Challenge Data : Prévision en temps réel de l'affluence à bord des train

Eline POT

12 novembre 2023

1 Observation et correction des données

1.1 Etude des p_iq_0

Afin de bien comprendre les données auxquelles on à affaire, on commence par tracer la matrice de corrélation des données. On remarque ainsi que les variables p_iq_j $(i, j \in \{1, 2, 3\})$ ont le plus de corrélation avec notre variable d'intérêt p_0q_0. La logique voudrait que les trains desservies récemment par notre station aient plus de corrélation avec p_0q_0 que les trains passés plut tôt. De même, on s'attend à ce que les stations les plus récentes de notre train aient le plus de corrélation avec p_0q_0 que les stations plus anciennes. Ici, ce raisonnement est vérifiée pour les variables p_0q_1, p_0q_2, p_0q_3. C'est à dire que plus on est proche de la station actuelle, plus la corrélation est élevée.

En revanche, cela ne se voit pas pour les variables p_1q_0, p_2q_0, p_3q_0. La matrice indique qu'il y a plus de corrélation avec la variable p_3q_0 qu'avec la variables p_2q_0. Cela est étonnant car cela signifie que le train $t - 3$ donne une meilleure idée du taux de remplissage du train t que le train $t - 2$, qui est pourtant plus récent.

On peut donc calculer des matrices de corrélation pour chaque train, et on se rend compte que ces anomalies concernent la majorité des trains. On conclut que les données sont erronées, et qu'il faut intervertir les colonnes p_1q_0 et p_2q_0 et / ou les colonnes p_2q_0 et p_3q_0 des trains concernés.

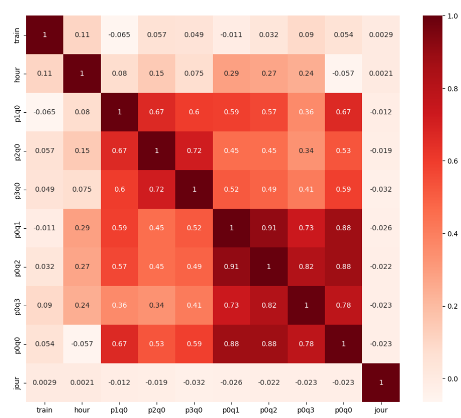

Figure 1 – Matrice de corrélation des données

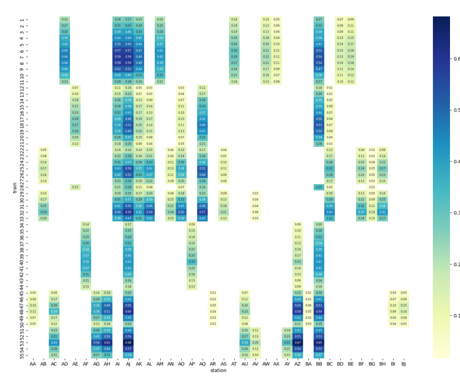

Figure 2 – Moyenne de p_0q_0 en fonction de *train* et *station*

1.2 Etude des p_0q_j

En traçant la heatmap de p_0q_0 en fonction du train et de la station comme sur la figure 2, on arrive à identifier l'ensemble des trajets possibles. On voit qu'il y a 8 trajets possibles, effectués par plusieurs trains différents chaque jour.

On se rend compte que l'ordre de visite des stations ne correspond pas à l'ordre alphabétique, et qu'il faudra déterminer cet ordre. En s'intéressant à chaque trajet, on peut déterminer l'ordre des stations desservies, puis se rendre compte que certaines colonnes p_0q_j sont erronées, et que des valeurs sont échangées pour certaines stations.

On peut corriger le jeu de données en se rappelant que, par définition, pour chaque train t et chaque station s, la variable p_0q_j donne le taux d'occupation du train t à la station $s - j$.

Une fois que les colonnes p_0q_j sont corrigées, il suffit de regarder la variable p_0q_1 de la station suivante pour avoir les prédictions des p_0q_0.

Cela fonctionne pour la majorité des cas, mais pas lorsque le train arrive à son terminus. Auquel cas, on peut appliquer un raisonnement similaire à précédemment mais en regardant le prochain train de cette même station. Les stations terminus sont les stations BB et AJ. Idéalement, il aurait fallu ranger les données p_jq_0 pour ces stations. Cela aurait été fastidieux car pour un même jour, il peut y avoir une quarantaine de trains qui passent par ces stations, et il aurait fallu déterminer l'ordre d'arrivée de ces trains, puis remettre les colonnes p_jq_0 aux bons endroits. A la place, on regarde chaque trajet, et on détermine, pour la dernière station, quel est le train et la colonne p_jq_0 qui permet d'accéder à la variable p_0q_0.

Cette méthode fonctionne pour la majorité des trains, mais pas pour le dernier train de la journée qui arrive à son terminus. Par ailleurs il reste quelques trains difficiles à prédire de la sorte, car il y a des données manquantes, auquel cas la variable p_0q_0 n'est pas accessible directement.

2 Modèles de Machine Learning

2.1 Preprocessing des données

On commence par s'intéresser à la date. On constate que les données d'entraînement et les données de test ne partagent pas de date commune. Il est donc inutile de garder la variable *date* telle quelle. A la place, on la convertit en jour de la semaine, qui pourrait déjà donner de bons renseignements. Par ailleurs, la description dans l'article de référence indique que les jours fériés ne sont pas dans les données, donc on ne peut pas filtrer les données selon cette condition.

On remarque aussi que la variable *way* ne contient qu'une seule modalité qui est 0. On peut donc l'enlever. Par ailleurs, la variable *composition* possède la valeur 2 en grande majorité sur le jeu d'entraînement, et ne contient que la modalité 2 sur le jeu de test. On enlève donc cette variable aussi. Enfin, la variable *hour* contient beaucoup de valeurs vides, et n'apporte pas beaucoup de renseignements, donc on ne la considère pas.

A ce stade du preprocessing, on a les variables suivantes : $\{$*train*, *station*, p_1q_0, p_2q_0, p_3q_0, p_0q_1, p_0q_2, p_0q_3, $jour$, $trajet$, *station_number*, *prediction*, *prediction_FIN*$\}$. La variable $jour$ correspond au jour de la semaine, $trajet$ est le numéro du trajet auquel appartient de train, *station_number* est le numéro de la station dans l'ordre du trajet du train, *prediction* est la variable p_0q_1 de la station suivante, *prediction_FIN* est la variable p_jq_0 d'un train suivant lorsqu'on est à la dernière station. On remplace les variables NAN des p_iq_j par des -1 (cela signifie qu'il n'y a pas de station précédentes ou de trains précédents). Lorsqu'on n'arrive pas à avoir de l'information pour variables *prediction* et *prediction_FIN*, on met des -1. Puis, on encode la colonne *station* avec la fonction LabelEncoder de sklearn afin d'avoir des données numériques.

2.2 Sélection du modèle

Afin de déterminer le meilleur modèle à appliquer à nos données, on utilise une méthode de Randomized Search Cross-Validation afin d'explorer de grands espaces d'hyperparamètres de manière efficace. Pour ce faire, on utilise la fonction RandomizedSearchCV de sklearn.

3 modèles de machine learning de la libraire sklearn ont été testés de cette façon : SVM (de noyau rbf car c'est le plus rapide), Random Forest et Gradient Boosting.

On sépare d'abord les données d'entraînement en des jeux de train et de validation. On utilise les données de train pour la Cross Validation et pour l'entrainement du modèle avec les meilleurs hyperparamètre, puis on determine la MAE de ce modèle sur le jeu de validation.

En procédant ainsi, on voit que la méthode de Gradient Boosting permet d'avoir les résultats les plus satisfaisants, avec une MAE sur le jeu de validation de 0.0033. Tandis que les méthodes SVM et Random Forest ont obtenus des scores de 0.0249 et 0.0038 respectivement.

La méthode de Boosting étant la meilleur, il semble pertinent d'essayer d'autres modèles basés sur cette approche, comme un modèle de d'Extreme Gradient Boosting. On utilise donc XGBRegressor de la librairie xgboost. En procédant de la même façon avec de la Randomized Search Cross-Validation, on obtient une MAE sur le jeu de validation de 0.0016.

2.3 Application du modèle

Le modèle qui permet d'avoir le plus petit score sur le jeu de validation est celui de XGBoost avec les hyper-paramètres : {'subsample' : 0.8, 'n_estimators' : 500, 'max_depth' : 15, 'learning_rate' : 0.05, 'gamma' : 0, 'col-sample_bytree' : 1.0}.

On peut donc l'appliquer aux données de test, puis remplacer les données par '*prediction*' et *prediction_FIN* lorsque cela est possible.

Cela permet d'avoir un score public final de $0,000501257349924042$ lors de la soumission.