GLO-4027 - Bike Sharing Demand

Premier traitement des données

20 mars 2019

Étudiants en programme d'échange international:
*Bastien **CHABAL***
*Corentin **GIRAUD***

Projet Kaggle – Repository GitHub

Sommaire

Introduction

Le projet que nous avons choisi est intitulé Bike Sharing Demand (plateforme Kaggle). On se propose d'analyser les données du système de partage de vélos de la ville de Washington, D.C.

Le système de partage de vélos au sein d'une ville est très simple : plusieurs kiosques sont répartis dans la ville et des personnes (abonnées ou non) peuvent louer un vélo et faire le trajet qu'elles souhaitent jusqu'à un autre kiosque.

Rappel général sur les données

Les données sont réparties en deux fichiers *.csv* distincts:

- Le **set d'entraînement** contient deux ans de données (année 2011 et 2012), où sont relevées toutes les heures différentes informations, dont le nombre total de locations. Ce set ne contient que les données des **19 premiers jours de chaque mois**.
- Le **set de test** contient des données exactement similaires au set d'entraînement, mais pour **tous les jours après le 20 du mois inclus**.

Après une première visualisation des données proposées par le set d'entraînement, on constate qu'il contient 10 886 entrées, réparties sur **12 attributs**. Chaque tuple représente les données pour 1 heure. Tous les attributs ont des valeurs, ce qui signifie qu'il n'y aura pas de prétraitement des données afin de corriger **l'intégralité**.

Transformation des données

Attribut date

Nous avons découpées l'attribut en plusieurs attributs correspondants chacuns à une unité temporelle.
Nos données sont donc maintenant munies des attributs *date*, *hour*, *month*, *year* et *weekday*. Classification des attributs

Nous avons donc trois types de valeurs différentes :

- Les attributs que l'on nomme *categoricalAttributeNames*: ce sont des entiers qui représentent des catégories (par exemple, season=1 représente le premier quart de l'année).
- les attributs numériques que l'on nomme *numericalAttributeNames*: ce sont des float qui représente une mesure numérique.
- les attributs que l'on nomme *dropAttributes*. On les supprime, on ne souhaite pas conserver dans l'entraînement du modèle car :
    - ce sont les attributs que l'on souhaite prédire (*count*)
    - ce sont des attributs redondants (*casual*, *registered*, *datetime*)

Division des données d'entraînement

On souhaite mettre en place un apprentissage supervisé. Ainsi, on divise *dataTrain* de cette façon :

- 70% pour l'entraînement
- 30% pour la validation.

Prédictions

Score RMSLE

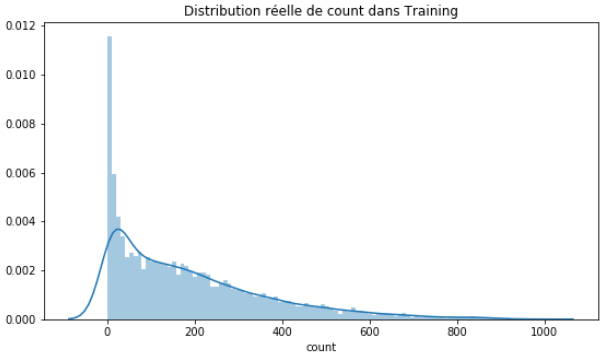
Pour tester nos résultats nous avons défini un *RMSLE scorer* au sein du code qui mesure le ratio entre la valeur prédite est la valeur réelle. D'après nos recherches sur cet indicateur, il est utilisé quand on ne veut pas pénaliser les différences entre deux valeurs (réelle et prédite) élevées.

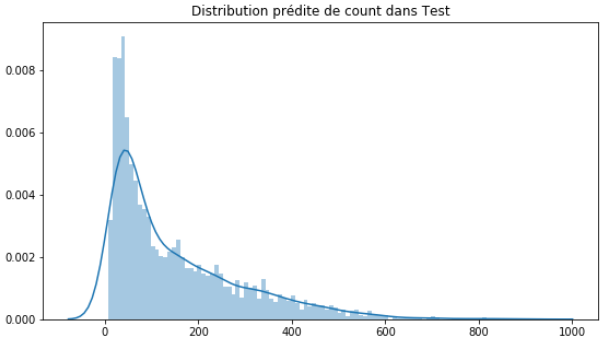
On constate également que le leader du projet Kaggle obtient un RMSLE de **0.3375**, ce qui peut être un bon référentiel.

Premier essai : prédiction par Régression Linéaire

Grâce à scikit-learn, on met ensuite en place un premier modèle de régression linéaire.

On obtient un RMSLE de **1.023**. Ce résultat est déjà très loin dans le leaderboard et nous indique que le modèle fait des prédictions de pauvre qualité. On peut vérifier cela avec les graphiques de distributions.





On constate des variations considérables entre la distribution réelle du *train set* et la distribution prédite du *test set*.

Il paraît donc évident qu'un modèle de régression linéaire ne convient pas à ce projet.

Prédiction par Random Forest

Pendant l'étape de recherche et d'implémentation d'algorithme, la librairie scikit-learn nous a vraiment facilité la tâche car elle permet la mise en place d'un modèle de machine learning assez rapidement.
Après avoir explorer plusieurs algorithme, nous nous sommes attarder sur Random Forest (ou prédiction par forêt aléatoire) et les résultats sont très satisfaisants. Avant de les présenter dans ce rapport, nous allons expliquer pourquoi ce modèle est adapté à nos données et à ce que l'on veut prédire.

Random Forest est basé sur les arbres de décision. C'est pourquoi il est intéressant de diviser la date en plusieurs attributs : les requêtes possibles sont plus nombreuses, on passe d'une requête unique sur la date à plusieurs requêtes sur l'année, puis la saison, puis le mois, puis le type de jour, etc.

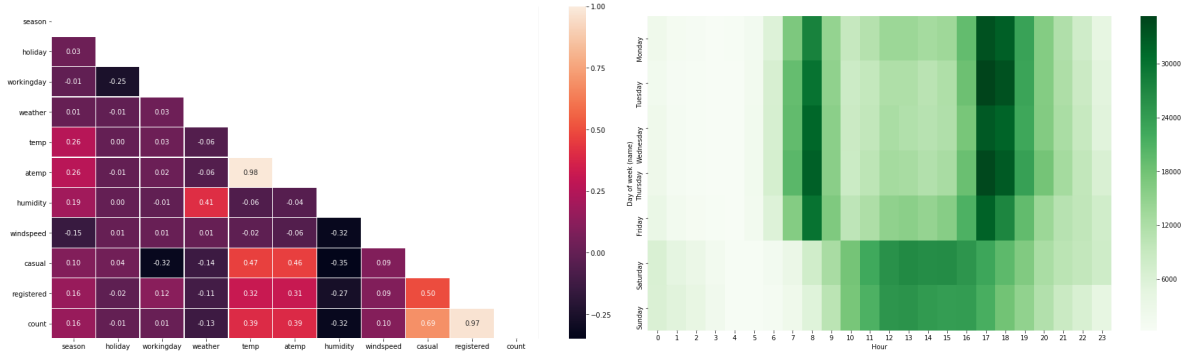
Pendant l'entraînement, on donne donc au modèle tous les éléments utiles à sa création (cf. Classification des attributs) et la vraie valeur de ce qu'il doit prédire. Le modèle apprend les relations qui existent entre les attributs et la cible (ici *count*), calcule les bonnes requêtes a posé pour faire la meilleure estimation possible.

Cependant, une prédiction a toujours un écart, avec la valeur réelle. Minimiser cet écart, c'est l'objectif de Random Forest. Le processus est simple : combiner plusieurs arbres de décision en un seul modèle, et prendre la moyenne de leurs prédictions.
Pour créer de la diversité dans cette forêt, chaque arbre n'a accès qu'à :

- un subset du set de données
- un subset du set d'attributs

Cela rend les prédictions plus robustes car chaque arbre part avec des connaissances différentes.

Pourquoi Random Forest est adapté au projet ?



Les attributs sont nombreux et variés. Après les avoir analysés dans les précédents rapports, voici les informations que nous en avons tirés :

- La matrice de corrélation (à gauche) ne donne pas de corrélation "flagrante" entre deux attributs. Ainsi *count* est influencé par l'ensemble des attributs, ce qui indique qu'il faut tous les considérer.
- La heatmap (à droite) représentant la somme des locations agrégées par heure et par jour de la semaine nous a indiqué que l'on pouvait faire un estimation plus précise en se posant les questions dans l'ordre suivant : de quel jour s'agit ? Est-ce un jour de travail ? Quelle heure est-il ?
  C'est d'ici que vient l'intuition de diviser l'attribut : plus de questions amène à une estimation plus précise.

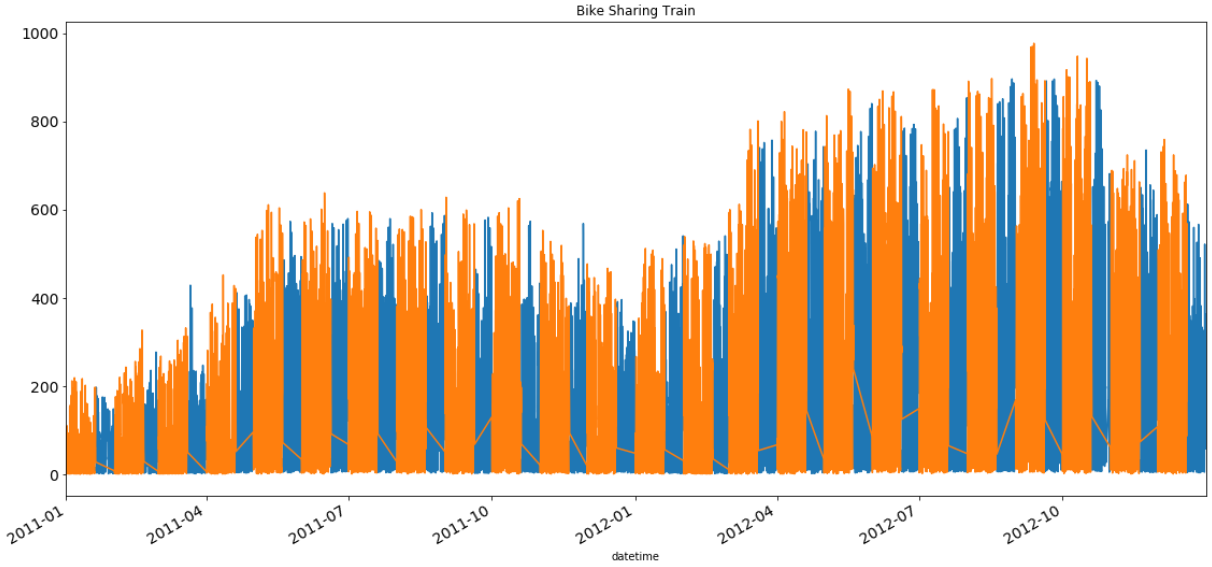
Voilà pourquoi l'algorithme Random Forest est adapté : nous disposons d'un large éventail d'attributs qui semblent chacun avoir une importance, et nous pouvons dégrossir le problème en une série de questions comme dans un arbre de décision.

Résultat

Nous avons donc implémenter un modèle utilisant Random Forest, entraîné sur 70% des données et testés sur les 30% restants.

On obtient un RMSLE de **0.3487.**

On peut alors visualiser le graphe complet : en orange les données connues (du 1 au 19 du mois), en bleu les données prédites (du 20 à la fin du mois).



Pour rappel, le code implémentant ce modèle est disponible ici.

Conclusion

Nous sommes satisfaits de ce résultats car l'algorithme Random Forest semble particulièrement adapté à la structure du problème et à ce que nous voulons prédire.

Cependant des améliorations sont possibles. Comme évoqué dans les rapports précédents, nous pourrions faire une prédiction sur l'attribut *casual* et une l'attribut *registered* car ils ont des comportements différents.

Références

https://medium.com/@viveksrinivasan/how-to-finish-top-10-percentile-in-bike-sharing-demand-competition-in-kaggle-part-1-c816ea9c51e1

https://www.kaggle.com/general/21582

https://medium.com/@williamkoehrsen/random-forest-simple-explanation-377895a60d2d

https://en.wikipedia.org/wiki/Random_forest