Mémoire de certification

Conduire un projet de sciences de données

CNCP 1527

Recruit Restaurant Visitor Forecasting

Predict how many future visitors a restaurant will receive

VIDAL Jordan

Data Science Starter Program

DSSP 7 – 2017

Sommaire

1. Abstract

Suite à la formation Data Science Starter Program proposée par l'Ecole Polytechnique Executive Education, un projet personnel doit être mené suite à 20 jours de formation présentielle. Ce papier est le résumé du projet que j'ai mené afin d'obtenir la certification.

Il est à noter que tous les output des commandes effectuées se trouve dans un fichier « nohup.out »

2. Introduction

Le projet est un sujet proposé par Kaggle, plate-forme en ligne de compétitions Data Science.

L'objectif est de prédire le nombre de visiteurs dans certains restaurants japonais à des futures dates précises. Les données fournies sont issues d'un système de réservation en ligne ainsi que les données de visites dans de nombreux restaurants. Ces informations permettent d'aider les restaurants à être beaucoup plus efficaces et de se concentrer sur l'experience client au niveau culinaire.

3. Présentation des données

Les données proposées proviennent principalement de deux sources différentes:

1. Le système Hot Pepper Gourmet (**hpg**): similaire a Yelp où les utilisateurs peuvent chercher des restaurants et effecteur des réservations en ligne.

2. Le système AirREGI / Restaurant Board (**air**): similar à Square, un système de contrôle de réservation et de caisse.

Ci-dessous la liste des 7 dataset fournis:

air_reserve.csv : réservations effectuées via le système AIR

hpg_reserve.csv : réservations effectuées via le système HPG

air_store_info.csv : informations sur les restaurants du système AIR

hpg_store_info.csv : informations sur les restaurants du système HPG

store_id_relation.csv : fichier permettant de joindre les restaurants à la fois des systèmes AIR et HPG

sample_submission.csv : équivalent au fichier de “test”, permettant de prédire, au bon format, le nombre de visiteurs

data_info.csv : fichier recensant des informations basiques sur les dates présentes dans les fichiers de données évoqués ci-dessus.

On y distingue les dataset préfixés par “air” et “hpg” pour les deux sources de données mentionnées ci-dessus. Ils contiennent les réservations, visites et autres informations nécessaires à la prédiction de futurs visiteurs à des dates spécifiques. Enfin, le fichier “store_id_relation.csv” permet de faire la jointure entre les restaurants communs aux deux systèmes de réservation.

*Dans la suite de ce memoire, par soucis de clarté, les dataset seront appelés par leur diminutifs: “tra” pour le **tra**in, tes pour “**tes**t”, “as” pour “**A**ir_**S**tore_info”, hs pour “**H**pg_**S**tore_info”, ar pour “**A**ir_**R**eserve”, “hr” pour “**H**pg_**R**eserve”, id pour “store_**ID**_relation”, hol pour “date_info”.*

4. Data Exploration

Afin de mieux se familiariser avec les données et suite à l’importation de celles-ci, une phase d’exploration est nécessaire. Cette phase exploratoire est primordiale pour mieux comprendre les données et la structure des dataset.

```
data = {
    'tra': pd.read_csv('data/air_visit_data.csv.zip', compression="zip"),
    'as': pd.read_csv('data/air_store_info.csv.zip', compression="zip"),
    'hs': pd.read_csv('data/hpg_store_info.csv.zip', compression="zip"),
    'ar': pd.read_csv('data/air_reserve.csv.zip', compression="zip"),
    'hr': pd.read_csv('data/hpg_reserve.csv.zip', compression="zip"),
    'id': pd.read_csv('data/store_id_relation.csv.zip', compression="zip"),
    'tes': pd.read_csv('data/sample_submission.csv.zip', compression="zip"),
    'hol': pd.read_csv('data/date_info.csv.zip', compression="zip").rename(columns={'calendar_date':'visit_date'})
    }
```

Image 1 - Dataset sources

Nous pouvons séparer ces dataset en 3 catégories: les réservations, les restaurants, le reste. Ces dataset sont désormais des dataframe pandas.

4.1 Réservations

Dans un premier temps, il m’a semblé pertinent de commencer mon exploration sur l’axe des réservations (analyse des dataset “ar” et “hr”). Mes premiers constats étaient les suivants:

1. Il y a vingt fois plus de réservations effectuées via le système HPG que via le système AIR (annex 1)
2. Les deux fichiers de réservations ont les même colonnes: seul l’ID du restaurant est préfixé par “air” ou “hpg” (annex 2)

3. Les deux dataset ont enregistré les réservations effectuées sur la même période de temps (annex 3)
4. Les visites ont aussi eu lieu sur la même période de temps, seuls les horaires varient de quelques heures (annex 4)
5. La distribution du nombre de visiteurs par réservations sur les deux systèmes AIR et HPG sont similaires : la plupart des réservations sont faites pour moins de 10 personnes (voir ci-dessous)

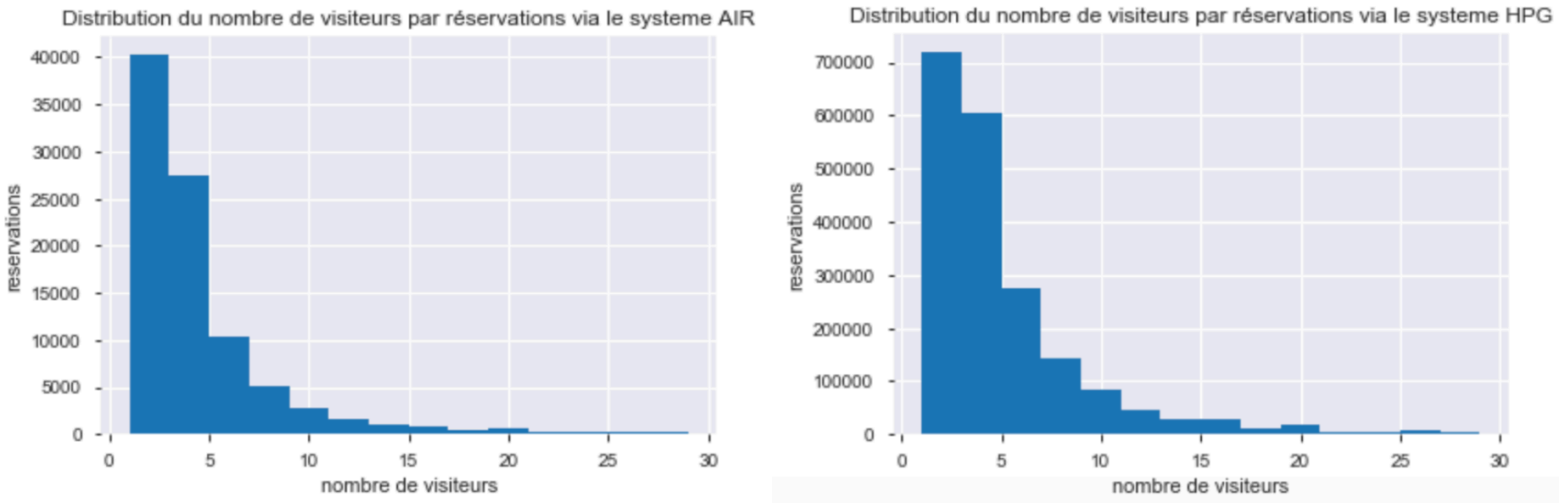


Image 2 - Distribution du nombre de visiteurs par réservations via AIR et HPG

On notera aussi que les deux dataset “ar” et “hr” contiennent tous deux une colonne “visit_datetime” et une colonne “reserve_datetime” correspondant respectivement à la date de venue au restaurant et à la date à laquelle la réservation s’est faite. Ces variables feront l’objet de création d’une variable intermédiaire “reserve_datetime_diff” (voir chapitre 5.2 sur le Feature Engineering).

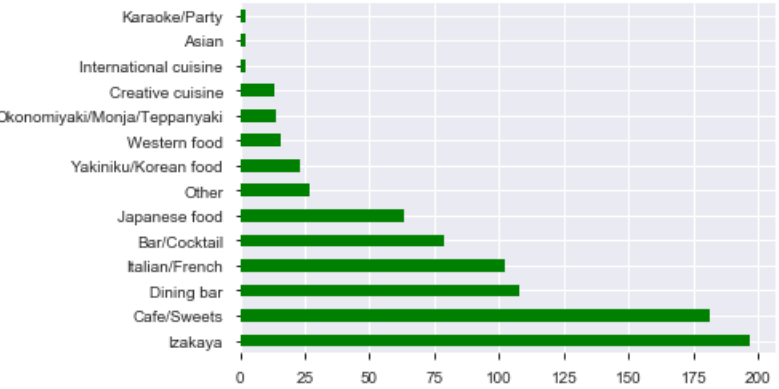
Afin de récupérer pour chaque “hpg_store_id” son equivalent dans le système AIR, “air_store_id”, il a fallut merger le dataset des réservations de HPG “hr” avec le dataset “id”. Il n’y a désormais que 28183 réservations dans le dataset “hr” (contre 2000320 avant cette opération (annex 5)

4.2 Restaurants

Dans un second temps, l’exploration a porté sur les dataset “as” et “hs” correspondant aux informations des restaurants.

Le système AIR comporte 829 restaurants de 14 types de cuisines différents et repartis dans 103 “area” du pays. Voici une répartition du nombre de restaurants uniques regroupés par

types de cuisines et les “area” dans lesquelles se trouvent le plus de restaurants inscrits sur le service AIR:



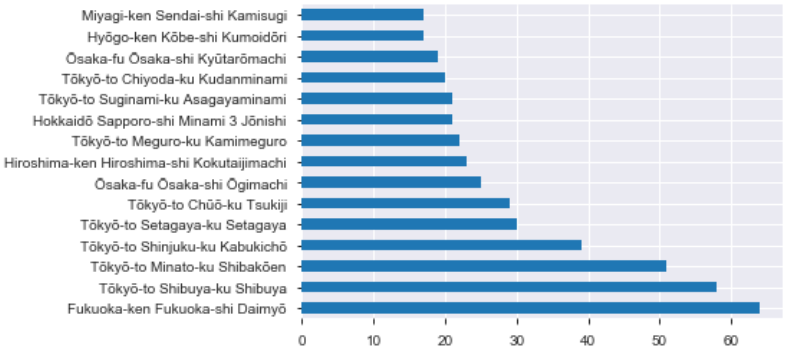
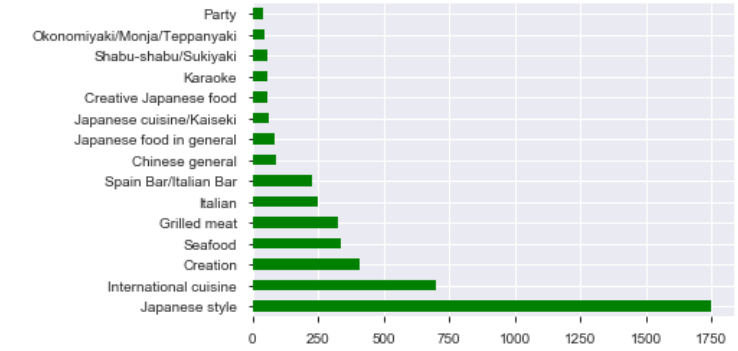


Image 3 - Répartition du nombre de restaurants uniques regroupés par types de cuisine et « area » dans lesquelles se trouvent le plus de restaurants inscrits sur le système AIR

On notera qu’il existe beaucoup de restaurants “Izakaya” (equivalents du bistro en France ou d’un pub au Royaume-Uni) et de Cafe. Il y aurait aussi plus de restaurants français/italiens que de restaurants a cuisine japonaise, surprenant. Enfin, “creative cuisine” est un type de cuisine qui suscite la curiosité.

Le système HPG comporte 4690 restaurants dans 34 genre de cuisines différentes et répartis dans 119 “area” du pays (seules les 15 premieres sont affichées):



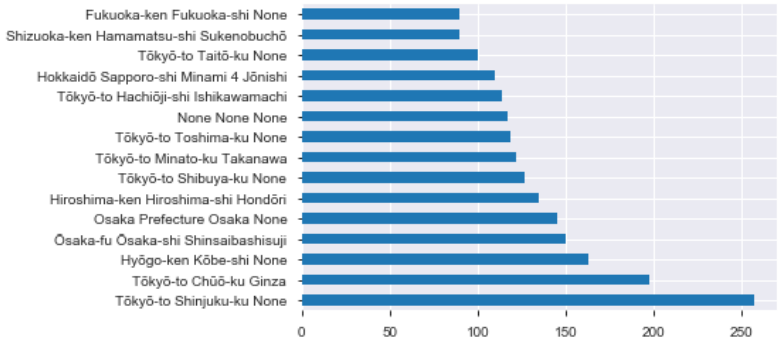


Image 4 - Répartition du nombre de restaurants uniques regroupés par types de cuisine et « area » dans lesquelles se trouvent le plus de restaurants inscrits sur le système HPG

Les deux systèmes proposent des types de cuisine assez similaires mais présentent quand même quelques écarts. Concernant les “area_name”, on remarquera que les valeurs des deux systèmes sont aussi bien différentes: “Tokyo-to Shinjuku-ku None” vs “Tokyo-to Shinjuku-ku Kabukicho” réfèrent-elles toutes deux a la ville de Tokyo ? Ce sujet est a investiguer.

Nous pouvons noter via cette étude qu’il y a bien un écart significatif entre le nombre de restaurants présents dans le système AIR (829) et le système HPG (4690). Grâce au fichier

store_id_relation nous allons pouvoir joindre les informations des restaurants inscrits sur les deux systèmes.

4.3 Autres dataset

Kaggle nous propose d'autres dataset que les réservations et les restaurants à étudier: **store_id_relation.csv, sample_submission.csv** et **date_info.csv**

Le premier dataset a déjà été utilisé pour récupérer pour chaque restaurant son identifiant HPG et AIR (si toutefois il est inscrit dans chacun des deux systèmes). Il contient 150 lignes ce qui représente une faible quantité par rapport au nombre de restaurants uniques dans les réservations faites via AIR (829) et HPG (4690).

Le dataset sample_submission est ce qui va nous servir de test. Il contient 32019 lignes et deux colonnes: une colonne "id" qui est un concaténation entre le "air_store_id" et une date au format *YYYY-MM-DD (comprises entre le 23 avril 2017 et le 31 mai 2017, voir annex 6),* puis la colonne "visitors" qui est le nombre de visiteurs a prédire. Toutes les valeurs de cette variable sont égales a 0. De plus, il y a 821 restaurant uniques (annex 7) sur lesquels il faut prédire le nombre de visiteurs pour chacun des jours entre le 23-04-2017 et le 31-05-2017.

Le dataset date_info.csv ("hol") nous donne des informations sur le jour de la semaine et sur les jours de vacances ("holiday_flg") entre le 01-01-2016 et le 31-05-2017. Ces données peuvent notamment influer sur le nombre de visiteurs dans les restaurants et seront a intégrer dans le modèle.

5. Data Preprocessing

Suite à l'exploration des données, il est plus facile d'appréhender le feature engineering et la modélisation.

5.1 Cleaning

Tout projet de data science implique une partie de nettoyage de données ("data cleaning") pour les rendre exploitables par tout modèle de machine learning. C'est une étape qui peut s'avérer fastidieuse dans un projet mais qui est un pré-requis essentiel pour la modélisation. Toutes les données fournies par Kaggle sont de très bonne qualité; grâce au package pandas profiling j'ai pu verifier qu'il n'y ait, par exemple, aucune valeur manquante dans chacun des dataset ou qu'il n'y ait pas de valeurs dupliquées et/ou aberrantes.

En terme de nettoyage seules des modifications sur les "air_area_name" et "air_genre_name" ont été effectuées afin de retirer les caractères tels que "/" et "-".

La partie cleaning s'est donc avérée plutôt rapide dans le cadre de ce projet.

5.2 Feature Engineering

Récupérer les données présentes dans les dataset et créer une variable cible ne suffisent pas à faire un modèle suffisamment prédictif: nous voulons extraire le plus d'information de nos dataset pour apporter le plus de valeur possible. C'est pourquoi l'étape de création de variables intermédiaires ("Feature Engineering") est cruciale pour la suite du projet.

5.2.1 Les variables temporelles: les dates

En utilisant la commande "dtypes" il est possible de voir les types de variables de chacun des dataframe pandas. On remarquera qu'il y a très souvent des dates (jour + horaires) desquelles il faut créer des variables catégorielles pour les intégrer dans un modèle. Les dates apparaissent en tant qu'objet qu'il faut transformer en "datetime" grâce à la commande "*pd.to_datetime".* Ainsi pour les colonnes "visite_datetime" et "reserve_datetime" des data frame "ar" et "hr" nous avons seulement la date[1] au format "YYYY-MM-DD".

Sachant que tout modèle machine learning n'accepte en entrée que des valeurs numériques, il faut décomposer les dates en jour, mois et année comme effectué sur la variable "visit_date" du data frame de train ['ar'], qui correspond aux dates de visite des réservations. Ainsi grâce aux commandes ci-dessous, nous avons une nouvelle colonne pour le jour de la semaine, le mois et l'année des visites.

```
data['tra']['visit_date'] = pd.to_datetime(data['tra']['visit_date'])
data['tra']['dow'] = data['tra']['visit_date'].dt.dayofweek
data['tra']['year'] = data['tra']['visit_date'].dt.year
data['tra']['month'] = data['tra']['visit_date'].dt.month
```

Image 5 - Décomposition des dates

Comme évoqué ci-dessus, seulement les valeurs numériques étant acceptées, le jour de la semaine se transformera de la manière ci-contre: 0 pour Lundi, 1 pour Mardi, …, 6 pour Dimanche.

Dans le dataset de test ['tes'], la colonne 'visit_date' se trouve dans l'id de la réservation. Ainsi nous appliquerons une fonction qui va séparer l'id du restaurant de la date de réservation avec les commandes ci-dessous:

```
data['tes']['visit_date'] = data['tes']['id'].map(lambda x: str(x).split('_')[2])
data['tes']['air_store_id'] = data['tes']['id'].map(lambda x: '_'.join(x.split('_')[:2]))
```

Image 6 - Extraction des de l'id restaurant et de la date de réservation

---

[1] A noter: Il a été choisi volontairement de ne pas récupérer les horaires de chacune des reservations dans le cadre de ce projet.

5.2.2 Les variables catégorielles

Une fois que les variables temporelles ont été traitées et transformées, il a fallût trouver les variables catégorielles de chacun des dataset.

Cette étape concerne les variables “air_area_name” et “air_genre_name”. Comme évoqué dans le chapitre 4 sur l’exploration des données, il existe 14 genres de restaurants et 103 types d’area_name. Les valeurs étant en plus sous forme de texte, on utilisera un LabelEncoder pour les convertir en variable catégorielles. Il serait peu judicieux d’utiliser un LabelEncoder sur les 103 noms d’area. C’est pourquoi nous n’allons garder que le premier élément des valeurs des “area_name” pour ne garder que le nom de la ville.

5.2.3 Les nouvelles variables créées

Une fois avoir traité les données déjà présentes dans les dataset, l’étape suivante est de la création de nouvelles variables.

Suite à l’exploration des données effectuée en debut de projet, la première variable a créer était la durée (en jours[2]) qui s’écoulait entre une réservation et une visite. La variable “reserve_datetime_diff” est donc une premiere variable ajoutée dans les data frame de réservations. L’hypothèse émise ici est qu’il pourrait y avoir une corrélation entre cette durée et le nombre de visiteurs à prédire dans certains restaurants à certains jours.

Le projet ayant pour objectif de prédire le nombre de visiteurs dans des restaurants, des nouvelles variables concernant les visiteurs cette fois-ci ont été créées:

min_visitors : le nombre minimum de visiteurs par restaurant et par jour de semaine

mean_visitors : le nombre moyen de visiteurs par restaurant et par jour de semaine

median_visitors : le nombre median de visiteurs par air_store_id et par jour de semaine

max_visitors : le nombre maximum de visiteurs par air_store_id et par jour de semaine

count_observations : somme du nombre de visiteurs par air_store_id par jour.

dow (“Day Of Week”) : jours de la semaine en variables numériques

rs1 : somme des délais (en jours) écoulés entre le jour de réservation et le jour de visite de visiteurs pour chacun des air_store_id par jour de la semaine

rs2 : moyenne des délais (en jours) écoulés entre le jour de réservation et le jour de visite de visiteurs pour chacun des air_store_id par jour de la semaine

---

[2] *Il a été une nouvelle fois décidé de ne pas prendre en compte les heures*

rv1 : somme du nombre de visiteurs par jour par air_store_id

rv2 : moyenne du nombre de visiteurs par jour par air_store_id

total_reserv_sum : somme du nombre de jours d'écart entre réservation et visite au restaurant

total_reserv_mean : moyenne du nombre de visiteurs par réservations

total_reserv_dt_diff_mean : moyenne du nombres de jours d'écart entre reservation et visite au restaurant

date_int : date de visite en integer (au format YYYYMMDD)

Toutes les variables citées ci-dessus ont été intégrées au train et au test afin d'avoir le même format "shape", pré-requis essentiel a la modélisation présentée dans le chapitre ci-dessous.

A noter: Les données géographiques des restaurants (longitude et latitude) n'ont fait l'objet d'aucun feature engineering dans le cadre de ce projet et font partie des axes sur lequel il aurait pu être interessant d'approfondir l'analyse. En effet, étudier les zones géographiques (zones urbaines, zones rurales) des restaurants pourrait avoir un impact sur le modèle.

6. Modélisation

6.1 Algorithmes Linéaires

L'objectif étant de prédire le nombre de visiteurs (variable continue) par restaurants, il s'agit ici d'une régression dans un apprentissage supervisé.

Une fois les étapes de data cleaning et feature engineering terminées nous avons des données prêtes pour créer un modèle de machine learning. En utilisant la fonction "train_test_split" fournie par scikit-learn, deux dataset ont été créés : un de "train" sur lequel nous allons entraîner notre modèle et un de "test" sur lequel nous allons évaluer sa performance. Le premier contient 80% des données et le deuxième en contient 20%.

```
y = train.pop('visitors') # On drop la colonne à prédire de y
X_train, X_test, y_train, y_test = train_test_split(train[col], y,
test_size=0.2, random_state=42)
```

```
X_train :  (201686, 50)
X_test : (50422, 50)
y_train : (201686,)
y_test : (50422,)
```

X correspond aux **variables explicatives** et **y** à la **variable cible**.

Image 7 - Shape des données de X_train, X_test, y_train et y_test

La métrique d'evaluation est définie par Kaggle, il s'agit de la RMSE : Root Mean Squared Error. Elle correspond à la racine carrée de la moyenne du carré des erreurs. C'est une unité de mesure fréquemment utilisée[3] dans les problèmes de regression qui calcule la difference entre les valeurs prédites (yî) par un modèle (ou un estimateur) et les valeurs réellement observées (yi).

$$RMSE = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2}$$

Image 8 - Formule RMSE

Les données étant prêtes, la métrique d'evaluation définie, nous allons pouvoir créer différents modèles, les comparer, les améliorer et choisir le plus pertinent dans le cadre de ce projet. Les sous-chapitres ci-dessous détaillent chacun des modèles testés.

6.1.1 La régression linéaire

Les premiers modèles testés sont des modèles linéaires qui sont adaptés à des larges dataset et des données à haute dimension.

Nous savons que les données de training sont utilisées pour estimer les paramètres d'un modèle supervisé de machine learning. La regression linéaire cherche les paramètres qui vont minimiser la RMSE entre les prédictions et les variables observées du dataset de training. Une régression linéaire simple peut aussi être utilisée pour modéliser une relation linéaire entre une variable de réponse, ici le nombre de visiteurs a prédire, et une ou plusieurs variable(s) explicative(s).

$$y_i = \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} + \varepsilon_i$$

Image 9 - Formule mathématique de la régression

où y_i est la variable prédite, x_i la valeur des variables explicatives, β un vecteur de paramètres inconnus et ε les erreurs entre les variables prédites et celles observées.

Il est possible d'utiliser la classe de scikit-learn "sklearn.linear_model.LinearRegression".

```
linreg = LinearRegression().fit(X_train, y_train)
linreg_pred = linreg.predict(X_test)
```

Image 10 - sklearn.linear_model.LinearRegression

Dans un premier temps nous utilisons la méthode "fit" qui permet d'apprendre les paramètres du modèle, puis la méthode "predict" qui pourra prédire le nombre de visiteurs.

```
print("training set score: %f" % linreg.score(X_train, y_train))
print("test set score: %f" % linreg.score(X_test, y_test))

## -- End pasted text --
training set score: 0.579304
test set score: 0.565060
```

Image 11 - Résultats Regression Linéaire

[3] La RMSE permet de punir plus sévèrement les grandes erreurs par rapport a la MSE, aussi très souvent utilisée pour les problèmes de régression.

Nous observons que le score de test, R^2[4] = 0.565, n'est pas un score performant. En revanche, le fait que le score de train et de test soient quasiment égaux permet d'affirmer qu'il est très probable que le modèle a "underfitté", c'est-à-dire qu'il ne peux correctement pas modéliser le dataset de training ni se généraliser à de nouvelles données. Les résultats de ce modèle étant faibles et ses performances sur le dataset de training insuffisantes, il ne sera pas retenu. Il est donc préférable d'essayer d'autres modèles.

Il est aussi possible de voir, d'un modèle linéaire "fitté", le poids des coefficients de chacune des 50 variables explicatives grâce à "lingreg.coef_" (voir annexe 8) et il est aussi possible de plotter le modèle (annexe 9).

Un score R^2 = 0.565 signifie que 56% de la variance du dataset de test est expliqué par le modèle. La performance peut changer si le dataset entier était partitionné d'une autre manière, c'est-a-dire avoir un dataset de train de 75% de différentes données. C'est pourquoi nous utilisons une validation croisée pour produire une nouvelle estimation de la performance de ce score.

```
scores = cross_val_score(linreg, X_train, y_train, cv=5)
```

Chaque tour de validation croisée permets d'entraîner de tester différentes partitions des données pour réduire la variabilité et peux nous donner une meilleure idée de la performance de l'algorithme.
Nous pouvons observer que les scores de validation croisée du 1er (0.6), 2e (0.584) et 4e (0.588) fold ont légèrement améliorer le score du modèle original (0.565)

```
Cross-validated scores: [ 0.60027265  0.58395875  0.56089625  0.5884316   0.50015356]
```

A titre d'exemple, voici les résultats des 10 premieres observations et leurs prédictions :

```
First 10 Predictions [ 17.65879935  11.49043542  22.94236448  11.22690248  14.88090359
  16.98326204  24.75024193  10.08335482  10.86074349  44.18380776]
First 10 Real Values 160503    33
74437     20
93611     24
36756     16
61315     10
99936     42
189042    48
78456      9
180807    11
47721     12
```

Image 12 - Les premières dix observations et leurs prédictions

[4] *Le R^2 permet d'avoir une idée générale de la performance du modèle en comparant l'écart à la moyenne de la variable y à l'écart de la prévision dans le cadre d'un modèle conditionnel.*

On note que certaines des prédictions sont assez éloignées de la réalité, ce qui nous laisse encore des possibilités pour améliorer notre modèle.

Le score RMSE de Ridge Regression = 11.2271

6.1.2 Ridge Regression

Au vu des résultats décevants de la regression linéaire, essayons une régression Ridge afin de comparer les résultats. Elle repose sur la même formule mathématique que la précédente, en essayant de prédire non seulement sur le dataset de training mais aussi sur une contrainte additionnelle. Cela signifie qu'indirectement chaque feature devrait avoir le moins d'impact possible sur la prédiction.

```
In [36]: ridge = Ridge().fit(X_train, y_train)
    ...: print("RidgeRegression training set score: %f" % ridge.score(X_train, y_train))
    ...: print("RidgeRegression test set score: %f" % ridge.score(X_test, y_test))
    ...:
RidgeRegression training set score: 0.578840
RidgeRegression test set score: 0.564767
```

Image 13 - Résultats de la Ridge Regression

Les résultats des erreurs de training et test (via le score r2) sont les même que précédemment. La méthode ridge n'a donc pas permis d'améliorer le modèle, d'autant plus que le RMSE de Ridge Regression (= 11.2308) n'est pas meilleur que le celui de la regression linéaire

6.1.3 Lasso Regression

Il existe une alternative a Ridge pour regulariser une regression linéaire: Lasso, qui va restreindre les coefficients d'être proche de 0 mais d'une manière différente (appelée la regularisation "l1"). De ce fait, certains coefficients vont être exactement égaux a 0, ce qui implique que certaines variables soient ignorées par le modèle. C'est une approche similaire a une "feature selection" que nous aborderons un peu plus tard. Avoir des coefficients exactement égaux permets souvent de plus facilement interpréter le modèle et peux révéler les features les plus importantes.

```
from sklearn.linear_model import Lasso
lasso = Lasso().fit(X_train, y_train)
print("training set score: %f" % lasso.score(X_train, y_train))
print("test set score: %f" % lasso.score(X_test, y_test))
print("number of features used: %d" % np.sum(lasso.coef_ != 0))

## -- End pasted text --
training set score: 0.574947
test set score: 0.561208
number of features used: 14
```

Image 14 - Résultats de la regression Lasso

Nous observons de nouveau des résultats semblables aux précédents. L'information à prendre en compte est le nombre de feature utilisées dans le modèle :

seulement 14/50. L'hypothèse que j'émets à ce niveau du projet est que les étapes de feature engineering et/ou de cleaning ont été mal effectuées aussi bien que la RMSE de Lasso = 11.2767 se rapproche des deux précédentes et ne permets pas de s'arrêter a ce stade du projet.

6.2 Algorithmes non linéaires

6.2.1 Random Forest

Apres avoir testé des algorithmes linéaires, nous allons essayer des algorithmes non linéaires. Dans un premier temps nous allons utiliser les Random Forest. C'est une collection d'arbres de décision qui ont été entraînés sur des sous-ensembles sélectionnés, de manière aléatoire, des données d'apprentissage et des variables explicatives. De manière générale, les prédictions des forêts aléatoires retournent la moyenne ou le mode de la prédiction de ses arbres qui la constituent.

A l'aide de la classe "sklearn.ensemble.RandomForestRegressor", sans aucun paramètres, nous allons évaluer un premier score.

```
rf = RandomForestRegressor(n_jobs=-1)
rf.fit(X_train, y_train)
rf.predict(X_test)
print("RandomForest training score: ", rf.score(X_train, y_train))
print("RandomForest test score: ", rf.score(X_test, y_test))

## -- End pasted text --
RandomForest training score:  0.921750611909
RandomForest test score:  0.530092238426
```

Image 14 - Résultats de Random Forest

Une fois que le modèle a "fitté" sur X_train et y_train, il prédit sur X_test. A noter que le "n_jobs = -1" permet de faire des traitements plus rapides en les exécutant sur tous le maximum de processeur.

Nous observons une forte difference entre le score de training (0.92) et de test (0.53). Ceci veut notamment dire que le modèle a trop bien appris sur le dataset d'entraînement mais qu'il ne prédit pas correctement sur des nouvelles données.

Nous avons une RMSE semblable aux précédentes (= 11.7). Il n'y a donc pas, à première vue, d'amélioration à utiliser une méthode non linéaire.

Enfin, il est intéressant de pouvoir déceler les variables qui ont le plus de poids dans le modèle ("feature importance"), afin de comprendre quelles sont celles qui influent le plus sur le nombre de visiteurs prédits (voir annexes 10 et 11).

Les résultats obtenus ci-dessus sont peu satisfaisants. Nous pouvons émettre l'hypothèse qu'il faille spécifier les paramètres dans le RandomForestRegressor pour tuner le modèle précédent. L'utilisation de la méthode GridSearch permet de parcourir et tester toutes les combinaisons de paramètres afin de trouver la plus optimale. C'est une étape coûteuse en ressources (%CPU utilisé) et qui peut être longue (plusieurs heures); lancer un GridSearch en tâche de fond (en utilisant la commande "nohup") a été un gain de temps non négligeable.

Seuls les paramètres max_depth et n_estimators ont fait l'objet d'un GridSearch. Le premier correspond à la profondeur maximal de chaque arbre et le deuxième au nombre d'arbres a entraîner. Avec max_depth = 11 et n_estimators = 100, en résultera un écart moins significatif entre le score d'erreur sur le training et sur le test et un RMSE légèrement meilleur

```
RandomForest GridSearch training score:  0.66134201745
RandomForest GridSearch test score:  0.580678697634
RandomForest RMSE with GridSearch: 11.0236
RandomForest RMSE without GridSearch: 11.7006
```

Image 15 - Résultats de Random Forest avec les paramètres de GridSearch

Suite au tuning du modèle à l'aide des hyper-paramètres, nous avons cette fois-ci RMSE = 11.02, ce qui une très légère amélioration.

Nous allons désormais utiliser l'algorithme "Gradient Boosting" afin d'étudier son comportement de prédiction et vérifier s'il est plus performant que les précédents.

6.2.2 Gradient Boosting

Les algorithmes de descente de gradient sont souvent utilisés dans ce genre de problème de par leur performance.

La classe fournie par scikit-learn "sklearn.ensemble.XGBRegressor" nous permet de tester la meme démarche (naive) que pour RandomForestRegressor: nous exécutons un premier modèle sans spécifier d'hyper-paramètres puis nous verrons comment optimiser le score en calculant les hyper-paramètres.

Le premier modèle nous donne les résultats ci-dessous:

```
XGB training score:  0.6003
XGB test score:  0.5783
XGBoost test RMSE :  11.055
```

Image 16 - Résultats de XGBoost

Nous pouvons constater que l'algorithme réagit correctement entre les données de train et de test. Lorsque nous comparons avec l'algorithme RandomForest, les résultats sont quasiment égaux en terme de RMSE (11.055 et 11.0236).

Essayons de trouver la meilleure combinaison d'hyper-paramètres à la classe XGBRegressor via la méthode GridSearch. Cette étape est cruciale pour que le modèle fournisse une meilleure performance de generalisation.

Après plusieurs tentatives et au vu du coût de cette opération, il m'a semblé plus judicieux de ne faire la recherche que sur le paramètre « max_depth » (compris entre 5 et 10), avec un learning_rate = 0.1 et un n_estimators = 100.

GridSearch a trouvé max_depth = 8, voici les résultats obtenus :

```
## -- End pasted text --
XGB training score:  0.6889
XGB test score:  0.5965
XGBoost with Grid Search RMSE : 10.8137
```

Image 16 - Résultats de XGBoost avec les paramètres GridSearch

Ces résultats ont donné une amélioration de seulement +1.82%

Il existe un autre méthode utilisée pour rechercher les hyper-paramètres : «hyperopt ». Cette fois-ci, il y a plusieurs autres paramètres sur lequel la recherche a été effectuée, tels que le gamma :

```
xgbr_d = {'gamma': hp.quniform('gamma', 0.0, 5.0, 0.1),
          'learning_rate': hp.choice('learning_rate', [0.1]),
          'colsample_bytree': hp.quniform('colsample_bytree',
                                          0.3,
                                          1.,
                                          0.05),
          'max_depth': hp.choice('max_depth', list(range(5, 10))),
          'min_child_weight': hp.quniform('min_child_weight', 1., 5., 1),
          'subsample': hp.quniform('subsample', 0.6, 1, 0.05),
          'nthread': hp.choice('nthread', [-1]),
          'n_estimators': hp.choice('n_estimators', [1500]),
          'objective': hp.choice('objective', ['reg:linear']),
          'reg_lambda': hp.quniform('reg_alpha', 0.0, 4.0, 0.1),
          'reg_alpha': hp.quniform('reg_lambda', 0.0, 4.0, 0.1)}
```

Image 17 - Paramètres a optimiser via la méthode hyperopt

La fonction « early_stopping » a permis d'optimiser cette recherche et d'écourter son temps de calcul (de plusieurs heures a quelques minutes). Plus d'arbres sont entraînés, plus le risque de sur-apprendre sur le dataset d'entraînement est probable. Early stopping permets de nous informer d'un dataset de validation et du nombre d'itérations après lequel l'algorithme devrait s'arrêter si le score sur ce dataset de validation ne s'est pas amélioré.

Les résultats sont légèrement meilleurs avec hyperopt qu'avec GridSearch :

```
XGBoost Hyperopt train error: 0.686
XGBoost Hyperopt test error: 0.6009
XGBoost Hyperopt RMSE : 10.754
Hyperopt error: 3.2787161641132374
```

Image 18 - Résultats de XGBoost avec les hyperopt

La métrique d'evaluation (RMSE) est ici meilleure que toutes les autres auparavant trouvées : 10.754

Nous avons donc utilise plusieurs algorithmes (linéaires et non linéaires). Certains se sont avérés bien moins prédictifs que d'autres, mais il était intéressant de pouvoir se pencher sur chacun de leurs résultats afin de les comparer.

Ci-dessous les résultats des algorithmes utilisés dans le cadre de ce projet. On notera une légère différence entre les algorithmes linéaires et non linéaires, ces derniers étant un peu moins performants. Nous pouvons observer que la métrique d'évaluation, RMSE, s'améliore lorsque XGBoost est utilisé. A fortiori, les scores d'erreurs sur le train et le test sont meilleurs. Avec XGBoost, il y a 32% d'erreur sur les données d'apprentissage et environ 40% sur les données de test.

Enfin, nous pouvons conclure que les prédictions effectuées grace a ce modèle, prédisent a +/- 10 visiteurs d'erreur par visite. C'est un taux d'erreur non négligeable lorsqu'on sait qu'en moyenne, il y avait en moyenne 5 visiteurs par reservations dans les données source (voir annexe 12)

	Train/Test	RMSE
Linear Regression	0.5793 / 0.5651	11.2271
Ridge Regression	0.5788 / 0.5648	11.2308
Lasso	0.5749 / 0.5612	11.2767
RandomForest	0.6613 / 0.5807	11.0236
XGBoost	0.686 / 0.6	10,754

Enfin, il m'a semble intéressant de pouvoir représenter sur un graph la différence des valeurs prédites par rapport à celles réellement observées (voir-ci contre + annexe 13)

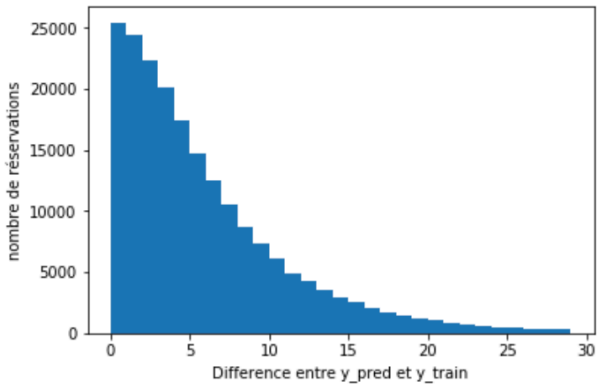


Image 19 - Différence entre y_pred et y_train

Conclusion

Ce projet a été un réel challenge qui m'a permis de comprendre tout le processus d'un projet data science, de la premiere étape de data cleaning a l'étape de modélisation. Les différents sujets abordes lors du DSSP7 m'ont aidé à mieux apprehender le sujet; la prise en main de python fut un pré-requis nécessaire au déroulement du projet. Les chapitres concernant le machine learning m'ont aussi mieux permis de developper ce projet. Enfin, le 1er data camp mené en groupe, se rapproche du sujet que j'ai choisi et m'a aidé a retrouver des points abordes lors du data camp.

il y a plusieurs axes sur lesquels j'aurais voulu avoir plus de temps pour pouvoir améliorer la phase modélisation. La premiere erreur effectuée fût de ne pas me retourner vers l'étape de feature engineering une fois la modélisation terminée. Les scores auraient pu être améliorés en revoyant les étapes de feature engineering. En effet, en imprimant le poids de chacune des variables des différents modèles (annexe 10), on constate une forte inégalité de poids entre la 1ere et la 2e et surtout des variables qui n'apportent très peu voire aucune valeur au modèle (voir annexe 10).

Il aurait aussi pu être judicieux de commencer par un premier modèle « Quick and dirty » fin d'évaluer la prédictivité d'un modèle avant/après feature engineering. De plus, dans le cadre de ce projet, aucun score n'a été soumis au leaderboard Kaggle. Il est donc difficile de pouvoir comparer ses résultats aux membres de la competition.

Annexes

1.

```
data['ar'].shape
(92378, 4)

data['hr'].shape
(2000320, 4)
```

2.

```
In [17]: data['ar'].columns
Out[17]:
Index(['air_store_id', 'visit_datetime', 'reserve_datetime',
       'reserve_visitors'],
      dtype='object')

In [18]: data['hr'].columns
Out[18]:
Index(['hpg_store_id', 'visit_datetime', 'reserve_datetime',
       'reserve_visitors'],
      dtype='object')
```

3.

```
print(min(data['ar'].reserve_datetime),max(data['ar'].reserve_datetime))
print(min(data['hr'].reserve_datetime),max(data['hr'].reserve_datetime))

## -- End pasted text --
2016-01-01 01:00:00 2017-04-22 23:00:00
2016-01-01 00:00:00 2017-04-22 23:00:00
```

4.

```
print(min(data['ar'].visit_datetime),max(data['ar'].visit_datetime))
print(min(data['hr'].visit_datetime),max(data['hr'].visit_datetime))

## -- End pasted text --
2016-01-01 19:00:00 2017-05-31 21:00:00
2016-01-01 11:00:00 2017-05-31 23:00:00
```

5.

```
data['hr'].shape
(2000320, 4)

data['hr'] = pd.merge(data['hr'], data['id'], how='inner', on=['hpg_store_id'])

data['hr'].shape
(28183, 5)
```

6.

```
In [22]: date_test = data['tes']['id'].str[-10:]

In [23]: print(min(date_test))
2017-04-23

In [24]: print(max(date_test))
2017-05-31
```

7.

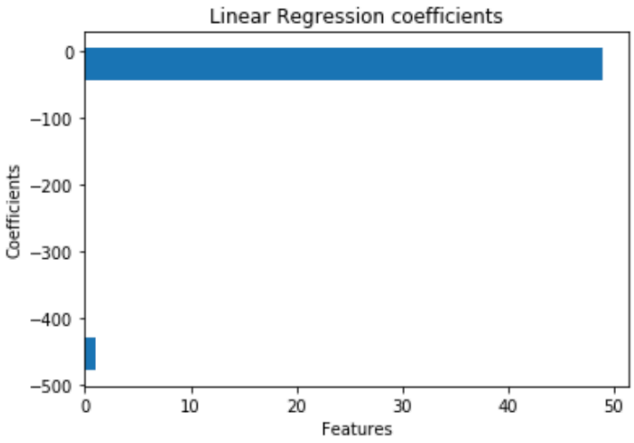
```
In [29]: unique_restaurants_test = data['tes']['id'].str[:-11]

In [30]: print(len(unique_restaurants_test.unique()))
821
```

8.

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.hist(linreg.coef_, orientation=u'horizontal')
plt.title("Linear Regression coefficients")
plt.xlabel("Features")
plt.ylabel("Coefficients")
```

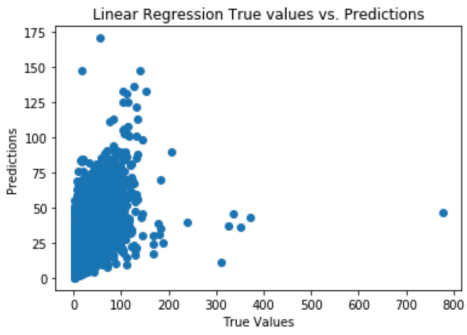
```
<matplotlib.text.Text at 0x11e3c1748>
```



9.

```
plt.scatter(y_test, linreg_pred)
plt.xlabel("True Values")
plt.ylabel("Predictions")
plt.title("Linear Regression True values vs. Predictions")
```

```
<matplotlib.text.Text at 0x11fc91a20>
```

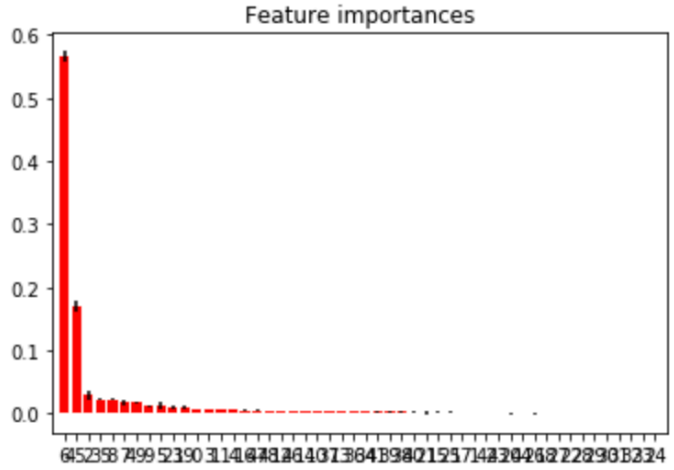


10.

```
feature_import = pd.DataFrame(data=rf.feature_importances_, index=X_train.columns.values, columns=['values'])
feature_import.sort_values(['values'], ascending=False, inplace=True)
feature_import
```

	values
mean_visitors	0.574544
date_int	0.167078
month	0.028307
rv1_x	0.023451
max_visitors	0.022363
median_visitors	0.018189
air_store_id2	0.017369
count_observations	0.012910
min_visitors	0.011730
air_area_name4	0.009378
air_area_name2	0.008394
dow	0.007861
air_area_name	0.007241
day_of_week	0.006997
holiday_flg	0.006599
air_genre_name1	0.006101
var_max_lat	0.005346
latitude	0.005304
var_max_long	0.005097
lon_plus_lat	0.004921
air_genre_name	0.004792
longitude	0.004709
air_genre_name0	0.004598
rs2_x	0.004430
rv2_x	0.004283
rs1_x	0.003644
rv1_y	0.003307
rv2_y	0.002966
rs1_y	0.002929
rs2_y	0.002860
air_area_name0	0.002111
air_area_name1	0.001613
air_area_name5	0.001416
year	0.001362
air_area_name3	0.001279
total_reserv_mean	0.000877
total_reserv_sum	0.000678
total_reserv_dt_diff_mean	0.000594
air_genre_name9	0.000592
air_genre_name6	0.000592
air_area_name7	0.000566
air_genre_name2	0.000375
air_area_name6	0.000246
air_area_name9	0.000000
air_genre_name3	0.000000
air_genre_name5	0.000000
air_genre_name4	0.000000
air_genre_name7	0.000000
air_genre_name8	0.000000
air_area_name8	0.000000

11.



12.

```
np.mean(data['ar']['reserve_visitors'])
```

```
4.4817489012535452
```

```
np.mean(data['hr']['reserve_visitors'])
```

```
5.0737846944488885
```

13.

```
np.histogram(np.abs(y_pred-y_train),range(100))
```

```
(array([25425, 24415, 22315, 20061, 17391, 14710, 12557, 10528,  8675,
         7293,  6070,  4924,  4224,  3523,  2978,  2541,  2068,  1723,
         1460,  1200,  1076,   817,   707,   612,   517,   473,   375,
          371,   298,   247,   225,   184,   178,   151,   123,   114,
           79,    97,    92,    61,    63,    65,    53,    34,    43,
           53,    27,    29,    23,    22,    23,    26,    24,    21,
           16,    18,     8,    19,    18,    16,    12,     9,     7,
            9,     6,     8,     7,     5,     7,     8,     6,     4,
            8,     8,     2,     2,     3,     2,     4,     6,     2,
            4,     3,     2,     5,     2,     2,     2,     3,     0,
            4,     3,     2,     2,     2,     0,     2,     0,     2]),
 array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
        34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
        51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
        68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
        85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99]))
```

Il y a 24525 reservation pour lesquelles, il n'y a pas d'erreur, 24415 où il y a 1 visiteur d'erreur, 22315 où il y a 2 visiteurs d'erreurs etc…