L2 MI - Mini Projet

Challenge **"Solve Xporters traffic volume problem"** - Groupe **Taxi**
Team composé de 6 membres :

- Fanoa RAZAFIMBELO <fanoa.razafimbelo@u-psud.fr>
- Antonin PAOLI <antonin.paoli@u-psud.fr>
- Albanio DE SOUZA <albanio.desouza@u-psud.fr>
- Taïssir MARCE <taissir.marce@u-psud.fr>
- Hilmi CELAYIR <hilmi.celayir@u-psud.fr>
- Mohammad AKHLAGHI <mohammad.akhlaghi@u-psud.fr>

URL du challenge : https://codalab.lri.fr/competitions/652
Dossier Github du projet : https://github.com/Fanoa/Taxi/tree/master/starting_kit

Contexte et description du problème :

Nous avons décidé à travers ce projet d'étudier l'ensemble de données Xporters.
L'objectif en tant que responsable d'un stand de limonade situé à côté d'une autoroute est de prédire le nombre de voitures qui passeront à une date, une heure, et des conditions météorologique données ! L'ensemble de données contient 58 caractéristiques et la solution est le nombre de voitures (de 0 à 7280) en une heure.
Pour cela, nous avons divisé le projet en 3 parties :

- **Pré-processing :** retravailler les données brutes données dans différents fichiers pour qu'ils soient exploitable par le modèle.
- **Modélisation :** trouver le meilleur modèle de régression pour l'exploitation des données ainsi que ses hyper-paramètres.
- **Visualisation :** afficher les résultats à l'aide de tableaux et graphes pour une meilleure compréhension du problème et des résultats.

Définition des différentes parties :

Partie 1 : Pre-processing (Fanoa et Taïssir)

La partie preprocessing consiste à préparer aux mieux nos données afin d'optimiser les performances de l'algorithme.[4]
Nous avons dans un premier temps essayé de détecter les anomalies dans le jeu de données. Pour cela, nous avons utilisé l'algorithme Isolation Forest proposé par scikit-learn qui va faire un partitionnement récursif des données afin d'isoler les anomalies. Puis, on va [2]réduire les dimensions du jeu de données en utilisant l'algorithme Principal component analysis(PCA)[3], cette réduction permet une amélioration dans le score final. Enfin, nous avons réalisé une feature selection en enlevant du jeu de données les données qui ont une faible variance.

Plus de détails sur le code peut être retrouvé dans le fichier README_preprocessing.ipynb dans notre Github Taxi.

Partie 2 : Modèle (Hilmi et Antonin)

Nous avons séparé les données en 2 ensembles : l'ensemble d'entraînement pour entraîner le modèle et l'ensemble d'évaluation pour tester les différents modèles et les comparer. Ensuite nous avons fait une liste de plusieurs modèles de régression. Pour chacun d'entre eux, nous l'entraînons sur le premier ensemble puis nous calculons le score sur le deuxième ensemble. Puis lorsqu'on a comparé les différents modèles et qu'on a trouvé le meilleur, nous recherchons ses hyperparamètres avec une fonction python.

Plus de détails sur le code peut être retrouvé dans le fichier README_Model.ipynb dans notre Github Taxi.

Partie 3 : Visualisation (Albanio et Mohammad)

Dans cette partie nous avons créé des figures pour la visualisation des clusters dans nos données[5]. Puis nous avons créé des graphes pour étudier l'erreur de prédiction sur le régresseur trouvé par l'équipe modèle. Et enfin nous avons tracé les performances du modèle d'apprentissage sur le jeu de données.

Plus de détails sur le code peut être retrouvé dans le fichier README_vizualisation.ipynb dans notre Github Taxi.

Résultats et Codes :

Figure 1 : Principe du feature selection sur un jeu de données

0	0	1
0	1	0
1	0	0
0	1	1

Pour effectuer une feature selection de nos données, nous calculons la variance de chaque colonne du tableau du jeu données à l'aide de la formule Var[X] = p(1 - p) avec p la probabilité de la colonne X de contenir des 0. Si la probabilité p est supérieur aux seuil défini : la colonne est supprimée. Nous avons pris arbitrairement comme seuil p = 0.7.
Dans le cas ci-dessus, par exemple, la première colonne a une probabilité ¾ de contenir des 0, ce qui est au dessus du seuil choisi. Notre algorithme va donc supprimé la colonne en question.[1]

Figure 2 : Statistiques sur les différents ensembles de données

	Nombre d'exemples	Nombre de caractéristiques	Données manquantes ?
Ensemble d'entraînement	19281	58	Non
Ensemble de validation	19282	58	Non

A l'aide de la fonction split de python, nous avons séparé les données en 2 ensembles. Cela nous aide à calculer un score sur l'ensemble de validation. Puis sur les 2 ensembles, nous gardons 58 features.

Figure 3 : Statistiques des différents modèles

	CVScore	(+/-)	ScoreTraining	ScoreValidation	Overfitted	Underfitted
decisionTree	0.895753	0.0110091	1	0.893379	False	False
KNeighbors	0.714077	0.0169543	0.824117	0.738121	False	True
Linear	0.154124	0.0397991	0.159084	-0.86189	False	True
RandomForest	0.939665	0.013485	0.989563	0.939579	False	False
ExtraTrees	0.93402	0.013007	1	0.93184	False	False
Bagging	0.940094	0.0132516	0.989845	0.940764	False	False
GradientBoosting	0.923824	0.0139217	0.921651	0.916327	False	False
AdaBoost	0.826138	0.0217314	0.820622	0.820696	False	True
GaussianProcess	-2.36806	0.121442	1	-2.31089	True	False

On peut voir que les modèle avec des arbres ont un score sur l'ensemble d'entraînement mais un score nul sur l'ensemble de validation ! De plus le modèle Bagging est celui ayant le plus haut score de validation croisée et le plus haut score sur l'ensemble de validation. Nous l'avons donc choisi !

Références :

[1]Scikit-learn 1.13. Feature selection https://scikit-learn.org/stable/modules/feature_selection.html
[2]Scikit-learn 2.7 Novelty and outlier detection
https://scikit-learn.org/stable/modules/outlier_detection.html
[3]Scikit-learn PCA https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
[4]Scikit-learn 6.3 Preprocessing Data https://scikit-learn.org/stable/modules/preprocessing.html
[5]Scikit-learn 2.3 Clustering https://scikit-learn.org/stable/modules/clustering.html