UNIVERSITÉ PARIS-DAUPHINE

Apprentissage Statistique - Projet n°3

Prédiction du coût de construction et / ou prix de ventes de bâtiments

Mathias Marciano
Prescillia Zeitoun
Ornella Fettaya
Meital Sitbon
Elsa Mimouni
Gary Bloch

Professeur : Mme Y.Liu

22 Mai 2022

Dans ce projet nous allons utiliser les méthodes de Machine Learning afin de prédire le prix actuel de vente appelé V-9 dans le jeu de données.

Description du jeu de données

https://archive.ics.uci.edu/ml/datasets/Residential+Building+Data+Set

Ce jeu de données est composé de 105 colonnes (ou variables) et de 372 lignes. Les colonnes sont réparties en deux catégories :

- PROJET PHYSIQUE ET FINANCIER
- VARIABLES ET INDICES ÉCONOMIQUES

Pour des raisons de temps de calculs nous allons travailler uniquement sur la première catégorie de colonne qui sont les données financières. Il nous reste alors 14 colonnes dans notre jeu de données:

- V-1 Localité du projet définie en termes de codes postaux
- V-2 Superficie totale du bâtiment
- V-3 Zone du terrain
- V-4 Coût total de construction estimé préliminaire sur la base des prix au début du projet
- V-5 Coût de construction estimé préliminaire basé sur les prix au début du projet
- V-6 Coût de construction estimé préliminaire équivalent basé sur les prix au début du projet dans une année de base sélectionnée
- V-7 Durée de construction
- V-8 Prix de l'unité au début du projet par m2
- V-9 Prix de vente réels en sortie

Toutes les variables sont des variables continues ainsi que notre target. Nous avons donc ici un problème de régression.

L'objectif est de prédire le **prix de vente réel** qui est la colonne V-9 dans notre jeu de données.

Exploratory Data Analysis (EDA)

Dans un premier temps, nous allons procéder à l'Analyse exploratoire des données (EDA) : c'est une des étapes majeures permettant d'affiner l'ensemble de données afin d'en apprendre plus sur les caractéristiques clés des données. Pour cela nous allons utiliser les packages Pandas, NumPy, Statistical Methods et Data visualisation.

On commence par réaliser le *tableau des corrélations* du jeu de données :

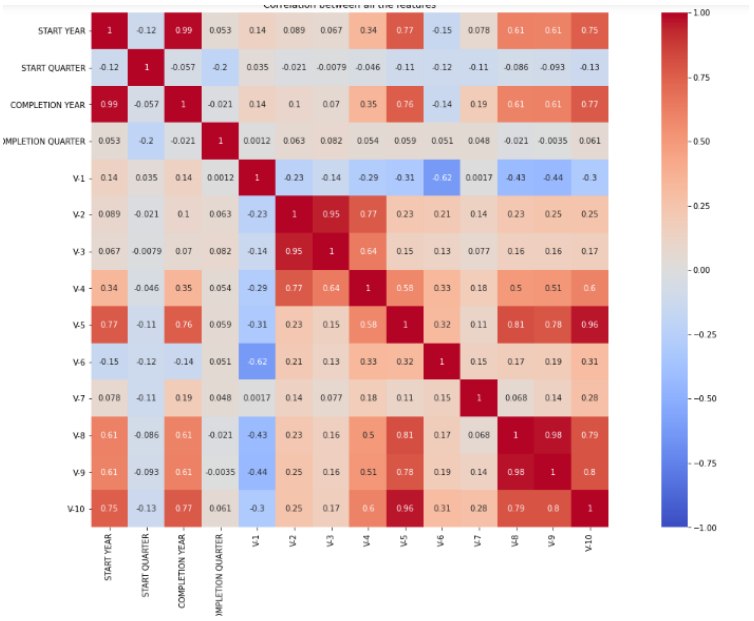
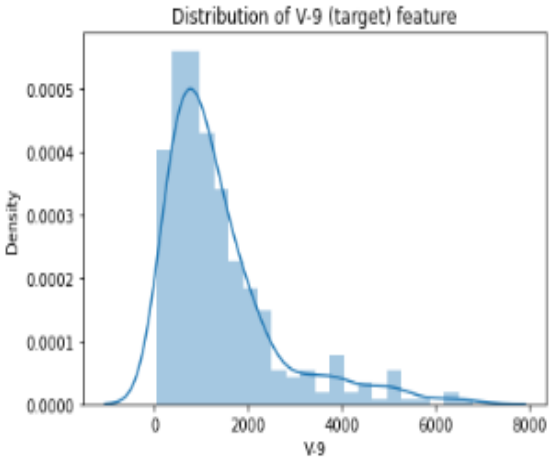


Tableau des corrélations

On peut remarquer plusieurs points intéressants sur ce tableau des corrélations :

- Il y a une grande corrélation entre les variables START YEAR et COMPLETION YEAR
- Il y a une grande corrélation entre les variables V-9 (notre target) et V-8
- Il y a une grande corrélation entre les variables V-2 et V-3

Il n'est pas utile de garder des variables qui ont de fortes corrélations entre elles. Nous allons donc en sélectionner par la suite et en supprimer d'autres.



Passons maintenant à *l'analyse de notre targe*t V-9 et sa distribution :

Comme nous l'observons, cette dernière ne semble pas suivre une loi Gaussienne avec notamment un Skewness (asymétrie) de 1.877 et un Kurtosis (aplatissement) de 3.750. La moyenne est de 1387.43 et le maximum est de 6800.

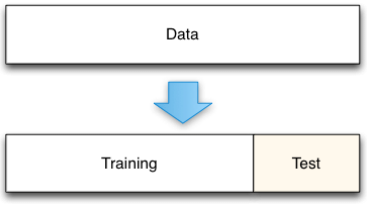
Pour conclure, il n'y a pas de valeurs manquantes dans tout notre dataset.

Modèles de Machine Learning sur données brutes :

Notre métrique pour évaluer le modèle sera le score. Lorsque le score est appelé sur les régresseurs, le coefficient de détermination - R2 est calculé par défaut. Comme dans les classificateurs, la méthode des scores est simplement un raccourci pour calculer R2 car elle est couramment utilisée pour évaluer les performances d'un régresseur.

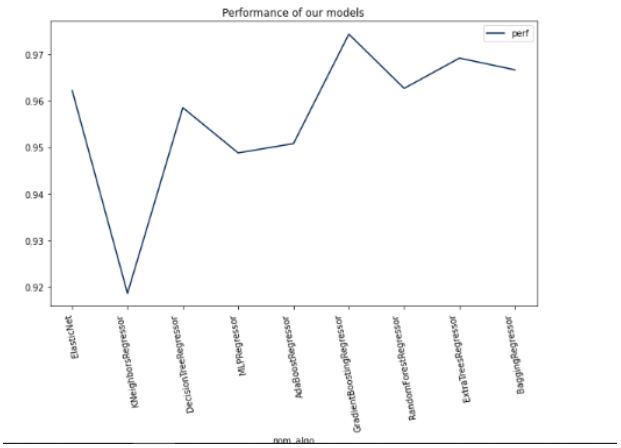
Il existe de nombreux modèles pour répondre à des problèmes de régressions. Pour ce projet, nous avons à la fois utilisé les modèles vu en cours, tels que LinearRegression, KNeighborsRegressor, et également réalisé de nombreuses recherches sur d'autres modèles tels que ElasticNet, DecisionTreeRegressor, MLPRegressor, AdaBoostRegressor, GradientBoostingRegressor, RandomForestRegressor, ExtraTreesRegressor, BaggingRegressor.

Nous allons diviser notre jeu de données en 2 parties. Une partie pour l'entraînement (Train) qui représentera 70% de notre jeu de données, et une partie de test (Test) qui représentera 30% du jeu de données.



Maintenant que nous avons choisi les différents modèles que nous souhaitons utiliser, et que notre jeu de donnés est divisé en deux, nous pouvons entraîner les différents modèles et obtenir le score R^2.
On obtient les résultats suivant:

nom_algo	perf
LinearRegression	0.948547
ElasticNet	0.948698
KNeighborsRegressor	0.893965
DecisionTreeRegressor	0.920040
MLPRegressor	0.946511
AdaBoostRegressor	0.937955
GradientBoostingRegressor	0.978037
RandomForestRegressor	0.947973
ExtraTreesRegressor	0.970119
BaggingRegressor	0.961011



Tous ces modèles nous proposent alors un score supérieur à 90%. Cependant, le modèle de **Gradient Boosting** se démarque des autres avec un score supérieur à 97% sur l'ensemble de test.
On remarque par ailleurs, que le modèle de **KNN**, est le modèle le moins performant. On peut être amené à penser que cet algorithme qui s'appuie sur les plus proches voisins ne nous propose pas une performance au même niveau que les algorithmes ensemblistes.

Pour en apprendre plus sur ce modèle, voici la documentation que nous avons utilisé :
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html

Feature Engineering

Le Feature Engineering est le processus de sélection, de manipulation et de transformation des données brutes en fonctionnalités utilisables. Nous allons donc essayer d'améliorer nos modèles avec quelques transformations sur nos données comme l'imputation, la gestion des valeurs aberrantes, le binning ou la transformation Standard Scaler.

Pour ce jeu de données nous allons donc retirer les colonnes "START YEAR", "V-3","V-1" car comme nous l'avons vu précédemment avec le tableau des corrélations, elles sont extrêmement corrélées avec d'autres features et n'apportent donc pas d'utilité à notre modèle.

Nous allons également passer par la fonction Standard Scaler() qui va nous permettre de mettre à la même échelle les variables.

Une fois ces modifications effectuées nous allons pouvoir relancer notre modèle de Gradient Boosting et voir que notre score a gagné environ 0.3% de précision en plus.

Recherche des meilleurs Hyperparamètres

Les modèles d'apprentissage automatique ont des hyperparamètres que l'on doit définir afin de personnaliser le modèle en fonction du jeu de données. Il existe souvent des heuristiques générales ou des règles empiriques pour configurer les hyperparamètres. Nous allons ici utiliser un GridSearch.

Le modèle de Gradient Boosting fait partie de la classe des ensembles et possède de nombreux hyperparamètres. En voici quelque uns:

- *loss*
- *learning_rate*
- *n_estimators*
- *subsample*
- *criterion*
- *max_depth*

En utilisant un Grid Search nous allons pouvoir tester les hyperparamètres et les valeurs suivantes:

- 'learning_rate' : [0.1,0.01],
- 'n_estimators' : [100,1000],
- 'subsample' : [0.1,0.2,0.3],
- 'max_depth' : [1,5,10,20]

Nous gardons ensuite ceux qui apportent au modèle un maximum de précision.

Une fois effectué (voir code), nous avons donc ces derniers:

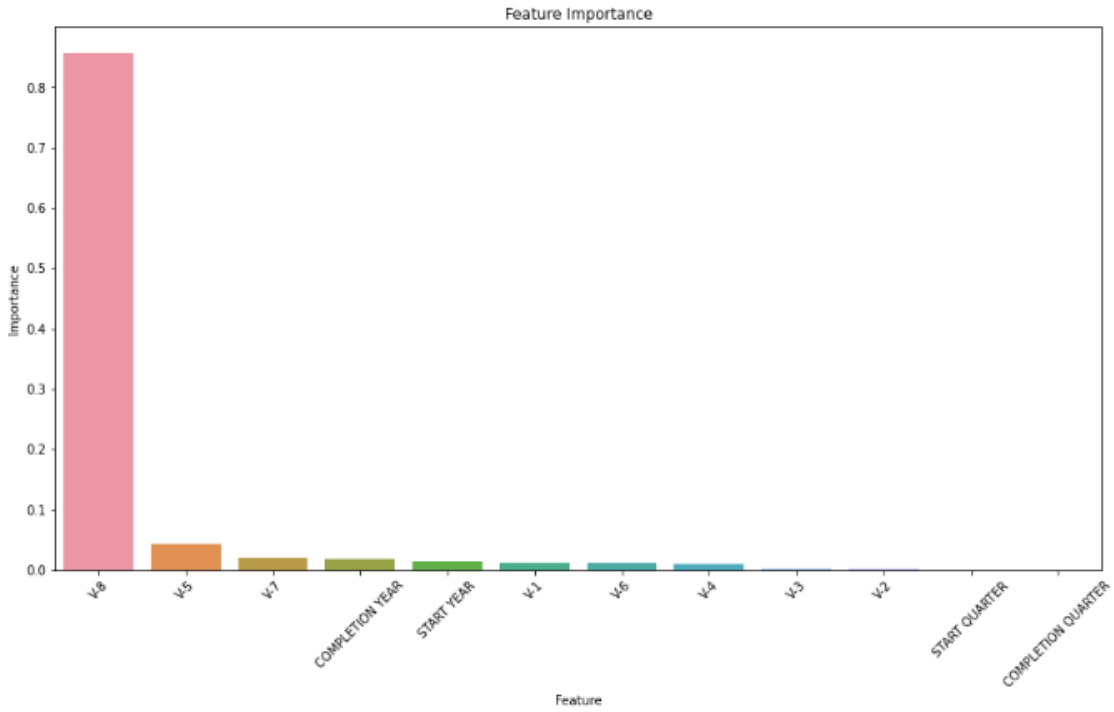
```
CV_rfc.best_params_
```

```
{'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 1000, 'subsample': 0.3}
```

Une fois relancé, notre modèle gagne encore en efficacité avec 0.04% de précision supplémentaire.

Conclusion

Pour finir nous allons analyser l'importance ou le poids des features sur notre target.
Nous avons alors les résultats suivants:



Comme on s'en doutait, la variable V-8 (Prix de l'unité au début du projet par m2) est la variable qui a le plus de poids dans notre modèle afin de prédire les prix de vente réels (sortie).
Avec un score de plus de 97 % sur l'ensemble de test, notre modèle de Gradient Boosting est pertinent.

Pour aller plus loin

La phase de Feature Engineering est la façon la plus efficace d'améliorer un modèle de Machine learning. Ici on aurait pu par exemple créer de nouvelles variables à partir de celles présentes ou encore la gestion des valeurs aberrantes (outliers) qui amènent du bruit dans notre modèle. Enfin il faut faire attention à l'overfitting qui est le meilleur moyen pour un modèle de se tromper lors de son apprentissage sur notre jeu d'entraînement. Il est de plus possible de jouer davantage avec les hyperparamètres afin d'améliorer la précision de notre modèle.