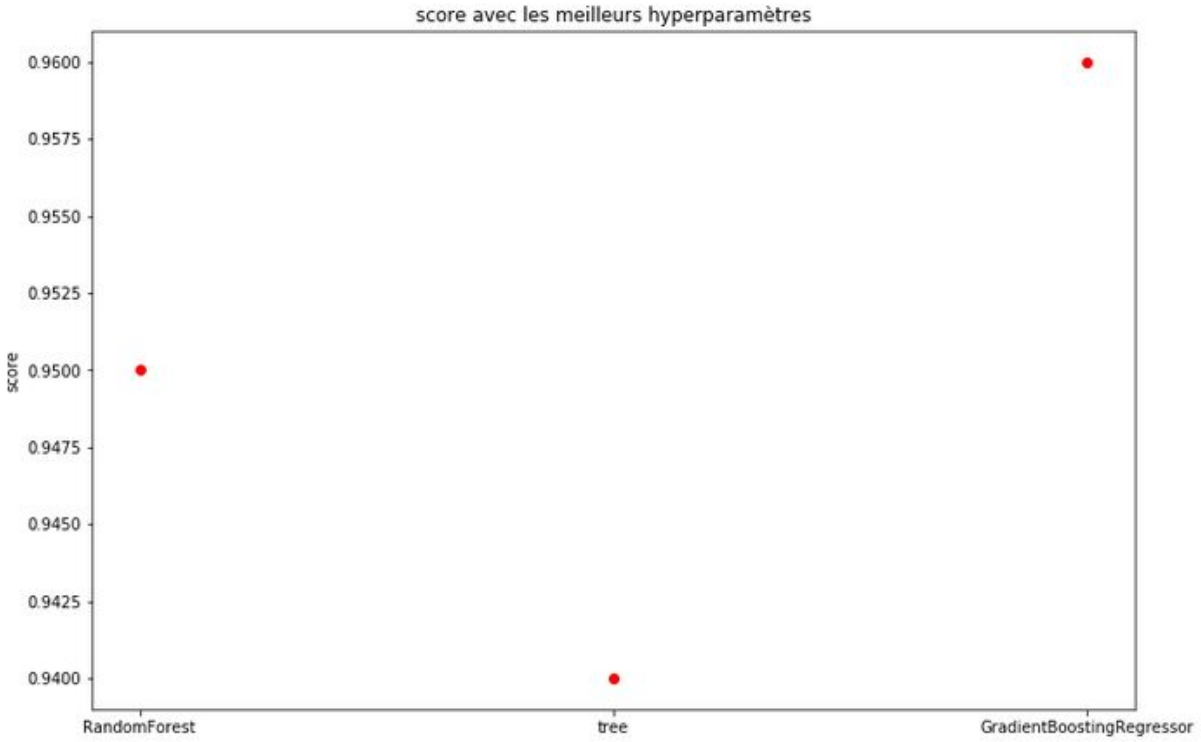
Les hyper-paramètres allant probablement jouer un grand rôle dans le choix du modèle et l'amélioration du score, nous avons voulu être capable de représenter le score d'un modèle en fonction de l'évolution d'un hyper-paramètre. Ici on remarque que le score augmente de façon logarithmique avec n_estimator. De plus, en augmentant ce paramètre, le score de ce modèle dépasse celui du modèle d'origine, d'où ce choix.

Figure 2 : Comparaison des performances



Il nous paraissait intéressant d'afficher le score calculé pour différents modèles afin d'aider à déterminer le plus efficace, ainsi que la variance correspondante, car un modèle performant mais très imprécis n'est pas forcément le meilleur choix.

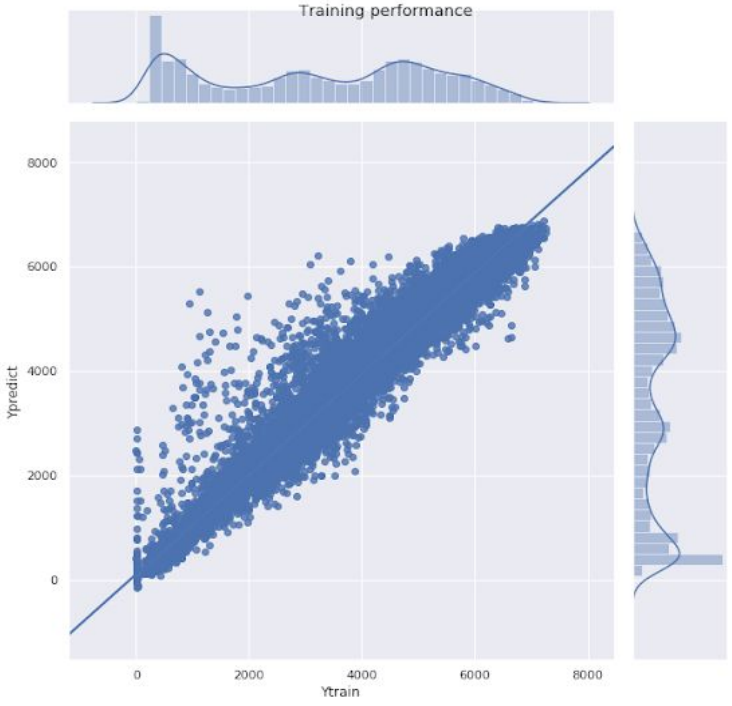
On remarque que le modèle implémenté dans le programme squelette d'origine (RandomForest) a le meilleur score et une variance très faible. Le modèle des KNeighbors semble quant à lui peu recommandé pour le remplacer .
Mais il restait ici à régler les hyper-paramètres. Nous avons donc choisi les trois meilleurs modèles et les avons confronté en ayant cherché leur meilleurs paramètres :

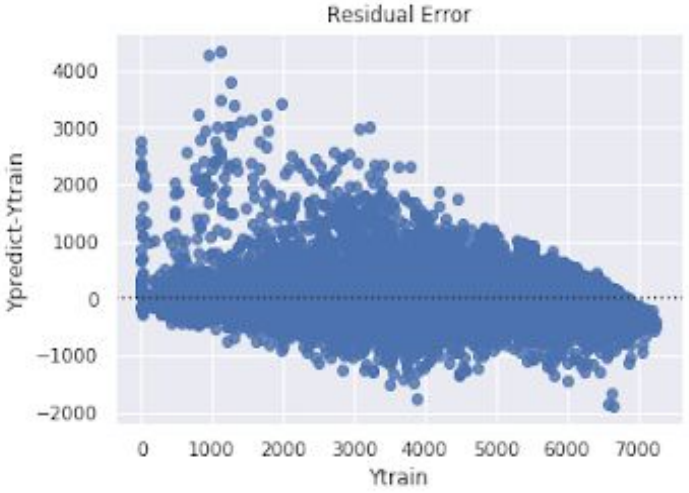


GradientBoostingRegressor est alors la meilleure méthode.

Résultats

Figure 3 : Fit et erreur résiduelle





Pour étudier la performance d'un modèle donné, nous avons affiché les données d'entraînement en fonction de la prédiction. La perfection serait la droite x=y représentée ci-contre.

Voici à gauche le résultat obtenu pour le modèle utilisé. L'allure globale du schéma montre que la prédiction est cohérente, bien qu'elle ne soit pas parfaite. On remarque que l'on peut également avoir une bonne visualisation de la densité de présence des données aux différentes positions, ce qui peut aider à comprendre lesquelles sont mal-interprétées. A droite, voici la position de chaque donnée dans un graphique (Ytrain,Ypredict) par rapport à la droite de prédiction. Cette visualisation est bien utile pour juger de la qualité de chaque modèle (elle donne également une idée de la densité du positionnement des données).
Grâce au graphique des erreurs résiduelles, on remarque que la majorité des données mal classées se situent dans les volumes de trafic les plus bas (en dessous de 2000), et sont classées beaucoup trop élevées. Il est difficile de déterminer quelles données exactement posent problème, mais ceci peut permettre d'améliorer encore le preprocessing.

Voici le score que l'on obtient finalement avec le modèle explicité précédemment :

RESULTS

#	User	Entries	Date of Last Entry	Prediction score ▲	Duration ▲	Detailed Results
1	MOTO	13	04/01/20	0.9589 (1)	0.00 (1)	View
2	Caravan	5	04/04/20	0.9570 (2)	0.00 (1)	View
3	Truck	33	04/04/20	0.9497 (3)	0.00 (1)	View
4	scooter	40	04/04/20	0.9477 (4)	0.00 (1)	View
5	AUTOCAR	27	04/04/20	0.9476 (5)	0.00 (1)	View
6	Velo-Xporters	5	04/03/20	0.9471 (6)	0.00 (1)	View
7	AUTOBUS	11	03/21/20	0.9467 (7)	0.00 (1)	View
8	Segway	7	02/28/20	0.9467 (7)	0.00 (1)	View
9	Taxi	40	02/21/20	0.9467 (7)	0.00 (1)	View
10	pavao	22	02/07/20	0.9467 (7)	0.00 (1)	View
11	automobile	28	03/31/20	0.1661 (8)	0.00 (1)	View
12	medichal	15	10/15/19	-2.6512 (9)	0.00 (1)	View
13	xporters	1	10/12/19	-2.6512 (9)	0.00 (1)	View

Conclusion

Originalité :

Nous pensons que notre code se démarque des autres par sa recherche et son efficacité. Nous n'avons que 5 entrées de code sur codalab car nous essayons de faire beaucoup de tests en interne pour modifier le score au maximum. Notre approche du preprocessing est relativement commune mais l'utilisation de 2 méthodes de preprocessing assez différentes est originale. Concernant la partie modélisation elle est assez commune à savoir trouver un algorithme en rapport avec le problème et l'améliorer à l'aide de méthode comme GridSearchCV. On a également effectué une fonction test_score qui nous a permis "d'automatiser" les tests et ainsi gagner un temps important.

Par son approche différente de la programmation en python et la recherche des solutions par soi-même, l'UE miniprojet est pédagogiquement intéressante. Nous n'avions quasi jamais fait d'introduction au machine learning (pour les gens qui n'ont pas fait l'UE IA qui est une UE optionnelle) et miniprojet n'est pas une UE des plus simples à appréhender pour des néophytes à la fois du python et de l'IA. Cependant l'utilisation et l'apprentissage de cette partie du python reste une nouvelle connaissance nécessaire au cursus informatique universitaire.

Message a la prochaine promo de L2 informatique : N'hésitez pas à poser des questions à vos professeurs pour toute interrogation sur le cours de "soutien" (allez à ce cours, 2h par semaine pour apprendre le machine learning c'est pas assez) et prenez l'UE python au S3. De plus apprenez à chercher sur internet car la plupart des clés à la résolution du problème sont disponibles, savoir chercher vous sera indispensable. Respectez vos deadlines avec une marge d'erreur significative afin d'avoir des retours des profs possibles et des modifications de dernière minute.

Références :

[1]sklearn:
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html
https://scikit-learn.org/stable/modules/preprocessing.html
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.LocalOutlierFactor.html#sklearn.neighbors.LocalOutlierFactor
[2]wikipédia:
https://en.wikipedia.org/wiki/Gradient_boosting

[3]sklearn:
>GridSearchCV
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
>RandomizedSearchCV
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html

[4] Le code du fichier README.ipynb ainsi qu'une fonction pour calculer le temps d'exécution:
>Time
https://docs.python.org/2/library/time.html

[5]seaborn
>Jointplot, Residplot
https://seaborn.pydata.org/generated/seaborn.jointplot.html

[6]Notes de Classe L2 mini-projets. Isabelle Guyon, 2019-2020.