Rapport de projet

Prédiction du prix de vente d'une voiture d'occasion selon les caractéristiques qui la décrivent.

Par Taïssir MARCÉ, Alexandre COMBEAU, Céline YAN, Hugo MOUDEN, Mathilde LASSEIGNE et Simon BIBITCHKOV

Préambule :

Le prix des voitures populaires et son évolution dans le temps sont des sujets sur lesquels une immense majorité de la population est amenée à se questionner. Il est important de connaître la tendance de prix d'un modèle de voiture lorsque l'on souhaite vendre une voiture.

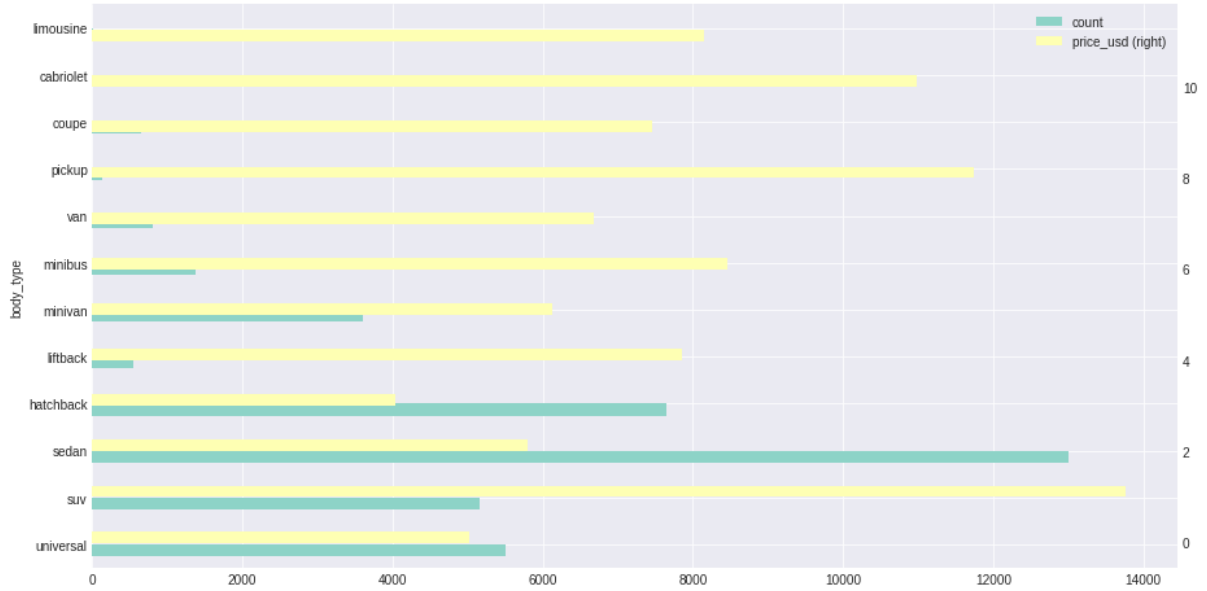
En fonction des caractéristiques fonctionnelles mais également de la marque, l'année de production et l'apparence de la voiture par exemple, le prix peut varier énormément. A partir d'une base de données contenant de telles caractéristiques et le prix d'une voiture, notre tâche consistera donc à prédire le prix des voitures en fonction des caractéristiques.

Dataset utilisé : https://www.kaggle.com/lepchenkov/usedcarscatalog
Projet github : https://github.com/dabeoh/ias_cars

Sommaire :

Visualisation

La visualisation permet d'avoir un aperçu des données sous forme de tableaux ou de graphes mais également de mettre en valeur et filtrer les données importantes.



Afin de visualiser globalement et efficacement nos données, nous avons commencé par les afficher sous forme de tableaux, chaque colonne représentant une caractéristique. Nous pouvons afficher les premières lignes du dataset grâce à la méthode `head()` afin de mieux comprendre le contenu et l'organisation du jeu de données. Comme précisé dans notre code, la méthode `describe()` nous permet d'obtenir, pour chaque colonne certaines données statistiques (moyennes, médiane, écart-type…) qui nous aident à mieux appréhender nos données. Puis nous affichons toutes les colonnes sous forme de liste afin de pouvoir visualiser immédiatement l'ensemble des caractéristiques.

Une feature intéressante à examiner est le type de carrosseries des véhicules ("`body_type`"). Nous en avons visualisé la répartition à l'aide d'un histogramme. Ainsi par exemple, dans le dataset donné, la majorité des véhicules sont des "sedan" (berlines), et le nombre de limousines est infime.
Ces observations nous permettent ensuite d'apporter un regard critique sur le prix moyen des véhicules selon le type de carrosserie (plus l'échantillon de véhicules est petit pour un type donné, plus la moyenne doit être relativisée).

Ainsi, pour chaque feature du dataset nous avons créé un histogramme nous permettant de comparer la fréquence d'une valeur de la colonne, avec le prix moyen des véhicules possédant ces features. L'ensemble des informations que nous tirerons de ces différentes représentations des données nous permettront de comprendre avec plus de subtilité l'effet des différentes méthodes de traitement, et de repérer plus facilement les limites de notre code.

Prétraitement

Notre travail de prétraitement sur le jeu de données se décompose en trois tâches principales

- *Nettoyage du jeu de données*
- *Transformation des données qualitatives en données quantitatives*
- *Standardisation des données*

Nettoyage

Après avoir observé notre jeu de données, nous avons remarqué que plusieurs colonnes n'étaient pas utilisables pour notre prédiction. Il est donc nécessaire de commencer par ***nettoyer le jeu de données*** *de ce qui n'apporte pas d'informations, pour ne pas surcharger inutilement les algorithmes que nous utiliserons.*

Pour réaliser la première étape, il nous a fallu identifier les colonnes qu'il serait judicieux de ne pas garder.

Dans notre jeu de données, les colonnes `feature_0` à `feature_9` sont des booléens qui décrivent si oui ou non une voiture possèdes certaines options d'achats (telles que les jantes en aluminium ou la climatisation). **La description fournie pour accompagner ces colonnes est imprécise**, et ne nous permet pas de savoir à quoi correspond chaque colonne. Nous ne pourrons donc pas les garder pour la suite.

De même, certaines lignes **contiennent des valeurs nulles**. D'après l'en-tête des colonnes, il n'est pas réaliste qu'une voiture soit caractérisée par une valeur nulle. Nous considérerons donc que les lignes concernées contiennent une erreur, et pour ne pas fausser l'apprentissage, elles seront également exclues.

Enfin, dans la colonne "`engine_fuel`", **les données ne sont pas correctement formatées** : il apparaît tantôt "`gas`", tantôt "`gasoline`". Comme cela désigne la même chose, on ne garde qu'un seul des deux termes, et on renomme toutes les occurrences de "`gas`" en "`gasoline`".

Une fois ce nettoyage accompli, nous pouvons nous concentrer sur les données que nous allons effectivement utiliser.

Valeurs Qualitatives

La deuxième étape consiste à ***adapter les colonnes dont les données sont sous forme de texte****. En effet, ce format de données n'est pas utilisable tel quel car les algorithmes de prédiction ne travaillent que sur des formats de données numériques.*

Ici, les colonnes `['manufacturer_name', 'model_name', 'color', 'engine_fuel', 'engine_type', 'transmission', 'body_type', 'state', 'drivetrain', 'location_region']` sont des chaînes de caractères. On doit donc convertir le contenu de ces colonnes de données avant de pouvoir les utiliser.

Toutes ces colonnes admettent un nombre fini de valeurs différentes. La plupart d'entre elles, étant des caractéristiques qualitatives, ne peuvent pas être hiérarchisées. De fait, une simple conversion en dictionnaire tel que : `["a" : 1; "b" : 2; "c" : 3]` n'est pas viable, car elle hiérarchise les données en donnant à "`c`" une valeur plus importante que "`a`", alors que ça n'est pas nécessairement justifié.

Nous allons alors encoder nos données en utilisant une méthode de "One Hot Encoding". Pour réaliser un tel encodage, on convertit une colonne en vecteur auquel on ajoute, pour chacune des valeurs différentes, une nouvelle colonne. Pour représenter un point de donnée, on utilise donc un langage booléen (0 ou 1). Ligne par ligne, on indique un '1' dans la colonne correspondant à la caractéristique présente. (il y aura donc 0 pour toutes les autres colonnes). Pour réaliser cet encodage sur toutes les caractéristiques qui s'y prêtent, on utilise la méthode de prétraitement fournie par la bibliothèque scikit learn "`sklearn.preprocessing.OneHotEncoder()`", pour encoder une colonne comme il convient. Ensuite, on ajoute chacune des colonnes ainsi converties au tableau qui représente le jeu de données.

Néanmoins, il ne faut pas négliger que pour chaque point de donnée, pour représenter une caractéristique, on remplace une unique valeur qualitative en un vecteur de K valeurs (valant 0 ou 1,

et K étant le nombre de valeurs différentes dans la colonne). La taille du jeu de données est donc logiquement augmentée. Si la valeur de K est trop grande, cela peut affecter l'efficacité du modèle. Ici, c'est le cas pour la colonne qui définit le modèle d'une voiture. Il y a dans cette colonne 1118 modèles différents. Pour remédier à cela, on a choisi de regrouper en une catégorie "`other_model`" les modèles dont le nombre d'occurrences est inférieur à une valeur k fixée. Cette valeur est un hyper-paramètre de la prédiction. Par exemple, en fixant k = 10, on réduit le nombre de modèles différents à 522. C'est à dire qu'en ne gardant la précision du modèle uniquement pour ceux qui apparaissent au moins 10 fois, on ignore les 594 autres modèles qui sont moins récurrents.

Pour en finir avec le prétraitement des données, on standardise les données. En faisant une première régression linéaire sur le jeu de données avant de les standardiser (*Dans le code : "1.4 Annexe - Régression linéaire sans standardisation"*), on constate qu'un grand nombre de valeurs qui sont censées représenter le prix de la voiture sont négatives, ce qui n'a pas de sens compte tenu du contexte. De plus, si on observe avec "`cars['price_usd'].describe()`" notre colonne de label, on remarque bien que les résultats devraient être compris entre 1 et 50000 (min. et max).

Standardisation

*La troisième étape du prétraitement est la **standardisation** des données. La standardisation est une étape indispensable au traitement de l'information recueillie. Si d'une colonne à une autre, les échelles sont trop disparates, il est possible que les résultats obtenus par les algorithmes de prédictions ne soient pas représentatifs de l'ensemble des colonnes.*

Pour standardiser les données, nous commençons par transformer toutes les valeurs obtenues à la suite du pré-processing afin qu'elles soient comprises entre 0 et 1. Cette mise à l'échelle est nécessaire car comme vu précédemment, nous avons utilisé un encodage OneHot qui impose un format de données entre 0 et 1. Pour réaliser cette étape, on utilise la méthode fit_transform. On note que pour la caractéristique de l'année de construction, on a d'abord effectué une petite opération permettant de travailler plutôt avec l'âge de la voiture (`age = 2020 - année de construction`).

Modèle

Nous avons testé deux modèles différents pour produire des prédictions concernant le prix des voitures. D'abord une **régression linéaire**, puis une **forêt aléatoire d'arbres de décisions**. Pour utiliser chacun des deux modèles, on utilise la méthode `gridSearchCV` de la bibliothèque `ScikitLearn`. Cette méthode permet d'optimiser les hyper-paramètres d'un modèle donné. Elle prend en paramètre une liste exhaustive des valeurs d'hyper-paramètres à tester. Puis, à partir de cette liste, chaque combinaison de valeurs d'hyper-paramètres va être testée, et sa fiabilité va être déterminée par une cross-validation. La méthode va nous permettre d'obtenir, de manière efficace et automatisée, la combinaison de valeurs des paramètres les plus optimisées pour la prédiction.

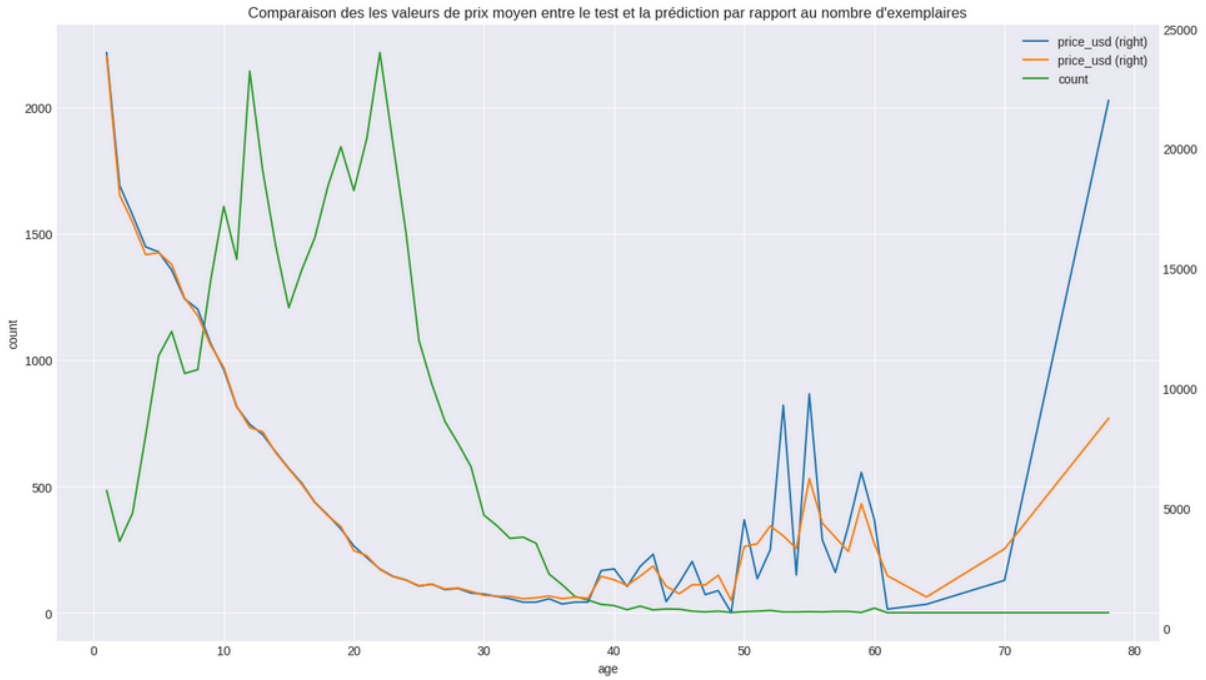
La première prédiction, réalisée à l'aide d'une régression linéaire, nous permet premièrement de constater que notre travail de standardisation a été fructueux, puisque les valeurs obtenues sont de moins en moins aberrantes. On retrouve tout de même des incohérences mineures, et on constate surtout que la courbe obtenue n'est pas strictement une droite. Cela s'explique par le fait que la répartition des prix n'est pas uniforme. Il aurait été nécessaire d'ajouter une étape (non négligeable) au prétraitement, qui serait un travail d'uniformisation de la répartition.
Ce faisant, le score que nous obtenons en utilisant une simple régression linéaire est très loin d'être satisfaisant.
Pour remédier à cela, on utilise l'algorithme `RandomForest`, qui nous permet d'obtenir un bien meilleur score.

L'algorithme de `RandomForest` se base sur l'utilisation d'arbres de décision. Un ensemble de points de données tirées aléatoirement sont analysées pour construire un ensemble d'arbres de décision le représentant. Après avoir éventuellement réitérer ces étapes, les différents arbres ainsi créés sont alors assemblés, et le résultat de la prédiction correspond à la moyenne des résultats de chacun des arbres de l'ensemble. C'est en définitive un algorithme qui effectue un grand nombre d'opérations lors de l'apprentissage**,** et permet donc en théorie de réduire les risques de biais : on combine un ensemble de prédictions moyennement efficaces pour obtenir en somme, une prédiction très performante. On a en effet obtenu un score très satisfaisant avec cet algorithme.
Ce gain de performance s'accompagne cependant d'une différence de complexité significative. En effet, l'algorithme qui utilise des forêts aléatoires d'arbres de décisions nécessite énormément de ressources et peut très facilement devenir très chronophage lors de son exécution. De plus, de nombreux regards critiques sur la méthode `RandomForest` en général lui attribuent un risque de sur-apprentissage.

Tests du modèle et analyses des résultats

Après avoir conçu des modèles de traitement de notre jeu de données, il est indispensable d'effectuer divers tests, comme nous l'avons fait déterminer lequel des deux modèles abordés sera conservé. On peut aussi choisir de tester les modèles en y soustrayant des caractéristiques qui semblent être les plus fortes, ou les plus représentées par exemple. Ici, observer les scores des modèles aura été suffisant pour choisir d'abandonner la régression linéaire.

Ce faisant, nous nous sommes par la suite principalement consacrés à l'analyse des prédictions proposées par le `RandomForest`. Au premier abord, la prédiction nous semble pertinente, et l'algorithme, qui obtient un score de 0.89 indique une prédiction cohérente. Cependant, ce score représentant une moyenne, nous avons également réalisé, à l'aide de différents masques filtrant le jeu de données, d'autres prédictions, afin de concentrer notre attention sur certaines features. Par exemple, lorsque l'on exclut les voitures les plus jeunes (toutes celles dont l'âge est inférieur à l'âge médian de l'ensemble des voitures). Appliquer un tel filtre permet d'observer d'intéressants phénomènes, notamment le fait que le score de la prédiction chute significativement lorsque les voitures sont trop anciennes. D'après les histogrammes de visualisation, on observe en effet que les voitures anciennes sont très nombreuses par rapport aux autres. On doit donc prendre un certain recul vis-à-vis de l'affirmation *"[L'utilisation d'une forêt d'arbres de décision aléatoire] permet donc en théorie de réduire les risques de biais"*. Même si cet algorithme a l'avantage de produire des résultats peu sensibles biais qui peuvent être causés par les valeurs aberrantes, il est très sensible à la répartition des données dans le jeu de données. Ici, on a beaucoup plus de données pour les voitures les plus vieilles (la moyenne d'âge des voitures étant à 17 ans), et il y a relativement peu de points de données pour les voitures de moins de 5 ans par exemple. Pourtant, en observant les données sur un histogramme, on voit que le prix de ces dernières est remarquablement plus élevé que les voitures plus anciennes. Comme l'algorithme est très peu sensibles aux valeurs drastiquement élevées (ici, le prix des voitures récentes), mais très sensible au valeurs sur-représentées (ici, les voitures âgées), l'algorithme réalise un sur-apprentissage sur les points de données sur-représentés. En conclusion, il est bien moins performant lorsqu'il s'agit de prédire le prix d'une voiture récente, car son apprentissage est basé sur les voitures âgées.



Price_usd (right) correspond à la prédiction, et price_usd (left) aux valeurs de test

Bien que nous n'ayons exploré que peu de modèles, l'un d'entre eux nous a permis d'obtenir un résultat que nous considérons acceptable.

Cependant, et pour conclure sur l'analyse de nos résultats, nous avons réfléchi à des pistes à explorer pour améliorer ces derniers. Tout d'abord, nous avons vu que la répartition du prix des voitures récentes est très différente des autres. Il pourrait être pertinent de séparer le jeu de données en deux catégories distinctes afin de pouvoir travailler sur des modèles plus simples et plus adaptés aux données utilisées. Ensuite, nous avons vu que les points de données ne sont pas uniformément répartis. Dans ces cas-là, le choix d'un algorithme basé sur les arbres de décision est très questionnable, car ces derniers sont très sensibles à la surreprésentation dans le jeu de données. L'utilisation d'un modèle plus adapté à une répartition inégale serait pertinente. Enfin, notre prédiction pourrait également être améliorée en développant le travail de prétraitement des données, qui manque actuellement d'une méthode de suppression des valeurs aberrantes.

Sources

Pour débuter notre travail, nous avons tiré du code suivant les lignes directrices de notre analyse : https://www.kaggle.com/abhimanyudasarwar/used-cars-price-prediction