X-PORTERS - Rapport de projet

Team name/project: X-PORTERS
Group name: AUTOCAR
Team members : MONSORO Nathan <nathan.monsoro@u-psud.fr>, EDEL Carole <carole.edel@u-psud.fr>, LEMAIRE Emilien <emilien.lemaire@u-psud.fr>, REMY Sophie <sophie.remy@u-psud.fr>, SEVE Marius <marius.seve@u-psud.fr>, VALLIMAMODE Jary <jary.vallimamode@u-psud.fr>
Challenge URL: https://codalab.lri.fr/competitions/652
Github repo of the project: https://github.com/emilienlemaire/autocar

Introduction :

Notre groupe fait parti du projet X-PORTERS. Nous avons, pour la plupart, choisi ce projet car il s'agissait de faire une régression de données contrairement à une classification dans les deux autres projets. Pour ce qui est du groupe en lui-même, nous nous connaissions à peu près déjà tous au moins de vue, donc nous avons mis très peu de temps à créer une bonne entente d'équipe. Ce projet nous a, pour le moment, apporté une connaissance plus accrue du fonctionnement et de l'utilité d'une régression, de l'importance du pré-traitement des données et bien évidemment de la nécessité de pouvoir visualiser nos résultats pour pouvoir tirer des conclusions. Les données qui sont à notre disposition pour ce projet sont en général des fichiers correspondants à des valeurs de certaines caractéristiques de nos données. Parmi elles on peut retrouver : le trafic en fonction du jour de la semaine, le trafic en fonction de l'état météorologique, etc… Pour étudier et traiter ces données nous avions un squelette de code de base que nous avons modifié et adapté pour mieux répondre aux contraintes de ce défi. Pour cela, nous nous sommes séparés en 3 sous groupes, chacun ayant une tâche bien spécifique, voici donc ce que chaque sous-groupe a réalisé.

Pre-processing (Emilien LEMAIRE et Sophie REMY):

Pour cette partie nous avons décidé de commencer par détecter les valeurs aberrantes. Pour ce faire nous avons utilisé la bibliothèque PyOD [1]. Dans un premier lieu nous avons échelonné les données pour que toutes les caractéristiques numériques (non booléennes) soient entre 0 et 1. Nous avons ensuite pris plusieurs modèles de détection de données aberrantes fournis par PyOD. Nous avons testé ces différents modèles afin de garder celui qui nous permettait d'avoir le meilleur score. Nous avons au final choisi d'utiliser **IForet**. Nous sommes passé de 38563 données à 36634.

Dans un deuxième temps, nous avons réduit le nombre de dimensions à l'aide du modèle PCA [2].

Dimension Reduction

```
In [47]: # Centering data for PCA

         from sklearn.decomposition import PCA

         # X_np[4] sont les données retournés par PyOD.IForest
         n_component = min(X_np[4].shape)

         pca = PCA(n_component)
         pca.fit(X_np[4])
         X_pca = pca.transform(X_np[4])
         Y_pca = pca.transform(Y_np[4])
```

Capture d'écran de l'utilisation du modèle PCA

Modèle (Carole EDEL et Nathan MONSORO) :

Le but de notre partie était de trouver le modèle et les méta-paramètres qui donnaient les meilleurs résultats sur les ensembles de test et de validation afin de trouver les prédictions les plus fiables. Pour cela, nous avons dû examiner plusieurs modèles différents, et passer en revue plusieurs groupes de méta-paramètres.

Avant toute chose, il nous fallait comprendre les données qui était à notre disposition. Nous avons donc d'abord inspecté les différents fichiers du projets, la construction de nos données, le nombre de features que nous avions à notre disposition etc… Bien comprendre les données du projet était primordial pour choisir un modèle adapté tant au nombre de données qu'à leur forme.

Avant de nous lancer dans les tests de modèles, nous avons regardé bon nombre de vidéo sur Youtube [a] expliquant plus en détails les modèles que nous avions pu voir sur le site de scikit-learn [b]. Après avoir mieux compris le fonctionnement des différents type de régression et comment certaines s'adaptent mieux à certains type de données, nous avons pu dresser une liste de modèles à tester.

Voici la liste des modèles que nous avons testés :

- ➔ Ridge : Le premier modèle que nous avons testé. Il avait un score si bas que nous avons d'abord pensé que nous l'avions mal utilisé. (0.1689 de score sur l'ensemble d'entraînement et 0.1696 sur celui de validation). Nous avons conclu que ce modèle n'était tout simplement pas adapté à ce que nous lui demandions.
- ➔ DecisionTreeRegressor : Le second modèle que nous avons mis à l'épreuve. Il avait un score parfait sur l'ensemble d'entraînement, bien qu'heureux d'avoir un aussi bon score après les piètres résultats du premier modèle, cela nous a mis la puce à l'oreille. Nous avons d'abord pensé que nous nous étions inquiété pour rien quand nous avons vu que le score sur l'ensemble de validation était de 0.9026, un très bon score. C'était donc le premier modèle valide que nous testions.
- ➔ KNeighborsRegressor : Le troisième modèle auquel nous avons soumis nos données. Les résultat de ce modèle était correct (0.8548 sur l'ensemble d'entraînement et 0.7477 sur l'ensemble de validation) mais il restait moins performant que le modèle des arbres de décisions.
- ➔ MLPRegressor : L'avant dernier modèle que nous avons testé. Ce modèle nous a causé beaucoup de soucis puisqu'il tournait à l'infini. Nous étions donc obligé de le stopper manuellement afin d'obtenir des résultats, mais nous ne pouvons affirmer que cela ne les a pas influencer. Les résultats que nous avions avec ce modèle étaient médiocres mais

stables (0.61557 sur l'ensemble de validation et 0.6176 sur les données de validation). Nous ne pouvions de toute évidence pas utiliser ce modèle pour des raisons purement techniques, puisqu'il nous forçait à le stopper manuellement.

➔ RandomForestRegressor: Le modèle fourni dans le starting kit, il a de bons résultats tant sur l'ensemble de test que celui de validation avec les méta-paramètres de base (0.9907 sur l'ensemble d'entraînement et 0.9455 sur celui de validation. Les résultats étant meilleurs que ceux de tous les autres modèle que nous avons testés auparavant, nous avons donc choisi ce modèle ci.

Pour tester les différents modèles, nous avons dupliqué le fichier initial et avons apporté des modifications dans ces copies. Ainsi, nous avions un fichier par modèle. Pour naviguer entre les modèles dans le README_model, nous avions ainsi juste à changer le " **from model_ import** model ". Nous avons ainsi pu collecter les scores de tous les modèles par exécution successive du fichier Jupyter.

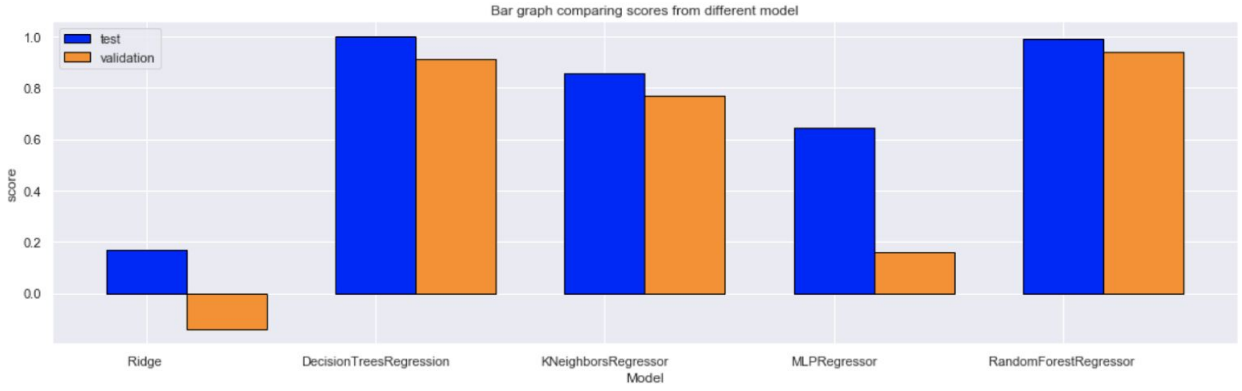
Pour le calcul de nos score sur les différents ensembles (entraînement et validation), nous avons dû séparer nos données en deux groupes pour avoir d'une part un ensemble de données pour entraîner notre modèle et un autre pour tester les performances de nos modèles sur d'autres données.

Après avoir sélectionné le modèle le plus performant (RandomForestRegressor), nous devions en déterminer les meilleurs hyper-paramètres. Pour ce fait, nous avons d'abord lu la documentation complète de cette méthode [c]. Après cela, nous avons dupliqué le fichier du modèle et testé le README_model sur les nouveaux fichier contenant le modèle sélectionné avec différents hyperparamètres. Le paramètre que nous avons fait varier est le n_estimators qui correspond aux nombres d'arbres dans la forêt. Après nos tests, nous avons déterminé qu'avec un n_estimators à 50, les scores et les temps d'exécution étaient les plus optimaux.

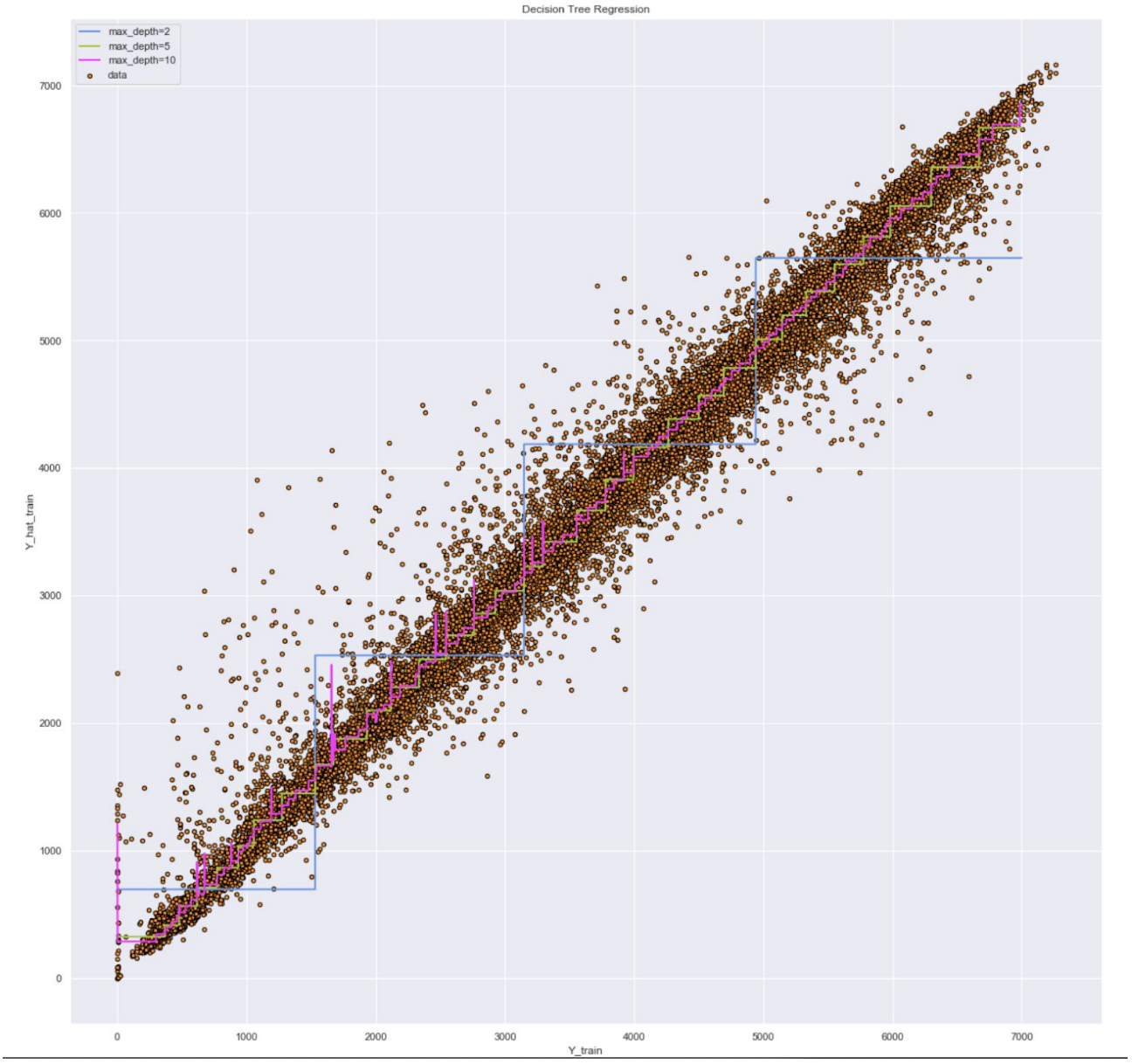
Ces tests nous ont permis de déterminer ce qui est, selon nous, le modèle le plus optimal.

Visualisation (Jary VALLIMAMODE et Marius SÈVE) :

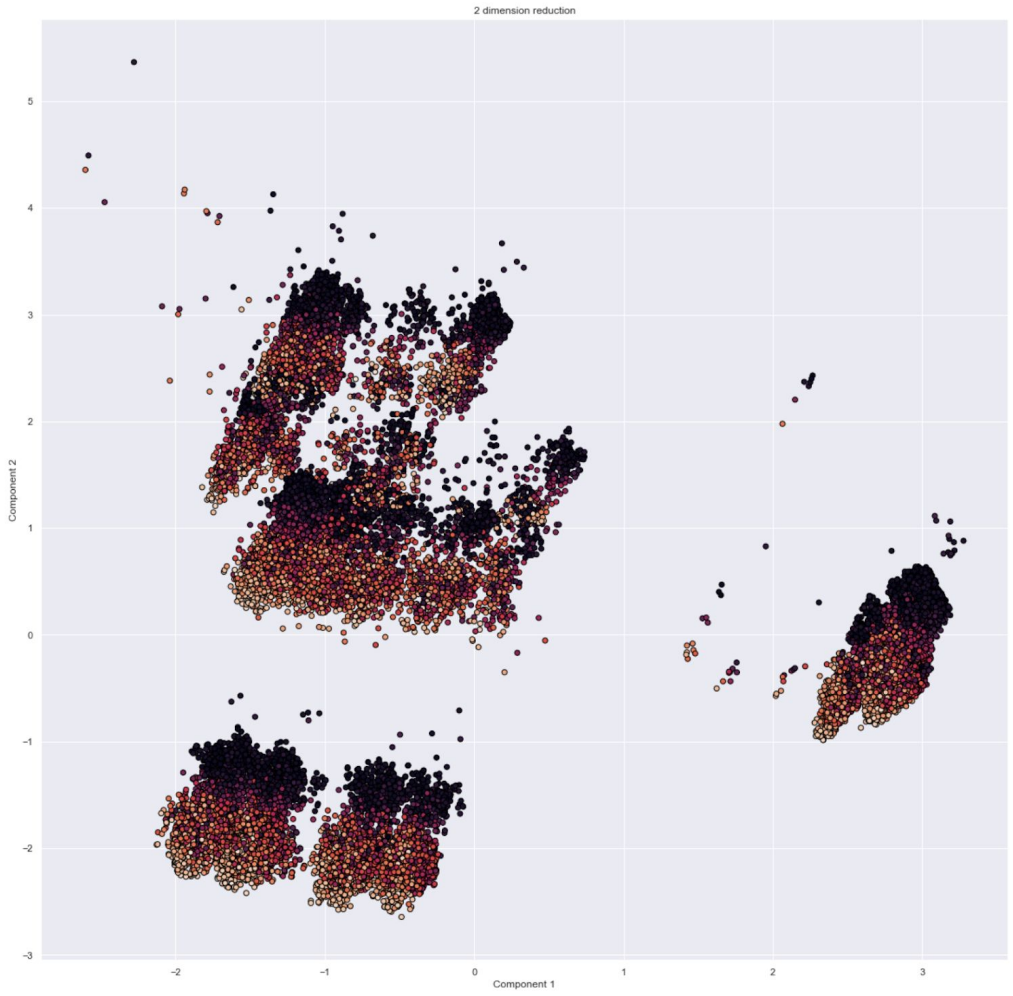
Dans un premier temps, avant de choisir le modèle optimal, le binôme **Modélisation** a du tester les performances de plusieurs méthodes. Nous avons donc voulu représenter ces performances sous forme d'un graphique à barres afin de mieux visualiser le meilleur modèle.



Dans un deuxième temps, nous avons adapté l'arbre de décision à notre data afin d'approximer le résultat attendu. On peut voir que la précision de cette approximation dépend de *max_depth*. En effet, si cette variable prend la valeur 2, la courbe obtenue est grossière. Cependant, il est nécessaire de préciser que le résultat voulu ne sera pas meilleur si *max_depth* est très grand. Cela est dû au fait qu'à partir d'un certain seuil, la courbe de précision est sensible au moindre variations (courbe rose) et présente des pics altérant sa bonne compréhension.



Dans un troisième temps, on a voulu représenter nos données sur un graphique afin de pouvoir trouver des clusters. Seul problème c'est que nous avions 58 features, nous avons donc procéder à une réduction de dimensions à l'aide de l'outil PCA. Nous avons d'abord normalise nos données afin d'éviter que des données soient ignorées au cours du processus dû à la diversité des données. On a choisi de réduire nos données à deux dimensions:



Réduire sur deux axes entraîne un regroupement assez extrême des features et donc une très large approximation. Cela nous permet certes de visualiser des clusters mais pas de trouver une corrélation entre les données. On a fait en sorte que la couleur de chaque donnée soit proportionnelle au “target”.

Conclusion :

Ce projet nous a permis d'acquérir beaucoup de compétences nécessaire à la manipulation de big data et au machine learning. Chaque groupe à bien avancé dans sa partie respective. Il nous reste tout de mêmes certains fonctionnalités à améliorer afin que notre score s’améliore encore.

Références :

Pré-Processing :

[1]
https://pyod.readthedocs.io/en/latest/

[2]
https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

Modèle :

[a]
https://www.youtube.com/watch?v=erfZsVZbGJI
https://www.youtube.com/channel/UCtYLUTtgS3k1Fg4y5tAhLbw

[b]
https://scikit-learn.org/stable/supervised_learning.html#supervised-learning

[c]
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

Visualisation :
[*]