Compte rendu du projet data

Hugo GOMEZ - Vincent SCHUCK

Présentation du projet

Le projet proposé par M. Guezguez consiste à réaliser les étapes d'un projet de data science en se basant sur la prédiction du prix de revente de voiture.

L'objectif final est donc de mettre en place un modèle de Machine Learning qui va prédire le prix auquel va être revendu le véhicule que l'on souhaite le plus précis possible.

Nous ne partons pas de zéro car nous avons eu comme base de départ un dataset de véhicule d'occasion récupéré sur un site de revente de voiture.

Les principales étapes vont être de s'approprier la donnée, la nettoyer, extraire et ajouter des features importantes pour la précision du modèle et enfin mettre en place le modèle le plus précis possible.

Fonctionnement global du projet

Dans le dossier du projet, vous trouverez 4 notebook et 3 fichiers csv. En effet, pour une meilleure lisibilité nous avons séparé le notebook final en 3 fichiers. L'intégralité des notebook est aussi disponible sur https://github.com/SchuckVincent/car-project-data.

Notebooks :

- **1_explo_cleaning.ipynb** : contient la partie exploration de la donnée, nettoyage de la donnée et la partie features engineering.
- **2_scraping_prix.ipynb** : contient la partie scraping d'un site avec les données sur le prix neuf des voitures.
- **3_ml_prediction.ipynb** : contient une petite partie de features engineering, la partie sur la sélection des features, la création du modèle et son amélioration
- **4_final.ipynb** : contient les trois parties précédentes concaténées en un seul fichier

Fichiers csv :

- **Data_cars.csv** : Fichier d'origine contenant une base de donnée de voitures d'occasion.
- **Cars_price.csv** : Fichier contenant le résultat du scraping (une base de donnée des prix neufs des voitures).
- **Data_cars_final.csv** : Fichier contenant la base de donnée final avec l'intégralité des features.

Documentation de la réalisation

Partie 1 - Exploration et Features Engineering

EXPLORATION DU DATASET (ENTRÉE 2 -> 7)

L'objectif ici était de regarder un peu l'état de la donnée que l'on avait et essayer de s'approprier bien le dataset.

Pour cette partie, nous avons choisi de travailler sur un dataset réduit de 20 000 samples pour que ce soit plus simple à manipuler, mais lorsque l'intégralité des modifications ont été apportée, nous avons relancé l'intégralité du notebook avec l'ensemble du dataset.

On commence par regarder la forme du dataset, les différentes features déjà présentes et regarder le contenu des premières lignes pour voir l'état de la donnée.

La première chose qui saute aux yeux est la colonne « Description » qui contient beaucoup d'informations potentiellement intéressantes, mais qui dans l'état actuel n'est pas exploitable. On observe aussi que certaines features comme « Mileage » ou « Online » doivent être légèrement changé pour quelle soit plus facilement exploitable.

On finit cette partie en regardant s'il y a des valeurs nulles, mais dans notre cas le dataset n'en contenait aucune.

FEATURES ENGINEERING (ENTRÉE 8 -> 123)

Pour cette partie, nous avons décidé de réfléchir sur l'ensemble des features que nous pouvions ajouter avant de faire le quelconque modèle basique. C'est peut-être une erreur, mais nous avons décider de prendre le parti d'explorer à fond les possibilités que nous donnait la problématique et uniquement par la suite de travailler sur les features les plus intéressantes que nous allions conserver pour le modèle final. Ce que nous présentons dans ce notebook suit la progression réelle de l'avancé de notre projet.

- Make (Marque du véhicule) (Entrée 9 -> 18)

Pour la marque de la voiture, nous avons créé 2 colonnes différentes (Make et Make_v2). La première contient sensiblement la même chose que les données de la colonne « Make » du dataframe original à part que nous avons regroupé certaines marques qui s'avèrent, après recherche, être au final les mêmes.

La seconde colonne, elle, ressemble à la première, mais regroupe les marques de voiture les moins représentées (moins de 30 apparitions). Nous les avons regroupé, après recherche manuelle, en 4 catégories de valeurs : « high », « medium », « low » et « sans-permis ». C'est une amélioration de la première version, mais nous souhaitions conserver les deux pour pouvoir comparer leur efficacité lors de la « features selection ».

- Model (Modèle du véhicule) (Entrée 19 -> 23)

Pour le modèle, nous avons aussi créé 2 colonnes. La première contient la même chose que les données de la colonne « Model » du dataframe.

La seconde contient l'extraction de la partie « model » contenu dans la colonne « description » qui est un peu plus détaillé. Nous avons appliqué dessus un filtre qui

permet de normaliser la génération du véhicule. C'est aussi une amélioration de la première version, mais nous souhaitions aussi conserver les deux pour pouvoir comparer leur efficacité lors de la « features selection ».

- Année de construction / âge / date de l'offre (Entrée 24 -> 32)

Pour l'année de construction, nous avons choisi de regarder la répartition des véhicules en fonction de l'année de construction. Nous avons observé 2 choses. La première est qu'il y avait un véhicule avec une année très supérieur à 2020. Nous avons décidé de supprimer cet outlier pour ne pas avoir de problème. La seconde est qu'il y avait très peu de véhicules en dessous de 1990. Bien que cela puisse correspondre à des véhicules de collection ou autre, nous avons pris le parti de les retirer.

Pour la date de l'offre, nous avons décidé de ne conserver que l'année, car le reste de la date de parution ne nous à pas paru intéressant.

Enfin, nous avons décidé de calculer l'âge de la voiture, qui consiste simplement à l'année de parution de l'offre soustrait à l'année de construction du véhicule. Nous savions que cette donnée tel quel n'apporterait sûrement rien au modèle, mais nous en avions besoin pour le calcul de l'usure que nous détaillerons par la suite.

- Kilométrage et usure (Entrée 33 -> 36)

Pour le kilométrage, nous avons simplement récupéré la donnée correspondante et avons retiré la partie lettrée pour pouvoir la transformer en nombre entier et manipuler cette donnée plus facilement.

Pour l'usure, nous avons simplement calculé la moyenne de kilomètre parcouru par année en fonction des véhicules. Nous avons donc simplement divisé le kilométrage total par l'âge du véhicule.

- Fuel (Carburant) (Entrée 37 -> 40)

Après avoir récupéré la donnée correspondante et avoir regardé la répartition des carburants, nous avons choisi de les séparer en trois catégories : « diesel », « essence » et « other_fuel ». Nous avons aussi choisi, par soucis de performance, de séparer cette colonne « Fuel » en trois colonnes correspondant aux trois catégories. Dans chaque catégorie, on retrouvera 0 ou 1 en fonction du carburant utilisé. Bien qu'il existe des fonction qui font cette manipulation automatiquement (comme oneHotEncoder de sklearn par exemple), nous avons décidé de le faire manuellement.

Nous avons tout de même conservé la colonne « Fuel » originale pour pouvoir comparer par la suite leurs performances.

- Gearbox (Boite de vitesse) (Entrée 41 -> 44)

Pour la boite de vitesse, nous avons fait la même procédure que pour le carburant sauf qu'ici, il n'y a que deux choix (« manuelle » ou « automatique »)

- Description

Nous avons découpé le bloc description en plusieurs sous-catégories en faisant de multiples « split » qui vont être détaillé ci-dessous.

- Version (Entrée 46 -> 48)

La version va être elle aussi découpé en plusieurs morceaux afin d'extraire des features intéressantes. Cependant, nous avons aussi choisi de garder cette colonne complète si jamais elle peut apporter quelque chose au modèle.

- Numéro de génération (Entrée 49 -> 55)

Le premier objectif avec cette colonne « Version » a été pour nous d'extraire la génération du véhicule, car un véhicule avec une génération plus élevée coûtera la plupart du temps plus cher. Pour ce faire, nous avons créé 2 dictionnaires de correspondance des générations. Un dictionnaire dit « classique » et un dictionnaire pour les BMW. Le tableau classique associe des chaînes de caractère récurantes comme « (2) » ou « II » à la 2e génération, etc. Pour BMW, nous nous sommes basés sur un tableau qui associe un code à une génération. Par exemple « (E66) » correspond à un véhicule de 4e génération.

Une fois ces tableaux créés, nous avons appliqué un filtre sur le premier mot de la « version ». S'il apparaît dans un de nos dictionnaires, on lui donne la génération correspondante, sinon il aura par défaut la 1e génération.

À chaque fois que l'on trouve une correspondance, on supprime le premier mot de la version afin d'être sûr que cette donnée ne nous gène pas par la suite pour extraire les autres informations de la version.

- Modèle spécial (Entrée 56 -> 61)

Après avoir extrait les générations de la version, nous avons observé qu'il y avait régulièrement des adjectifs qui sortait en tant que premier mot de la version. Après renseignement, il s'agit d'un mot qui définit une spécificité du véhicule (« SW », « ESTATE », etc.). Nous avons décidé d'extraire ce mot en appliquant un filtre qui extrait le premier caractère s'il est uniquement composé de caractères alphabétiques.

Une fois fait, nous nous sommes rendu compte que seulement un faible pourcentage des voitures avait ce mot. Nous avons tout de même gardé la feature et cela nous à au moins permis de nettoyer un peu plus la version du véhicule pour pouvoir travailler plus facilement sur l'extraction des autres features intéressantes contenu dans cette colonne « version ».

- Cylindrée (Entrée 62 -> 71)

Pour l'extraction des cylindrées, qui est une feature qui nous paraissait intéressante, nous nous sommes retrouvé fasse à la difficulté que les constructeurs ne marquent pas le nombre de cylindrée du véhicule de la même manière.

Nous avons remarqué 3 manières :

- Représenté par un float à une décimale (ex : 1.6)
- Représenté par un int (ex : 63)
- Représenté par un int à l'intérieur d'une chaîne de caractère (ex : XDRIVE30D)

Pour gérer cela, nous avons créé plusieurs fonctions. 2 fonctions qui vont permettre de tester si la valeur est un nombre entier ou un nombre décimal. Une fonction qui va permettre de transformer un entier en un float « artificiel » (ex : 63 -> 6.3). Et enfin une fonction qui va permettre d'extraire les entiers présents dans une chaîne de caractère.

Nous avons ensuite procédé par étapes : on extrait d'abord les décimales, ensuite les entiers que l'on transforme et enfin les chaînes de caractère avec des nombres dedans.

Avec cette méthode, il n'y a que 3% des cylindrées qui ne sont pas trouvés.

- Chevaux (Entrée 72 -> 75)

Pour les chevaux, nous avons fait une fonction qui permet de récupérer les entiers supérieurs à 9 dans ce qui restait de la version, car c'était la dernière feature que nous souhaitions extraire de la version.

Nous avons juste appliqué cette fonction sur l'ensemble des versions et nous avons extrait environ 80 % des chevaux ce qui est correct mais pas forcément suffisant.

- Amélioration des chevaux et des cylindrés (Entrée 76 -> 83)

Pour compléter les chevaux et les cylindrés manquant, nous avons mis en place deux fonctions qui permettent de récupérer les valeurs manquantes. Pour ce faire, lorsqu'il manque par exemple les chevaux d'un véhicule, on récupère son cylindré et on regarde la médiane des cylindrées de tous les autres véhicules qui ont les mêmes chevaux. On fait de même dans l'autre sens. Cela n'est pas super précis, mais permet d'avoir une donnée cohérente et de diminuer le nombre de valeurs manquantes.

A la fin de ce processus, on obtient 99 % des chevaux et 96 % des cylindrés, ce qui est bien mieux.

- Puissance Fiscale (Entrée 84 -> 85)

Pour la puissance fiscale, on récupère simplement la donnée présente dans la description avec un « split »

- Portes (Entrée 86 -> 89)

De même pour les portes, à la différence que nous avons décidé de remplacer les valeurs nulles par la valeur 1 pour avoir une certaine cohérence.

- Couleurs (Entrée 90 -> 96)

Pour les couleurs, nous les avons regroupé en fonction de leurs couleurs. Nous aurions pu plus développer cette partie en récupérant des adjectifs qui augmentent la valeur, comme « Métallisé » ou « Chrome » mais nous avons considéré que cela allait être un gros travail pour un apport sûrement très faible.

- Options (Entrée 97 -> 110)

Pour les options, la première chose que nous avons faite a été de les transformer en tableau pour extraire le nombre d'options que le véhicule contient.

Nous avons par la suite fait un count vectorizer sur les options pour pouvoir voir quels étaient les mots les plus fréquents et par la suite extraire les options les plus fréquentes. Nous avons pris les 40 options les plus fréquentes et avons fait autant de colonnes que d'option (à la manière du Fuel). Cependant, cette méthode est clairement améliorable, car on retrouve beaucoup de mot qui ne corresponde pas vraiment à une option, comme « volant » ou « ar » par exemple.

- Prix neuf (Entrée 111 -> 120)

Après avoir scraper le site (explication plus bas), nous avons essayé d'associer le prix du modèle neuf à la voiture du dataset. Pour ce faire, nous avons créé une fonction qui permet de « match » les voitures. Tout d'abord, cette fonction récupère l'ensemble des voitures qui ont la même marque et le même modèle que la voiture en entrée. Sur l'ensemble de ces voitures, on fait un fuzzy matching sur la version pour trouver laquelle est le plus semblable. S'il le fuzzy matching trouve un modèle, on retourne son prix, sinon on retourne le prix médian des voitures avec le même modèle et la même marque. Cela fonctionne plutôt bien, mais ne retourne que 55 % de prix neuf.

Partie 2 - Scrapping

Pour le scraping du site AutoPlus, nous ne souhaitons pas nous éterniser, car nous avons suivi votre méthode. Nous avons fait le choix de scraper vraiment tout le tableau pour être sûr d'avoir le plus de modèle de voiture possible et nous avons sauvegarder le dataset dans un fichier csv pour pouvoir le manipuler plus facilement.

Partie 3 - Machine Learning et modèle final

PREPROCESS ET SPLIT (ENTRÉE 9 -> 19)

Après nous être réapproprié le dataset final avec l'ensemble des features, nous nous sommes rendu compte que certaines features n'était pas prêtes à être manipulée.

Les features textuelles comme « Make » ou « Model » ont dû être changé pour pouvoir être utilisée. Dans des cas où il y a beaucoup de valeurs différentes, nous avons préféré utiliser « LabelEncoder » de sklearn.

Nous avons ensuite séparé le label (« Price ») et les features et procédé au split du dataframe.

COMPARAISON XGBOOST / RANDOMFOREST (ENTRÉE 20 -> 30)

Pour commencer, nous avons comparé xgboost et randomforest pour savoir lequel allait nous donner la meilleure précision sur un modèle de régression linéaire très basique avec le minimum de features.

À la fin du test, xgboost était légèrement plus précis que randomforest, nous avons donc préféré xgboost pour la suite de nos tests.

AMELIORATION DU MODEL - METHODE 1 (ENTRÉE 31 -> 98)

Pour faire le feature engineering, nous avons utilisé 2 méthodes. La première a consisté à ajouter une feature et de regarder son impact sur la précision. Si cela augmente, on garde la feature, si cela baisse ou si c'est insignifiant, on ne la garde pas. Nous n'allons pas détailler feature par feature, car cela serait redondant.

À la fin de ce travail, nous avons gardé 13 features : Make_v2, Model_Det, Model_year, Mileage, Diesel, Essence, Other_fuel, Mecanique, Automatique, Cylindre, Chevaux, Puissance_fisc, Prix_neuf et la précision de notre modèle était de 91.6 %.

AMELIORATION DU MODEL - METHODE 2 (ENTRÉE 99 -> 111)

Étant donné que cette méthode était longue et difficile à mettre en place lorsqu'on a beaucoup de features, nous avons chercher à industrialiser le procédé.

Nous avons donc utilisé la fonction « get_booster » qui permet de trier les features en fonction de leur importance. Nous avons ensuite gardé uniquement les features avec un score supérieur à 1000.

Le modèle final avec cette méthode à eu un score quasiment identique à la précédente méthode.

MODÈLE FINAL (ENTRÉE 112 -> 125)

Pour finir, nous avons modifié les paramètres du modèle afin d'optimiser au maximum le score de performance de notre modèle. Ces modifications ont étaient faite en tâtonnant, car il était difficile pour nous de vraiment trouver les meilleurs paramètres directement.

Notre modèle final à une précision de 91.77 %.

Conclusion

Ce projet était très intéressant et nous à permis de mieux comprendre comment bien appréhender ce genre de problématique. Bien que la plupart de notre code est loin d'être très propre et professionnel, nous espérons qu'il soit tout de même compréhensible. Nous en tirons bien évidemment beaucoup de positif.