Luximon Davina
E4

Rapport - “Guess who's coming (back) to dinner”

I. Traitement des données

Premièrement, il a fallu traiter les données, car certaines colonnes telles quelles n’était pas exploitables et ne donnaient aucunes informations pertinentes. Les quatre fichiers csv contenaient des données à transformer et traiter.

a. Fichier restaurant.csv

En premier, sur le notebook, le fichier restaurant.csv a été traité en premier. J’ai d’abord supprimé les colonnes *abbr*, *tel*, *name*, *cityarea*, *timezone* car jugées inutiles, en effet *abbr*, *tel* et *name* ne donnent aucunes informations au modèle. De plus cityarea contenait beaucoup trop de NaN et donc faire un fillna n’aurait pas été judicieux. Enfin la colonne timezone avait la même valeur sur toutes les lignes et donc n'apporte pas d’informations qui permettent de différencier ce qu’on veut.

Maintenant deux colonnes qui avaient l’air intéressantes à traiter étaient *cdate* et *opening_hours*. En suivant l’ordre du notebook, j’ai d’abord essayé de transformer la colonne opening_hours en entiers (en minutes) de temps d’ouverture. Comme on peut le voir, il y a la présence de caractères chinois, j’ai donc premièrement récupéré toutes les heures en utilisant les formats. J’ai ensuite supprimé tous les doublons, car après avoir regardé, certaines lignes étaient sous forme 14:00-14h30(14:00 + caractères), il a donc fallu les enlever. Enfin j’ai du les convertir en minutes, les soustraire une à une en faisant attention aux moments ou l’heure de fermeture était plus petite de l’heure d’ouverture (ex : 18:00-00h30). Dans ce cas il fallait rajouter une journée entière au 00h30. Après un peu moins d’une dizaine de fonctions pour traiter cette colonne, elle avait des valeurs sous la formes suivante : 390, 2700, 1020, etc…

Pour *cdate* c’est la fonction ConvertDateTime qui s’en occupe, car elle contient une date et une heure, que nous allons transformer en nombre de jours écoulés depuis cette date. Elle transforme toutes les dates ayant le bon format grâce à re.compile et sinon renvoie (pour des facilités de traitement), après ceci tous les NaN sont remplis avec fillna par la moyenne de cette colonne. Ce sont tous les changements que j’ai trouvé pertinent pour le fichier restaurant.csv.

b. Fichier member.csv

Pour le fichier member.csv, le traitement a été beaucoup plus simple, j'ai d'abord supprimé la colonne *gender* car déjà présente dans le fichier train.csv, donc il n'y avait aucun intérêt à la garder. De plus, la colonne birthdate, donc date de naissance de la personne inscrite, était sous forme mm/dd/yyyy, et sinon 0000-00-00. Donc il a fallu adapter notre fonction ConvertDateTime à ce format, c'est donc pourquoi il y a la fonction ConvertDate, dont le format ne prend pas en compte une heure. De la même façon, si le format ne correspond pas (ex : 0000-00-00), la cellule était mise à NaN et après avoir converti toutes les lignes, toutes celles mises à NaN étaient fillna par la moyenne de la colonne. Enfin la colonne city a été supprimée, car après vérification, elle ne contenait pas le même nombre de villes lors du merge avec train et test, rendant la prédiction impossible. Pour des raisons de simplicité, les noms des villes n'ont pas été traduites dans le dataframe, mais je les ai regardées et traduites une à une afin de voir quelles étaient les différences après les merge. Ce sont toutes les modifications qui ont été apportées au fichier member.csv.

c. Fichier train.csv

Pour le fichier train.csv, j'ai d'abord regardé où étaient les NaN avant le merge, on voit que seulement la colonne *purpose* en contient, je les ai donc tous remplacé par "Autre" afin de ne pas avoir à les supprimer et perdre des informations. Ensuite pour les colonnes *cdate* et *datetime* la fonction *ConvertDateTimeT* a été créée car le format était encore une fois différent, en effet le premier élément était : 1/29/2014,,7:30:00,PM , il a donc fallu utiliser %p dans le format afin d'indiquer le PM/AM et prendre en compte les virgules qui étaient "en trop".
Après avoir converti ces deux colonnes, une troisième a été créée afin de faire la différence entre *cdate* et *datetime* afin d'indiquer le temps (en jours) écoulé entre le jour de réservation et le jour de la réservation. En effet, plus l'écart est grand plus la personne a de risques d'oublier, et inversement dans un restaurant très connu réservera beaucoup à l'avance etc...
Ensuite j'ai effectué les deux merge avec les fichiers restaurant.csv et member.csv, et ai supprimé tous les id, car devenu inutiles dans le dataframe. La colonne purpose contenait des valeurs différentes pour la même signification, j'ai donc créé des catégories, et il en restait 7 différentes à la fin pour simplifier un peu plus le dataframe. On se retrouve avec :

array(['Sweet day', 'Dîner entre amis', 'Repas en famille', 'Autre', 'Birthday', "Dîner d'affaires", 'Date importante'], dtype=object)

Et finalement, un *pd.get_dummies* a été effectué sur ce dataframe afin de le fournir au modèle pour le train et fit dessus.

Toutes les modifications qui ont été faites sur train.csv ont été reproduites sur le fichier test.csv afin d'avoir exactement le même nombre de colonnes et le même

type de données dans chaque. On se retrouve donc avec 69 colonnes dans train et 68 pour test ce qui permet donc d'effectuer une prédiction des données.

II. Prédictions des données

A. Régression logistique

Après avoir essayé beaucoup de modèles vus en cours, je me suis tournée vers la régression logistique car beaucoup de paramètres peuvent être optimisés. C'est donc pourquoi comme vu dans Machine Learning 2, j'ai effectué une grid Search afin de récupérer les meilleurs hyperparamètres pour notre dataframe.
Les paramètres optimaux testés ont été :
max_iter = 1000, C=0.1,class_weight= {1: 0.6, 0: 0.4}, penalty='l1', solver='liblinear'
Le temps d'exécution était d'environ 6h, code ayant tourné durant la nuit, je n'ai pas souhaité tester plus de paramètres car perte de temps assez conséquent (en effet, je ne pouvais rien faire d'autre sur l'ordinateur durant ce temps).
Le meilleur score obtenu grâce à ce code (Première partie des prédictions dans le notebook) a été de 0.67669 (Score privé). Après encore beaucoup d'essais, j'ai jugé avoir poussé la régression logistique jusqu'à son maximum et ai donc décidé d'essayer d'autres modèles.

B. Random Forest Regressor

En cherchant plus dans le cours que nous avions effectué, j'ai repris le random forest que nous avions vu. J'ai là aussi essayé de le pousser à son maximum tout en prenant en compte les capacités de mon ordinateur…

```
Entrée [66]: # Utilisation d'une RandomGrid afin de trouver les meilleurs hyperparamètres.
             # Création du modèle à 'tuner'.
             rf = RandomForestRegressor()
             # Recherche aléatoire des paramètres, en utilisant 3 fold pour la validation croisée.
             # Recherche parmi 60 différentes combinaisons, et utilisation de tous les noyaux disponibles.
             rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter = 60, cv = 3, verbose=2, random_state=4
             # Fit du model aléatoire.
             rf_random.fit(X_train, y_train)

             Fitting 3 folds for each of 60 candidates, totalling 180 fits

             [Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
             [Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed: 24.3min
             [Parallel(n_jobs=-1)]: Done 138 tasks      | elapsed: 241.3min
             [Parallel(n_jobs=-1)]: Done 180 out of 180 | elapsed: 361.7min finished
```

Cela aura pris moins de temps à l'exécution, et les meilleurs paramètres tirés sont :

```
Entrée [67]: rf_random.best_params_

    Out[67]: {'n_estimators': 1600,
              'min_samples_split': 5,
              'min_samples_leaf': 1,
              'max_features': 'auto',
              'max_depth': 10,
              'bootstrap': True}
```

Ceci m'a permis d'obtenir un score maximal de 0.68558 (score privé) et donc de maintenir l'ancienne place (3e).

C. XGboost

Enfin, j'ai voulu essayé une des dernières méthodes vu dans un des cours, le boost. Bien que je n'ai pas encore compris le principe complètement, j'ai réussi à le mettre en place dans mon code, et ai essayé d'optimiser deux des paramètres (après avoir essayé plus, mon ordinateur ne pouvait pas tenir).
Les deux paramètres sont : n_estimators=23, learning_rate=0.15.
Avec ceci j'ai obtenu mon score maximal de 0.68898.
Je pense que la route à suivre serait le XGboost, cependant je doute que mon ordinateur ne puisse tenir l'exécution de l'optimisation, c'est donc pourquoi j'ai décidé de m'arrêter ici. Ayant obtenu la 6e place, je pense que malgré des lenteurs d'exécution, j'ai réussi à comprendre énormément de principes derrière le Machine Learning.