X-PORTERS - Rapport de projet

Team name/project: X-PORTERS
Group name: AUTOCAR
Team members : *MONSORO Nathan* (2bis) <nathan.monsoro@u-psud.fr>, *EDEL Carole* (1bis) <carole.edel@u-psud.fr>, *LEMAIRE Emilien* (2bis) <emilien.lemaire@u-psud.fr>, *REMY Sophie* (2bis) <sophie.remy@u-psud.fr>, *SEVE Marius* (1bis) <marius.seve@u-psud.fr>, *VALLIMAMODE Jary* (1bis) <jary.vallimamode@u-psud.fr>
Challenge URL: https://codalab.lri.fr/competitions/652
Github repo of the project: https://github.com/emilienlemaire/autocar
Youtube video : https://youtu.be/n74BQQPQG0o
Last submission on Codalab : https://codalab.lri.fr/my/competition/submission/12937/input.zip

I/ 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. La métrique utilisée dans ce projet est r2 metrics. Cette métrique permet de déterminer l'erreur d'un modèle et de ses prédications. Le modèle détermine cette erreur en prenant le rapport entre la différence entre notre modèle et le modèle moyen et la différence entre le meilleur modèle et le modèle moyen (le meilleur modèle étant un modèle où l'on ne fait pas d'erreurs de prédictions). 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.

II/ 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. Après avoir testé plusieurs modèles nous avons choisi le modèle **IForest** qui nous donnait les meilleurs résultats. Nous avons testé ce modèle pour un ratio de données aberrantes allant de 0.005 à 0.5. Bien qu'aux alentours de 0.05 nous avions un score satisfaisant, il restait toujours en dessous de celui sans détection de données aberrantes nous avons donc choisi de ne pas procéder à ce preprocessing. Le score de base avec l'échelonnage étant de

0.993. Pour choisir le meilleur modèle nous avons pris un certain nombre de modèles les avons entraînés un par un puis avons comparé le score de chacun. Puis à partir de celui ayant eu le meilleur score nous avons modifié la proportion de données aberrantes du modèle et comparer ces scores.

Dans un deuxième temps, nous avons réduit le nombre de dimensions à l'aide du modèle PCA [2]. De la même manière que pour les données aberrantes, le PCA n'a fait que diminuer notre score. Nous avons donc choisi de ne pas utiliser cette fonctionnalité.

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 1 : Utilisation du modèle PCA

Nous avons aussi essayé de réduire le nombre de features mais cela ne faisait encore que baisser notre score. Ce choix des features a été fait en fonction de la corrélation entre les features et la cible.

URL vers le fichier test (Section Preprocessing):
https://github.com/emilienlemaire/autocar/blob/master/starting_kit/README_pre-processing.ipynb

9	0,9524
19	0,9472
29	0,9524
39	0,9557
49	0,9562
59	0,9860

Capture 2 : Score observé en fonction des nombres de features utilisé (sans autre processus de preprocessing que l'échelonnage)

Finalement d'après nos expérimentation le seul processus de preprocessing affectant positivement notre score est l'échelonnage des données entre 0 et 1.

URL vers la classe preprocessing :
https://github.com/emilienlemaire/autocar/blob/master/starting_kit/sample_code_submission/model.py

III/ 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 [3] expliquant plus en détails les modèles que nous avions pu voir sur le site de scikit-learn [4]. 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. C'est un modèle majoritairement utilisé pour les problème inverse (trouver les causes d'un phénomène à partir de ces effets). Le modèle a été développé par le mathématicien Tikhonov. Ce modèle 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, en effet notre projet n'est pas vraiment un problème inverse.
- ➔ DecisionTreeRegressor : Cette méthode consiste en la construction d'un arbre de décision afin de déterminer dans notre cas la densité du trafic en fonction des valeurs de certaines features. Cette arbre consiste, une fois construit, en un ensemble de chemins qui mène à différentes prédictions. C'est 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 : La méthode des k plus proches voisins est une méthode qui vise à associer à nos données leur résultat puis à baser nos prédictions sur les "voisins" de cette donnée. Cette méthode dépend beaucoup de la structure interne des nos données. C'est 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 : Il s'agit d'un perceptron multicouche. Celui ci fonctionne comme un réseaux de neurones organisé en plusieurs couches. C'est un système de propagation direct où la dernière couche est donc la couche de sortie. 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: Contrairement à avoir juste un seul arbre qui fournit des prédiction, les forêts d'arbre permettent de combiner les prédictions de plusieurs arbres légèrement différent pour avoir une prédiction plus précise et fiable. C'est un des modèle les plus utilisé pour traiter ce genre de problèmes. 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.

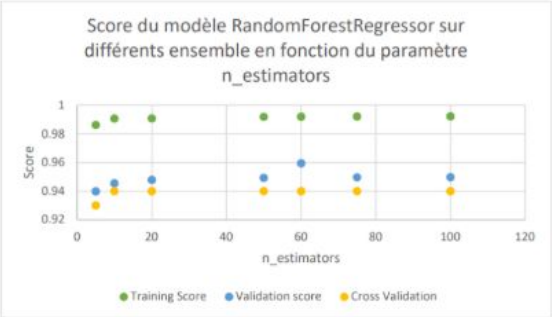

Tableau 1 : Scores des différents modèles testés

Num Modèle	1	2	3	4	5	51	52	53	54	55	56
Nom Modèle	Ridge	DecisionTreesR	KNeighborsReg	MLPRegressor	RandomForestR	RandomForestR	RandomForestR	RandomForestR	RandomForestR	RandomForestR	RandomForestR
n_estimators	_	_	_	_	10	20	5	50	100	75	60
Training score	0.1689	1	0.8548	0.6157	0.9907	0.9908	0.9863	0.9919	0.9923	0.9922	0.992
Validation score	0.1696	0.9026	0.7477	0.6176	0.9455	0.9478	0.9398	0.9493	0.9498	0.9496	0.9595
Cross Validation	-0.14	0.91	0.77	0.16	0.94	0.94	0.93	0.94	0.94	0.94	0.94
									Bons resultats mais lent		Modèle et param choisis

Figure 1: Scores du modèle RandomForestRegressor avec les différents méta-paramètres

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.

URL de nos différents fichiers modèles pour tester leurs performances :
https://github.com/emilienlemaire/autocar/tree/master/starting_kit/models/models

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 [5]. 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.

URL de notre fichier modèle :
https://github.com/emilienlemaire/autocar/blob/master/starting_kit/sample_code_submission/model.py

IV/ Visualisation (Jary VALLIMAMODE et Marius SÈVE)

URL de notre fichier visualisation (Nous avons eu un probleme d'affichage bien que chaque balise ouvrante en ferme une autre… Nous avons donc mis un deuxieme lien vers un fichier lisible téléchargeable):
https://github.com/emilienlemaire/autocar/blob/master/starting_kit/README_visualisation.ipynb
http://www.mediafire.com/file/x1beb46c5c4gusk/README_visualisation.html/file

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.

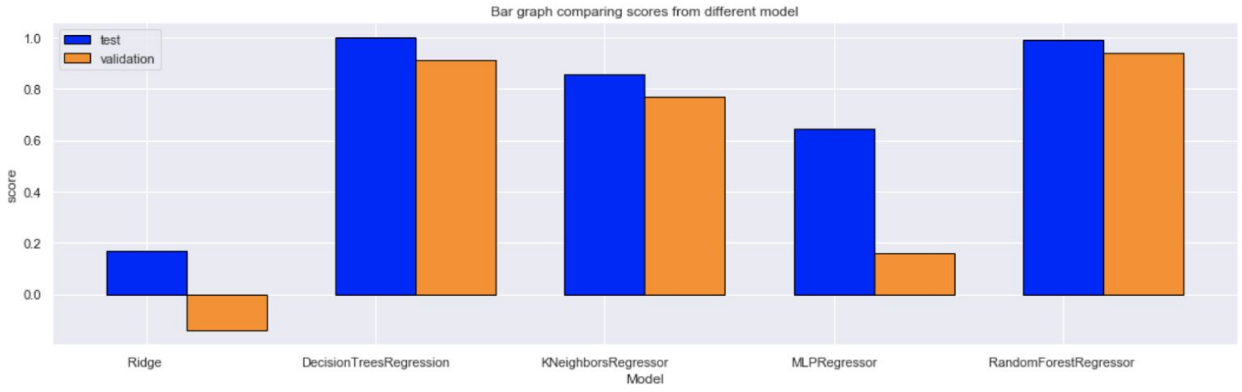

Figure 2 : Graphique à barres comparant les performances des différents modèles

Dans un deuxième temps, nous avons adapté l'**arbre de décision** [6] 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.

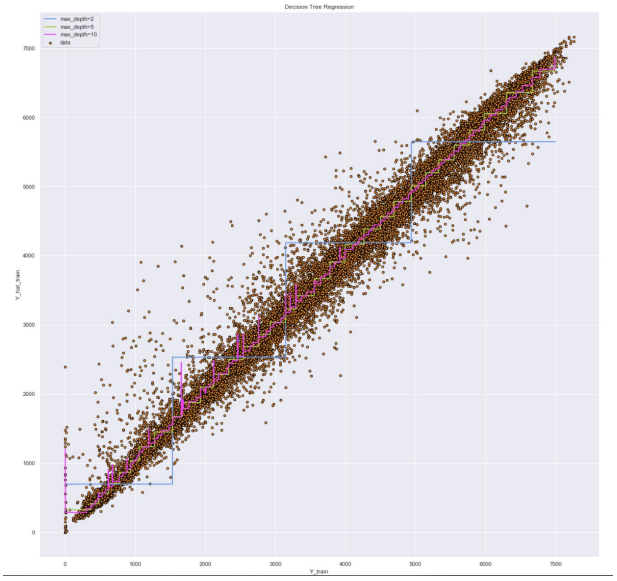

Figure 3 : Graphique de l'arbre a decision avec différentes valeurs de max_depth afin de visualiser l'erreur dans la prédiction

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 59 features, nous avons donc procéder à une **réduction de dimensions** à l'aide de l'outil PCA [7]. 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 volume de trafic avec une échelle allant de 0 à 7000 voitures.

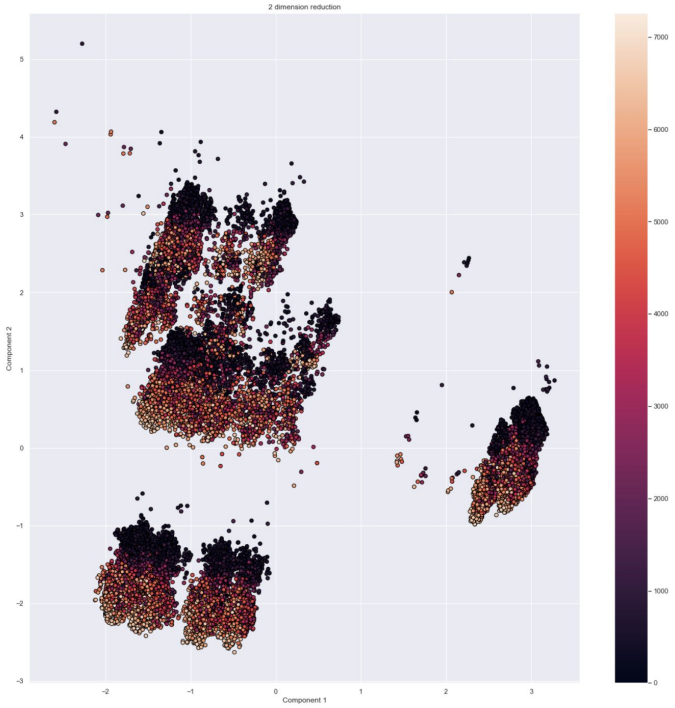

Figure 4 : Graphique des clusters formés à partir d'une réduction des donnees a deux dimensions

V/ Conclusion

Après avoir épuré nos données et choisi le modèle Random Forest qui est pour nous le plus performant, nous avons testé notre algorithme sur Codalab avec un nouveau jeu de données. Sur Codalab nous obtenons un score de 0.9476.

Ce projet nous a permis d'acquérir beaucoup de compétences nécessaires 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 certaines fonctionnalités à améliorer afin que notre score s'améliore encore.

VI/ Références

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

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

Modèle :
[3]
https://www.youtube.com/watch?v=erfZsVZbGJI
https://www.youtube.com/channel/UCtYLUTtgS3k1Fg4y5tAhLbw

[4]
https://scikit-learn.org/stable/supervised_learning.html#supervised-learning

[5]
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

Visualisation :
[6]
https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

[7]
https://scikit-learn.org/stable/auto_examples/tree/plot_tree_regression.html#sphx-glr-auto-examples-tree-plot-tree-regression-py