Rapport Mini-projet

Projet : Xporters
Nom du groupe : Autobus
Membres du groupe:
Marion Pobelle <marion.pobelle@u-psud.fr> groupe 1bis
Garance Roux <garance.roux@u-psud.fr> groupe 1
Romain Soliman <romain.soliman@universite-paris-saclay.fr> groupe 1bis
Elie Raspaud (ne fait plus l'UE) <elie.raspaud@u-psud.fr> groupe 1bis
Aimane Benammar <aimane.benammar@universite-paris-saclay.fr> groupe 2
Lucas Foissey <lucas.foissey@u-psud.fr> groupe 1bis
Responsable : Herilalaina Rakotoarison <herilalaina.rakotoarison@universite-paris-saclay.fr>
Challenge URL : https://codalab.lri.fr/competitions/652
Presentation video URL :
https://www.youtube.com/watch?v=JGqbjVWef5o&t=4s
Github repo of the project : https://github.com/Ladiscou/AUTOBUS2

Motivation du choix XPORTERS

Nous avons décidé de travailler sur le projet Xporters. En effet, il s'agissait du seul problème de régression nous ayant été présenté et son contexte actuel a attiré notre curiosité. Le nombre de caractéristiques à gérer ainsi que leurs natures définissent une application unique et proche de la réalité de notre environnement.

Nous étions aussi intéressé d'apprendre par le biais de cet exercice l'utilisation du machine learning dans des cas concrets.

Données et description du problème:

Notre but est de prédire le nombre de véhicules qui traversent l'autoroute suivant certains paramètres. Il s'agit d'un problème de régression, on a donc observé le trafic routier pendant un temps donné puis implémenté un algorithme de machine learning qui nous a permis de prédire le trafic routier à venir sur cette même autoroute. Nous devions faire attention au phénomène de surapprentissage et avons cherché à approcher de 100% le score de notre modèle.

Les paramètres de la prédiction sont constitués d'un set composé de 38 563 données réparties en 59 features, tout cela représentant des conditions influençant le trafic routier telles que l'heure, la météo, le prix de l'essence et bien d'autres.
Les statistiques des données utilisées sont résumées dans le ***Tableau 1*** (cf *Annexe 1*).

Approche de chaque groupe

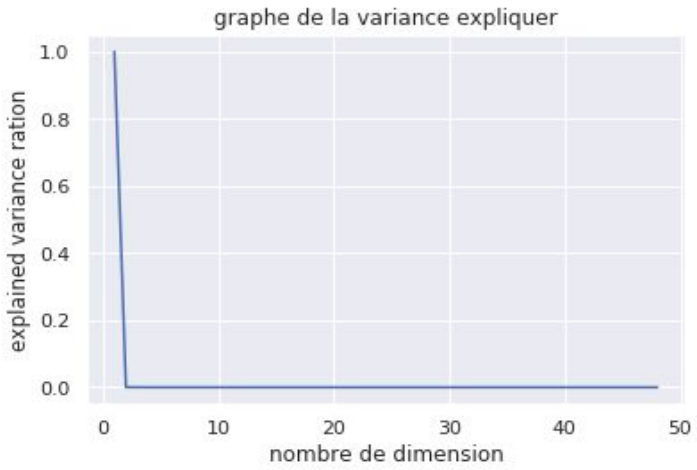
Preprocessing :

Le but de cette partie était dans un premier temps de sortir les outliers (données isolées pouvant perturber l'algorithme d'apprentissage) de notre set de données, puis de réduire la dimension des données pour faciliter le travail dessus par la suite. Nous avons dans un premier temps utilisé *LocalOutlierFactor* ([1], cf *Annexe 2*) qui nous a permis de mettre à jour nos données. Nous avons décidé, après plusieurs tests, d'enlever 8% des données ce qui nous semblait un juste équilibre pour éviter d'en enlever trop. Nous avons ensuite décidé de réduire le nombre de dimension de nos données en utilisant l'outil *PCA* ([2]) qui projette les dimension de façon à fusionner les données utiles entre elles. Pour finir nous avons utilisé SelectKBest([3]) pour calculer le score de l'importance des features du modèle et ainsi obtenir un classement de l'importance des features.

Nous avons par la suite pris la décision de rajouter des dimension à la main à nos données avant de faire toute opération de preprocessing dessus, Nous avons rajouté par exemple la features heure de pointe (qui prend des valeur entre 0 et 1 pour faciliter sa compréhension par PCA) pour cela on évalue l'heure de la journée et le jour de la semaine auxquelles ont été prises les données grâce aux features déjà présentes et cela nous permet de savoir si la donnée a bien été prise pendant une heure de pointe ou non.

Nous avons enfin tenté de calculer le nombres de features optimiser à mettre dans PCA. pour cela on utilise la fonction pca.explained_variance_ratio_ pour obtenir la variance de notre jeux de donnée préprocessé selon le features et nous traçons le graphe suivant :

```
Out[18]: <function matplotlib.pyplot.show(*args, **kw)>
```



Nous ce qui indiquerait de conserver un nombre de feature très faible mais nous avons préféré ne pas considérer ce tableau et ne pas inclure ce processus dans nos résultats finaux.

Modélisation :

Nous avons d'abord testé 5 modèles de régression provenant du module *scikit-learn : NearestNeighbors, RandomForest, RBF-SVM, LinearSVM et DecisionTreeRegressor*. Ensuite, nous avons calculé le score, avec des hyper-paramètres arbitraires, de chacun de ces modèles

sur un ensemble de validation obtenu en partageant notre set grâce à la fonction *train_test_split* ([4]) appliquée aux données. Cela nous a permi de créer un ensemble de validation fictif afin de pouvoir tester notre modèle et prévenir l'over-fitting de manière plus efficace. Le modèle le plus adapté choisi, nous avons étudié ses caractéristiques : fonctionnement, hyper-paramètres et applications étaient l'objet de notre intérêt.

Nous devions ainsi définir quels étaient les hyper-paramètres de notre modèle les plus pertinents à utiliser sur nos données. Afin de mener notre recherche, nous avons décidé d'utiliser la méthode *Random Search ([5])*, car de nombreuses combinaisons d'hyper-paramètres étaient à essayer. Nous avons traversé une phase de recherche afin d'établir une courte liste d'hyper-paramètres intéressants à utiliser pour notre modèle, nous nous sommes documentés sur chacun d'entre eux afin de définir des intervalles de recherche pertinents pour rendre notre algorithme de recherche plus efficace. Les paramètres clés de notre recherche étaient de prévenir les situations dites d'*over-fitting* ([6]) et de réduire notre temps de recherche. L'over-fitting aurait rendu notre modèle inutilisable sur d'autres set de données appliquables au contexte du projet car il aurait été trop centré sur le set de données initialement utilisé et donc pas assez général.

Ainsi, nous avons pu définir quel était le modèle le plus efficace à appliquer à notre problème, ainsi que les hyper-paramètres les plus intéressants à utiliser pour une étude pertinente de nos données.

Lien vers le README contenant les tests :
https://github.com/Ladiscou/AUTOBUS2/blob/master/starting_kit/README_model.ipynb

Visualisation :

Notre partie du travail était de représenter au mieux les résultats. La première approche a été de savoir quels graphiques seraient les plus pertinents, ce qui variait en fonction du projet choisi. Pour cela, nous avons fait une liste simplifiée des algorithmes souvent utilisés lors de projets d'intelligence artificielle. En comparant notre liste avec les consignes demandées, nous avons conclu qu'il fallait faire une classification des données et une visualisation de régression. Le problème était qu'on ne savait pas sur quelles données travailler, ce que nous développerons plus dans la partie des premières approches.

Organisation de notre groupe

Le premier binôme constitué de Lucas (et Elie initialement), a travaillé sur la partie preprocessing de notre projet. L'objectif était de faire un premier traitement sur les données avant de travailler dessus en enlevant les données in-intéressantes et en réduisant leur dimension.

Le second binôme constitué de Marion et de Aimane, a travaillé sur la partie modélisation de notre projet. Leur objectif était de trouver le modèle le plus adapté aux données du problème ainsi que les hyper-paramètres correspondant les plus intéressants.

Le dernier binôme constitué de Garance et Romain, a travaillé sur la partie visualisation de notre projet. Leur objectif était de représenter au mieux les résultats obtenus afin de pouvoir visualiser quelle solution de notre projet était la plus pertinente.

Premiers résultats

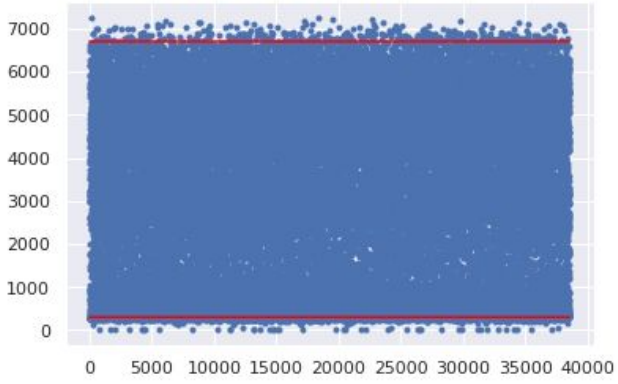
Preprocessing :

On commence par ajouter la feature Heure de pointe:

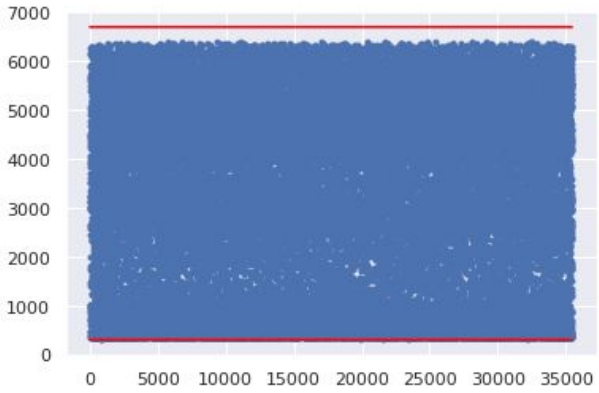
ıt_drizzle	weather_description_thunderstorm_with_light_rain	weather_description_thunderstorm_with_rain	weather_description_very_heavy_rain	crowded	target
0	0	0	0	0.0	759.0
0	0	0	0	0.0	4085.0
0	0	0	0	1.0	3528.0
0	0	0	0	0.0	2636.0
0	0	0	0	1.0	4226.0
...	...	...	...	...	...
0	0	0	0	0.0	2967.0
0	0	0	0	0.0	5958.0
0	0	0	0	1.0	6591.0
0	0	0	0	0.0	4366.0
0	0	0	0	1.0	5240.0

On observe qu'on a rajouté la colonne crowded.

Après avoir observé la répartition des données grâce aux outils de visualisation de pyplot et avoir fait différents tests pour comprendre l'algorithme des k-voisins nous avons décidé de trouver les outliers en utilisant cet algorithme avec en paramètre la moitié du nombre de donnée (on notera que cette fonction étant très gourmande en terme de calculs nous avons créer une fonction plus légère qui divise le jeu de donnée en 4) ce qui a permis d'enlever surtout les données extrêmes comme le montre les graphes ci dessous:



Donnée avant le filtrage (target en y, index en x)



Donnée après le filtrage

Par la suite nous avons décidé de réduire le nombre de dimension de nos données à 48. En effet cela nous permettait de conserver les données les plus intéressantes que nous observions avec linearRegression([7]) qui nous permet de faire certaines opérations sur notre jeu de données notamment une évaluation du score.
Pour finir nous avons fait le classement des features et obtenu un top 10 qui classait les features selon un score d'importance:

	Specs	Score
2	rain_1h	2795.129136
16	weather_main_Rain	122.124013
1	temp	7.614380
11	weather_main_Clouds	7.377375
9	year	7.106427
10	weather_main_Clear	6.165868
48	weather_description_sky_is_clear	5.133682
8	month	2.312206
15	weather_main_Mist	2.250224
38	weather_description_mist	2.250224

On notera que le score obtenu pour rain_1h parait très élevé par rapport à d'autres scores que nous attendions plus élevés notamment le jour de la semaine et l'horaire.

Nous transférons ainsi les données préprocessées à la partie model pour sortir des résultat.

Modélisation :

Les premiers résultats sur les données obtenus par notre binôme à l'aide du fichier README.ipynb ([8]) sont résumés dans le ***Tableau 2*** (cf *Annexe 3*). Ces résultats montrent que le modèle le plus adapté à notre problème est le modèle *Decision Tree*, avec les

hyper-paramètres *min_samples_split=12*, *min_samples_leaf=29*, *max_features=33* et *max_depth=11*.

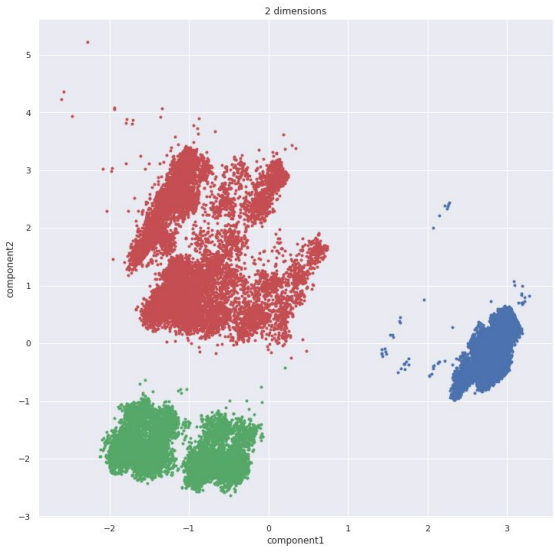
Le score obtenu sur nos données pour ce modèle par cross-validation est de 0.95 ce qui est excellent en prenant en compte la taille de l'ensemble des données d'entraînement et les risques d'over-fitting. Lors de notre recherche de modèle, nous avons été confronté à d'étonnants résultats. En effet, certains modèles obtenaient des résultats aberrants tels que des scores négatifs. Il n'était pas question d'utiliser ces modèles !

Le modèle *Decision Tree* structure les données suivant deux phases : une première phase de construction qui, sur la base d'un ensemble de données, applique un processus récursif de division de l'espace des données. On cherche lors de ce processus, à maximiser la variance inter-classes. Il faut donc faire attention à ne pas fixer une profondeur trop élevée, au risque que l'algorithme soit trop centralisé sur un set de données et donc non-généralisable. La deuxième phase de l'algorithme consiste en la suppression des branches peu représentatives des données dans l'arbre, afin d'avoir un arbre le plus généralisable à d'autres données possibles. Nous avons ainsi pu appliquer notre modèle aux données issues du preprocessing et avons obtenu un score par cross-validation de 0.78 sur celle-ci. Ce score n'est pas optimal, mais il reste très correct. Nous avons calculé la moyenne ainsi que la variance sur le score obtenu afin de vérifier que le phénomène de surapprentissage n'était pas présent dans nos résultats. Avec une moyenne de 0.77 et une variance de 0.13, nous avons conclus que notre modèle ne faisait pas d'over-fitting sur nos données !

Visualisation :

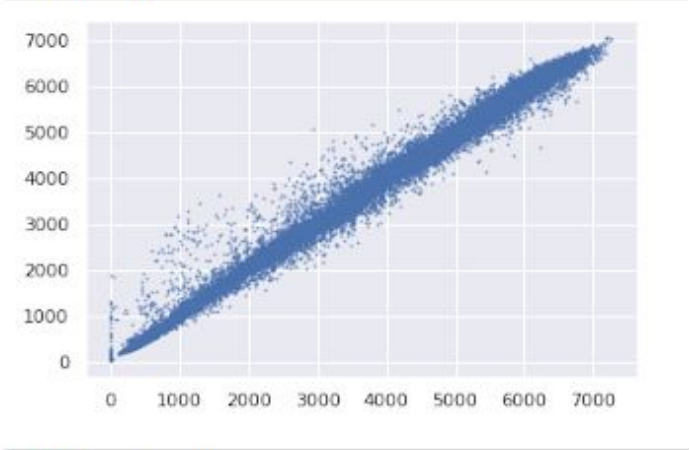
Il s'agissait en partie de représenter les résultats des autres binômes, ce qui était compliqué puisqu'on ne pouvait pas travailler sur le même code en même temps. On a donc décidé de traiter les données dont on disposait sur le fichier de référence (README.ipynb ([8]). Il fallait faire en sorte que l'algorithme fonctionne avec n'importe quel tableau.

Nous avons fait deux graphiques, le **Graphique 1** ci-dessous représente les données classées par l'algorithme K-means([9]). Le tableau donné *data* était de dimension (38563, 60). Afin de traiter ces données, il faut d'abord convertir le tableau *data* en deux dimensions. Nous nous sommes donc inspirés de la fonction PCA ([2]) dont le groupe Preprocessing s'est servi. Comme les valeurs varient entre 10^{-3} et 10^{5} , il faut avant tout normaliser le tableau *data* avant d'appliquer la fonction Principal Component Analysis (PCA) pour éviter des pertes de données à cause de la trop grande différence d'échelle.
Nous avons donc obtenu le graphique suivant :



Graphique 1 : classement des données par la fonction KMEANS

Le second graphique était plus simple à réaliser puisqu'on avait un graphique de référence (**Graphique 2**, voir ci-dessous) déjà implémenté dans le code README.ipynb ([8])

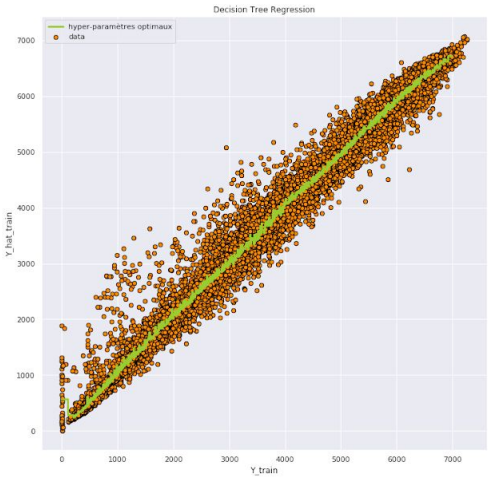


Graphique 2 : graphique de référence

Il ne restait plus qu'à représenter un modèle de régression ajustée en utilisant la fonction DecisionTreeRegressor ([10]). La partie modèle a donné les hyper-paramètres les plus optimisés soient :

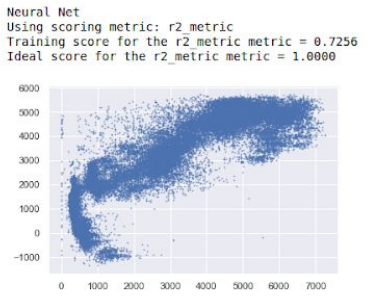
- max_depth = 11
- max_features = 33
- min_samples_leaf = 29
- min_samples_split = 12

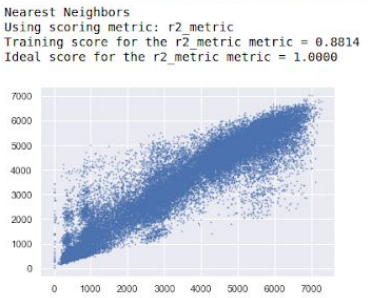
On obtient ainsi le graphique suivant :

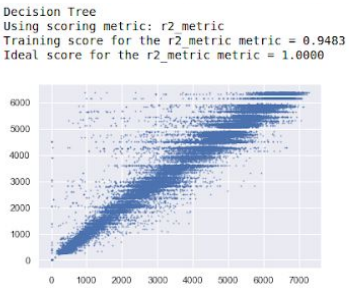


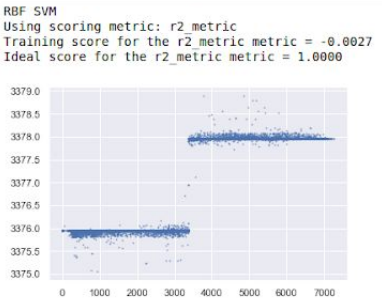
Graphique 3 : courbe obtenue par DecisionTreeRegressor avec les hyper-paramètres optimaux

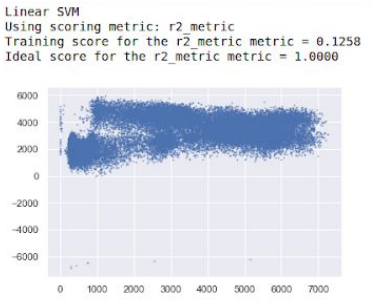
Notre partie a également permis de choisir quel modèle était le plus adapté en représentant chaque modèle sur un graphe :

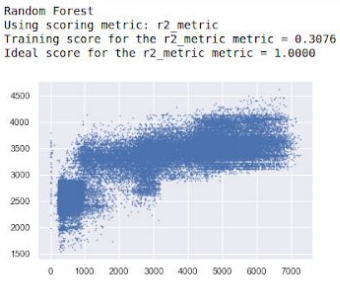












Conclusion

Ce projet, aussi intéressant qu'amusant, nous a tous permis de renforcer nos connaissances du machine learning et nous a appris à améliorer notre méthode de travail lors de projet de groupe. Nous sommes ravis d'avoir participé à ce projet et encourageons toute personne intéressée ainsi que les étudiants des années suivantes à se lancer dans l'aventure !

Annexes

Annexe 1:

Tableau 1 : Statistiques des données

Dataset	Num. Examples	Num. Variables/ features	Sparsity	Has categorical variables ?	Has missing data ?
Training	**38563**	**59**	**0**	**0**	**0**
Valid(ation)	**4820**	**59**	**0**	**0**	**0**
Test	**4820**	**59**	**0**	**0**	**0**

Annexe 2:

Description de l'algorithme de filtrage pour détecter les outliers:

La fonction filter suis donc le processus suivant:
Crée mon outil de LocaloutlierFactor de parametre **k** et **n**
Créer mon tableau **inliers** de -1 et 1 en appliquant l'outil a **data**
For **i** ∈ **inliers**:
  Remplit un tableau **outliers** des indice des **i** == -1
Return outliers

Fonction de mise à jour des données:
Prend en paramètre mon tableau de donnée **data** et mon tableau **outliers**
For **i** ∈ **outliers**:
  Enlève la donnée d'indice **i** dans **data**
Return **data**

Annexe 3 :

Tableau 2 : Premiers résultats

Method	Nearest Neighbors	Linear SVM	RBF SVM	Decision Tree	Random Forest
Training	**0.8733**	**0.1241**	**-0.0029**	**0.9489**	**0.3950**
CV	**0.73**	**0.12**	**-0.00**	**0.93**	**0.32**
Valid(ation)	**0.7424**	**-0.3535**	**-0.0035**	**0.9358**	**0.3844**

References

[1] Unsupervised outlier detection using Local Outlier Factor, bibliothèque sklearn.neighbors scikit-learn developers, 2007 - 2019,https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.LocalOutlierFactor.html#sklearn.neighbors.LocalOutlierFactor

[2] Principal component analysis, bibliothèque sklearn.decomposition scikit-learn developers, 2007 - 2019, https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html?highlight=pca#sklearn.decomposition.PCA

[3] Select features according to the k highest, bibliothèque sklearn.feature_selection scikit-learn developers, 2007 - 2019scores,https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html?highlight=selectkbest#sklearn.feature_selection.SelectKBest

[4] Split arrays or matrices into random train and test subsets, bilibothèque sklearn.model_selection' scikit-learn developers, 2007 - 2019 https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html?highlight=train_test_split#sklearn.model_selection.train_test_split

[5] Randomized search on hyper parameters, bilibothèque sklearn.model_selection scikit-learn developers, 2007 - 2019, https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html?highlight=random%20search#sklearn.model_selection.RandomizedSearchCV

[6] Définition d'overfitting, WILL KENTON Updated Jul 2, 2019, https://www.investopedia.com/terms/o/overfitting.asp

[7] Ordinary least squares Linear Regression, bibliothèque sklearn.linear_model, scikit-learn developers, 2007 - 2019 https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html?highlight=linearregression#sklearn.linear_model.LinearRegression

[8] fichier README.ipynb, code de référence du projet : https://github.com/Ladiscou/AUTOBUS2

[9] K-Means clustering, bibliothèque sklearn.cluster.KMeans scikit-learn developers, 2007-2019,https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html?highlight=k%20means#sklearn.cluster.KMeans

[10] A decision tree regressor, bibliothèque sklearn.tree, scikit-learn developers, 2007-2019, https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html?highlight=decisiontreeregressor#sklearn.tree.DecisionTreeRegressor