FIGS, un nouveau modèle de régression

Mémoire du projet OpenClassrooms

Développez une preuve de concept

David Depouez

Aout 2023

1 Introduction

Lors du troisième projet, *Anticipez les besoins en consommation de bâtiments*. Le but, était de modéliser la consommation énergétique ainsi que la quantité de gaz à effet de serre émis. Pour résoudre ce problème de régression, on utilisait des algorithmes d'apprentissage supervisé.
Lors de la présente étude, on va reprendre le travail précédent en essayant une nouvelle méthode de régression prometteuse à savoir *FIGS*[1].

2 *FIGS*

On va voir ici une description succincte de *FIGS*. Veuillez consulter [2] pour une analyse plus poussée de l'algorithme.
En principe donc, *FIGS* reprend l'algorithme *CART* utilisé pour faire poussé des arbres de décisions binaires mais il l'étend afin de faire croitre plusieurs arbres.
Dans *CART*, à chaque itération, le nouveau split choisi parmi toutes les possibilités est celui qui va minimiser la fonction de cout

$$J = \frac{n_G}{n} MSE_{Gauche} + \frac{n_D}{n} MSE_{Droite}$$

avec MSE l'erreur quadratique moyenne

$$MSE_{Noeud} = \frac{1}{n_{noeud}} \sum_i (y_i - \bar{y}_{noeud})^2$$

Lors de l'apprentissage de *FIGS*, il peut à chaque itération, choisir de faire grandir un arbre déjà existant comme *CART* ou bien de créer un nouvel arbre. Pour choisir le meilleur split, il substitut au vecteur y de la cible, le vecteur des résidus

$$r_i = y - \sum_{Abres} f_{prediction}(x_i)$$

et choisit le split qui minimise la fonction de cout. L'intérêt des sommes d'arbres, réside dans la résolution des duplications qui peuvent survenir lorsque des additions sont présentes. Afin d'illustrer ceci, prenons l'exemple de modèle suivant à trois variables contenant une addition:

$$y = 1_{x_1>0} + 1_{x_2>0} \times 1_{x_3>0} \tag{1}$$

Lors de l'apprentissage, les modèles *CART* et *FIGS* vont au final aboutir à ce qu'on peut voir sur la Figure 1. On distingue clairement sur la droite de la figure,

[1] *FIGS* pour Fast Interpretable Greedy-Tree Sums.

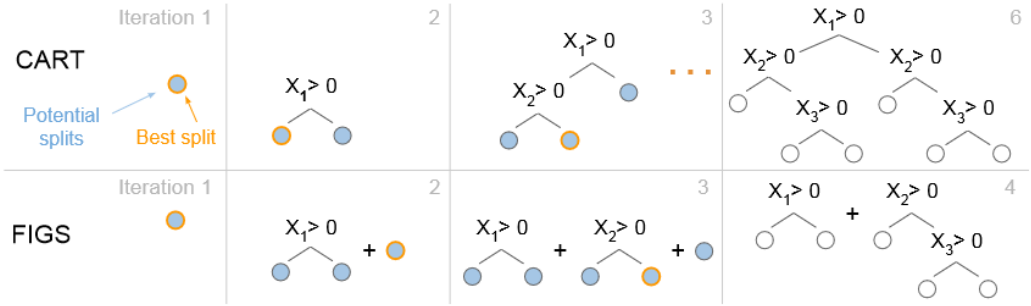

Figure 1: Exemple d'arbres construits par *CART* et *FIGS* correspondant à l'équation 1 (Crédit image : Y.S.Tan et al [2]).

qu'à cause de l'addition, il y a une duplication du deuxième arbre de *FIGS* dans *CART*.
Les méthodes ensemblistes à base d'arbres de décisions, ont aussi plusieurs arbres. Mais ici, les arbres, sont en compétitions les uns avec les autres et ne croissent pas indépendamment. Des méthodes comme les *random forest* par exemple, ne résolvent pas le problème lié à l'addition. De plus *FIGS* présente l'avantage de fournir un modèle qui reste interprétable. On peut néanmoins faire du *bagging* sur *FIGS*.

3 Code utilisé

Afin d'utiliser *FIGS*, on va se baser sur le package python *imodels* [3]. Le code [2] reprend les mêmes interfaces que celles du package *Scikit-Learn* et est globalement compatible avec lui. Il propose différentes classes [4] afin d'instancier des objets *FIGS* autant pour la régression que pour la classification.
Des classes permettant de faire de la cross validation ainsi que de la recherche de paramètres optimaux, un peu à la manière d'un *GridSearchCV* dans *Scikit-Learn* sont également présentes.
Le projet étant encore relativement jeune, il souffre d'un manque de documentation concernant les interfaces et il n'est pas toujours évident de comprendre la signification de certains attributs ou méthodes de classes.

[2] *imodels* version 1.4.0.

4 Les Données

Comme base de travail, on va utiliser la version du dataframe nettoyé lors du projet initial. Suite aux résultats de la précédente étude et afin d'optimiser les performances, on va supprimer de ce dataframe les features identifiées comme non pertinentes.
On se contentera ici, de l'analyse sur la consommation d'énergie et supprimerons donc la cible que constitue les gaz à effet de serre. On fera par contre, comme précédemment, des modélisations avec et sans l'*énergie star score* afin de jauger de son importance. Sur la Figure 2 on peut voir un exemple du dataframe obtenu après ces modifications.

	YearBuilt	NumberofBuildings	NumberofFloors	PropertyGFATotal	ENERGYSTARScore	SourceEUI(kBtu/sf)	SiteEnergyUse(kBtu)	SteamUse(kBtu)	Electricity(kBtu)	NaturalGas(kBtu)
0	1927.0	1.0	12.0	88434.0	60.0	182.500000	7.226362e+06	14.510597	15.188220	14.059596
1	1996.0	1.0	11.0	103566.0	61.0	176.100006	8.387933e+06	-2.302585	14.991963	15.153299
2	1969.0	1.0	41.0	956110.0	43.0	241.899994	1.551976e+07	16.886654	16.227627	14.216834
3	1926.0	1.0	10.0	61320.0	56.0	216.199997	6.794584e+06	14.610513	14.833969	14.409507
4	1980.0	1.0	18.0	175580.0	75.0	211.399994	1.417261e+07	-2.302585	15.496079	15.153299

Figure 2: Premières lignes du Dataframe réduit.

Les corrélations entres les différentes variables sont illustrées sur la Figure 3. On remarque que la cible (SiteEnergyUse) est très fortement lié à l'électricité. Cette dernière, ainsi que d'autres variables intéressantes sont représentées dans les diagrammes de points de la Figure 4.

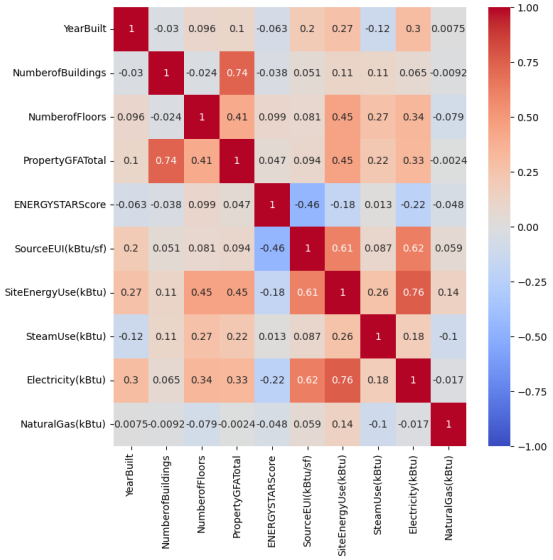

Figure 3: Tableau des corrélations deux à deux entre variables du dataframe réduit.

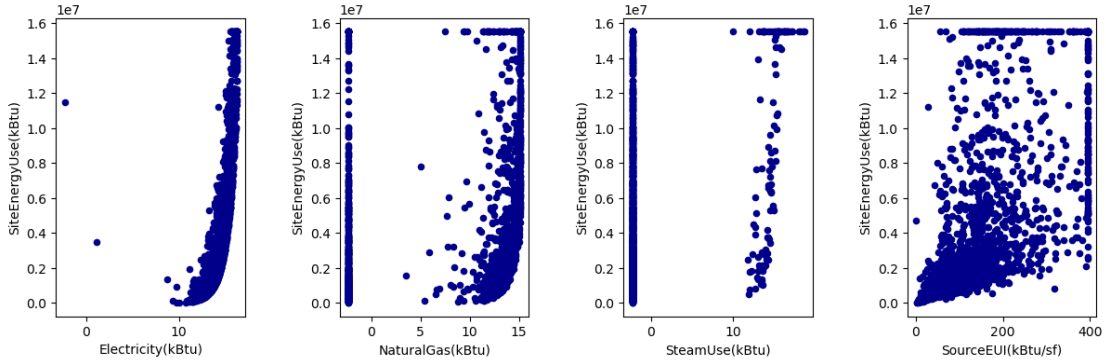

Figure 4: Nuages de points de la cible par rapports aux variables les plus pertinentes.

5 Méthodologie

Pour évaluer *FIGS*, on va le comparer avec d'autres algorithmes et pour chacun d'eux évaluer les trois scores ci dessous.
l'erreur quadratique moyenne, MSE

$$MSE = \frac{1}{n}\sum_{n}(f(x_i) - y_i)^2$$

la racine carré de l'erreur quadratique moyenne, $RMSE$

$$RMSE = \sqrt{MSE}$$

et le Coefficient de détermination, R^2

$$R^2 = 1 - \frac{\sqrt{\frac{1}{n}\sum_n (f(x_i) - y_i)^2}}{\sum_n (y_i - \overline{y})^2} \text{ avec}$$
$$\overline{y} = \frac{1}{n}\sum_{n} y_i$$

On séparera le dataframe avec 80 % des données sur un ensemble d'entrainement et les 20 % restant pour un ensemble de test. Les données seront également centrées et mise a l'échelle comme dans le projet initial.
Deux familles d'algorithmes seront comparées, à savoir les méthodes *simples* qui utilisent des algorithmes interprétables, et les *méthodes ensemblistes* qui sont souvent plus performantes mais perdent le coté interprétable (boites noires).

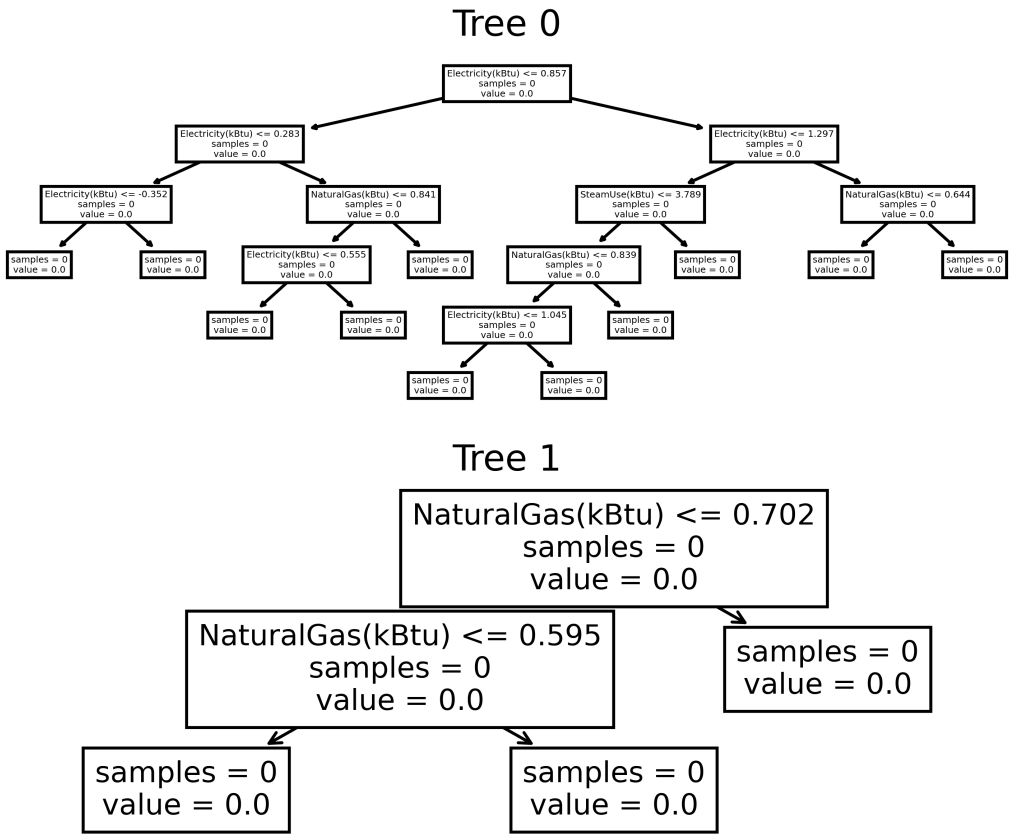

Figure 5: Représentation graphique des arbres déterminés par l'algorithme *FIGS* avec ses valeurs par défaut (cas avec l'énergie star score).

5.1 Les méthodes simples

5.1.1 Raw *FIGS*

On va commencer par implémenter une version simple de *FIGS* adaptée à notre cas avec la classe *FIGSRegressor()* disponible en [4].

```
class FIGSRegressor (max_rules: int = 12, max_trees: int = None,
                     min_impurity_decrease: float = 0.0,
                     random_state=None, max_features: str = None)
```

Le paramètre *max rules* est comme on le voit fixé à douze par défaut et correspond au nombre de splits maximum au total sur tous les arbres. Après entrainement, on obtient les deux arbres de la Figure 5. On remarque qu'une première séparation est faite sur les variables électricité et consommation de gaz naturel.

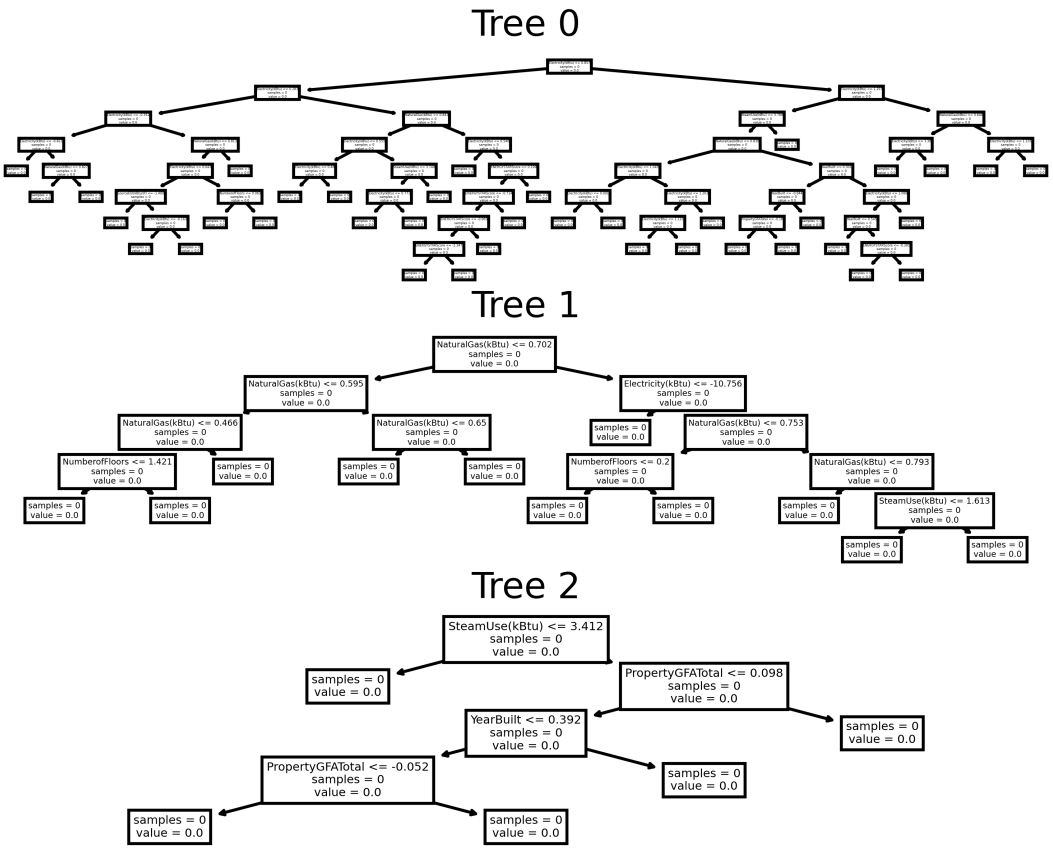

Figure 6: Représentation graphique des arbres déterminés par l'algorithme *FIGS* avec des paramètres optimisés (cas avec l'énergie star score).

5.1.2 *FIGS* CV

On fait ensuite des tests avec *FIGSRegressorCV()*. C'est une classe qui permet d'optimiser les paramètres et de de faire de la validation croisée.

```
class FIGSRegressorCV (n_rules_list: List[int] = [6, 12, 24, 30, 50],
                       n_trees_list: List[int] = [5, 5, 5, 5, 5],
                        cv: int = 3, scoring='r2', *args, **kwargs)
```

Dans ce cas la , on obtient la structure en arbres de la Figure 6. On voit maintenant, trois arbres. Par rapport au cas précédent, un arbre sur la consommation de vapeur est crée.

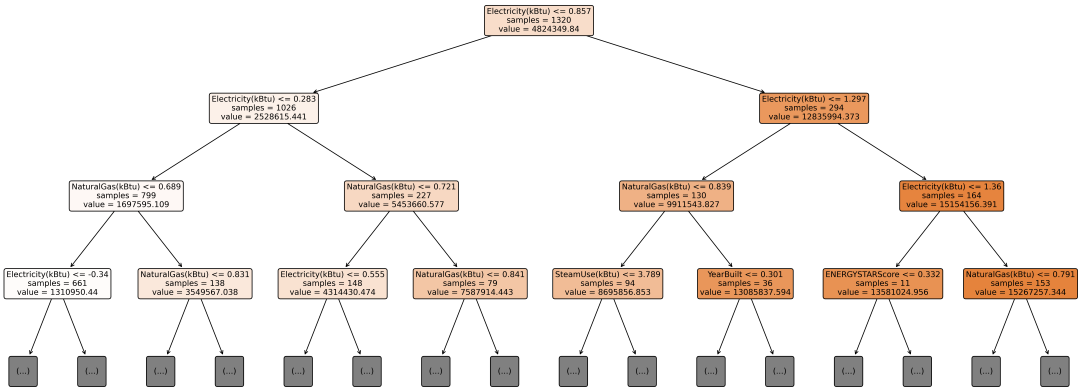

Figure 7: Représentation graphique d'un arbre de décision simple (cas avec l'énergie star score).

5.1.3 Arbre de décision simple

On implémente à titre d'algorithme classique un arbre de décision. La valeur *max depth* à été fixée à quatre afin d'éviter le sur apprentissage. Pour illustration voir la Figure 7.

5.1.4 Arbre de décision optimisé

Avec un *GridSearchCV* on optimise les hyper paramètres de l'arbre de décision pour avoir un équivalent de la méthode FIGS *CV*.

5.2 Les méthodes ensemblistes

5.2.1 Bagging *FIGS*

On va utiliser les paramètres optimaux trouvés lors de l'apprentissage de *FIGS CV* (5.1.2) afin de créer un objet *FIGSRegressor()*. On l'utilisera comme itérateur simple dans un objet *bagging*. On fera varier le nombre d'itérateurs afin de trouver la meilleure combinaison.

5.2.2 Bagging sur arbre de décision

Même chose que pour le *Bagging FIGS* mais avec un objet *DecisionTreeRegressor()* et les meilleurs paramètres trouvés précédemment en 5.1.4.

5.2.3 Random Forest

Pour finir on utilise directement la classe *RandomForestRegressor()* qui revient a faire du *Bagging* sur arbre de décision mais de manière encore plus optimisée. Ici également, on recherchera les meilleurs paramètres à l'aide de *GridSearchCV*.

6 Résultats

Les Tableaux 1 et 2 illustrent les bons résultats obtenus par les méthodes *FIGS* dans les deux cas étudiés.
Le *FIGS* optimisé obtient un meilleur score R^2 que l'arbre de décision optimisé mais il est moins efficace en terme de performance. D'un autre coté, le modèle *FIGS* par défaut (Raw) obtient un score équivalent sur l'ensemble de test par rapport à l'arbre de décision optimisé mais avec des performances bien meilleures cette fois. Si par contre on regarde les deux autres scores (MSE et $RMSE$) alors le *FIGS* dans sa version optimisé se rapproche des performances des méthodes ensemblistes. Il les dépasse même dans le cas sans énergie star score.
Si des additions sont en jeu, alors *FIGS* est censé mieux réagir. Dans notre cas, il est concevable que la consommation d'énergie soit la somme des trois principales sources de consommations que sont l'électricité, le gaz et la vapeur. C'est ce que le modèle optimisé à identifié (Figure 6).
De manière générale, les méthodes ensemblistes, comme on pouvait si attendre, donnent de meilleurs scores que les méthodes simples.
A propos des temps de calculs, ils sont significativement plus longs pour les modèles basés sur *FIGS*. Peut être que cela sera améliorer lors de prochaines releases. Le *Bagging* FIGS qui est particulièrement long à été implémenter ici avec une classe *imodels* embarquée dans une classe *scikit-learn*. A voir si les temps si longs mesurés sont inhérents au modèle ou bien liés à notre implémentation.

Scores comparés

Nom	MSE Train	RMSE Train	R2 Train	MSE Test	RMSE Test	R2 Test	Temps (s)
Raw FIGS	6.94×10^{11}	833283	0.972	9.84×10^{11}	992021	0.963	0
FIGS CV	1.24×10^{11}	351827	0.995	8.16×10^{11}	903056	0.969	14
Raw Decision Tree	7.42×10^{11}	861579	0.970	1.29×10^{12}	1135658	0.952	0
Decision Tree CV	3.41×10^{11}	584287	0.986	9.85×10^{11}	992293	0.963	2
Bagging FIGS	9.36×10^{10}	306016	0.996	6.26×10^{11}	791363	0.977	1183
Bagging Decision Tree	3.03×10^{11}	550270	0.988	8.00×10^{11}	894697	0.970	5
Random Forest	2.85×10^{11}	534309	0.988	8.11×10^{11}	900613	0.970	69

Table 1: Scores des différents algorithmes entrainés avec l'énergie star score.

Scores comparés

Nom	MSE Train	RMSE Train	R2 Train	MSE Test	RMSE Test	R2 Test	Temps (s)
Raw FIGS	6.94×10^{11}	833283	0.972	9.84×10^{11}	992021	0.963	0
FIGS CV	1.27×10^{11}	356131	0.995	7.70×10^{11}	877625	0.971	14
Raw Decision Tree	7.44×10^{11}	862798	0.970	1.29×10^{12}	1136930	0.952	0
Decision Tree CV	2.93×10^{11}	541284	0.988	9.50×10^{11}	974548	0.964	2
Bagging FIGS	9.45×10^{10}	307363	0.996	6.41×10^{11}	800488	0.976	1197
Bagging Decision Tree	2.34×10^{11}	483789	0.990	7.61×10^{11}	872206	0.972	5
Random Forest	3.01×10^{11}	548417	0.988	8.50×10^{11}	922106	0.968	65

Table 2: Scores des différents algorithmes entrainés sans l'énergie star score.

7 Conclusion

L'algorithme *FIGS* se comporte particulièrement bien et confirme les attentes placées en lui. Reste à suivre les futures applications et l'amélioration éventuelle des performances de son implémentation.

References

[1] Berkeley Artificial Intelligence Research *FIGS: Attaining XGBoost-level performance with the interpretability and speed of CART* - Post Article, 2022.

[2] Y.S.Tan et al. *Fast Interpretable Greedy-Tree Sums* - arXiv:2201.11931v3, 2023.

[3] C. Singh *Imodels package*.

[4] C. Singh *FIGS API from imodels package*.

[5] A. Géron. *Hands-on Machine Learning with Scikit-Learn, Keras and Tensorflow.* - O Reilly, 2019.