Can You Catch It ?

IDASM104 : Projet interdisciplinaire

UNIVERSITÉ DE NAMUR

Jean ALBRECQ, Antoine P.,
Cyprien L., Jessica D.

Professeurs : S. Faulkner B. Frénay V. Salnikov.

2020-2021

Table des matières

Chapitre 1

Introduction

Le but de ce rapport est d'expliquer la démarche et la méthodologie qui a guidé l'élaboration des modèles de machine learning et de l'analyse des données fournie par les *opendata stib-mivb*. Ce rapport est constitué des différentes parties : l'analyse et la préparation des données, l'entraînement des modèles de régression et de classification et l'analyse de leurs résultats. Nous nous sommes concentré uniquement sur une ligne de tram mais le système pourrait facilement être étendu au reste du réseau.

L'étape d'analyse et la préparation des données met en lumière les notions de normalisation, la détecteur d'*outliers*, la selection de *features*. La visualisation des données est également une partie importante de l'analyse des données. En suite dans l'étape d'entraînement des modèles passe par un phase de selection des méta-paramètres et d'optimisation des prédictions.

1.1 Présentation du projet

Il nous a été demandé de développer un nouveau service ou une analyse pertinente par rapport au défis de la mobilité. Plusieurs opendata nous était proposées, nous avons décidé de choisir celle de la STIB. Nous avons choisi de mettre en place une interphase permettant de savoir si prochain tram qui arrivera à un stop que l'on attend aura du retard ou non.

Chapitre 2

Préparation et visualisation des données

2.1 Récolte des données

La première étape est de toute évidence la récolte des données. Notre projet nous demandais d'avoir accès à un historique de retard mais malheureusement cette historique ne fait pas partie des datasets des opendata STIB. Nous avons donc développé un script python [1] nous permettant de constituer cet historique de retard. Pour obtenir le délais, le script compare le temps d'arrivée théorique (qui nous est fournis par les fichiers GTFS [2]) et l'heure d'arrivée prévue (qui nous est fournie par l'api "*waiting time*"). Le délais est enregistré dans un fichier csv. En plus du délais, le script enregistre la température, la vitesse du vent, l'humidité et la visibilité grâce à l'api OpenWeather [3]. Un nouveau fichier csv est généré chaque jour.

Nous avons dans un premier temps récolté les données pour deux stop (les numéros 0089 et 6608G, voir leur emplacement dans l'annexe A.1) du premier novembre au douze novembre. Les fichiers csv sont disponibles sur le *repository* [4] du projet. Dans un second temps, nous avons récolté les données de tout les stops d'une ligne de tram (la ligne 39). La position de tout les stops de la ligne sont visible sur l'annexe A.2 [5]. Les données récoltées durant cette deuxième phase sont disponible sur le *repository* du projet [6]. Les analyses qui suivent se basent sur les données de cette deuxième récolte de données.

2.2 Premier coup d'œil à la structure des données

La commande `data.info()` indique qu'il y a 12798 lignes sans valeur pour la colonne `delay`. Pour chacune des *features* un graphique du nombre d'occurrences par valeur a été créé (voir l'annexe B.1). On peut voir sur ces graphiques que plusieurs *features* ont toujours la même valeur. On peut également voir que des retards on été enregistré pour d'autres lignes que la numéro 39, il faudra donc supprimer ces dernières.

Dans le graphique en annexe B.1 on remarque qu'il y a moins d'occurrences de valeur quatre pour la *features* `hour`. Cela est du au fonctionnement de l'API de la STIB. On peut également

1. Le code de ce script est disponible à l'adresse suivante : lien github du script
2. General Transit Feed Specification
3. Documentation disponible ici
4. Disponible ici
5. Une carte GoogleMyMaps est également disponible ici
6. Disponible ici

voir qu'il y a plus de d'occurrences pour les valeurs entre dix et quinze de la *features* `hour`, cette différentes est due à l'heure de démarrage et d'arrêt du script de récolte des données.

Le boxplot en annexe B.2 montre la répartition des retards du 19 septembre sur la ligne 39 au stop 0089. On remarque que la majorité des valeurs de délais se situe entre une minute d'avance et une minute de retard.

2.3 Préparation des données

Les lignes du dataset pour les quelles la colonne `delay` n'avait pas de valeur ont été supprimées. Les lignes dont la valeur de la colonne `line` n'était pas égale à 39 ont également été supprimées. Les features ayant toujours la même valeur ont été supprimées. Les colonnes `trip`, `theoretical_time`, `expectedArrivalTime` et `date` ont été supprimées car elles sont pas été jugées utiles. Les colonnes `theoretical_time`, `expectedArrivalTime` et `date` ont été supprimées car les valeurs étaient du type `string`. La colonne `date` a été considérée comme redondante, sa valeur étant déjà stockée dans les colonnes `hour`, `minute` et `day`.

2.3.1 Stratification des données

La stratification des données ne nous a pas semblé utile car le dataset n'est assez grand pour rendre cette dernière nécessaire. Cependant dans un but pédagogique nous avons quand même stratifié la colonne `hour`, afin que la répartition des différentes valeurs reste identique dans le dataset ainsi que dans le test-set.

2.4 Visualisation des données

2.4.1 Création du test-set

Cela pourrait paraître étrange de mettre de coté une partie des données à ce moment. Les données n'ont même pas encore été vraiment visualisée et nous devons encore en apprendre plus avant de choisir quel algorithmes utiliser. Cependant si le test-set est créé maintenant c'est pour éviter le *snooping bias*. Nous avons choisi de constitué le test-set de vingt pourcent des données du dataset.

2.4.2 Premières visualisations

La première visualisation générée (annexe B.3) est la répartition des délais en fonction de l'heure. On remarque qu'après dix-huit heure on a soit une avance ou un retard de vingt minutes ou une variation de cinq minutes par rapport à l'horaire théorique, sans valeur intermédiaire. On remarque également que la majorité des délais ont une valeur nulle. On peut également voir qu'il une augmentation des délais après quinze heure jusqu'a dix-neuf heure.

Le second graphique (annexe B.4) indique le délais moyen par heure. On y remarque une augmentation des délais entre six et neuf heure, à quinze heure et ainsi qu'a vingt-et-une heure. Le pick de retard de vingt-et-une heure vient sans doute du couvre feu de vingt-deux heure, les autres pick quand à eux sont à priori du au traffic de Bruxelles.

Sur la dernière infographie (annexe B.5) on remarque une hausse des températures de midi à seize heure.

2.4.3 Recherche de corrélation

Étant donné que notre dataset n'est pas trop grand nous pouvons facilement calculer le *coefficient standard de corrélation* entre chaque paire de *feature*. Comment on peut le voir avec le bout de code suivant (Voir listing 2.1), le retard est très peu linéairement corrélé avec les autres *features*.

```
>>> corr_matrix = data.corr()
>>> corr_matrix["delay"].sort_values(ascending=False) # warning: this check only
    linear correlation
delay          1.000000
rain           0.018600
temp           0.009013
wind           0.002024
hour          -0.000627
minute        -0.006874
humidity      -0.016347
visibility    -0.034723
day           -0.079071
Name: delay, dtype: float64
```

Listing 2.1 – Coefficient standard de corrélation pour la *feature* `delay`.

La *heatmap* (disponible en annexe C.1) nous permet de confirmer que la *feature* `delay` n'est pas linéairement corrélée avec les autres *features*. Elle nous fournis cependant des informations supplémentaires comme le fait que la *feature* `temp` est linéairement corrélée avec la *feature* `wind`, ainsi qu'inversément linéairement corrélée avec la *feature* `humidity`. Cependant ces corrélations reste faibles, comme on peut le voir sur la figure C.2. On voit bien que globalement quand la valeur de la *feature* `humidity` chute quand celle de la *feature* `temp` augmente mais on reste cependant loin d'une belle ligne droite.

2.4.4 Combinaison de *features*

La *feature* `hour_minute` est une nouvelle *feature* que nous avons créé en combinant les *features* `hour` et `minute`. Ce regroupement des *features* est fait dans un but pédagogique car de manière générale on a plutôt tendance à séparer les *features*. La séparation des *features* est une bonne façon de les rendre plus utilisables par l'algorithme de machine learning car la plus part du temps les datasets possèdes des colonnes du type `string` qui viole le principe de *tidy data*. La division des *features* permet améliorer les performances du modèle en découvrant des informations potentielles. Ce qui a déjà été fait en divisant la *feature* `date` en les *features* `year`, `month`, `day`, `hour` et `minute`.

Visualisation de la nouvelle *feature*

Le graphique en annexe C.3 nous montre la répartition des délais en fonction de l'heure de la journée. On remarque que le graphique garde évidement sa forme de fourche caractéristique.

En réaffichant les coefficient standard de corrélation en prenant en compte cette nouvelle *feature*, on remarque qu'elle est encore moins linéairement corrélée avec la *feature* `delay` que ne le sont les *features* `hour` et `minute`.

2.5 Les *features* de type `string`

La plus part des algorithmes de machine learning préfèrent travailler avec des nombres plutôt qu'avec du texte, c'est pourquoi nous convertissons la *feature* `stop` en différentes catégories représentées par un nombre.

2.6 Pipeline

Un pipeline a été créé pour faciliter l'exécution des étapes de transformation des données. Le pipeline effectue les opérations suivantes, premièrement amputer les lignes ayant au moins une valeur nulle dans l'une des colonnes. En suite vient l'étape d'ajout de la *feature* `hour_and_minute` qui peut être déactivée à la volée à l'aide d'un paramètre. Ensuite les données sont standardisées et l'étape finale est la création des catégories des stops comme vu dans la section précédente.

Chapitre 3

Modèles et résultats

Ce chapitre est divisés en deux partie la première concerne la partie de l'équipe "regression" (constituée de Jean A. et de Cyprien L.) et la seconde concerne la partie de l'équipe "classification" (constituée d'Antoine P. et de Jessica D.). Chacune des équipes est partie sur un vision différente du problème, comme leur nom l'indique le premiere équipe a vu le problème comme un problème de régression tandis que la second comme un problème de classification.

3.1 Partie 1 - Regression [1]

Différent modèles ont été entraînés afin de sélectionner le meilleur d'entre eux. Le premier a avoir été testé est le modèle de regression linéaire, les prédictions obtenues sont disponible dans le tableau en annexe D.1. Le modèle fonctionne mais n'est pas précis du tout. Pour nous rendre compte à quel point le modèle se trompe nous calculons la *root-mean-square error* (RMSE), cette dernière est égale à 2.47. C'est vraiment impressionnant, mais le modèle ne serait-il pas en,train d' *overfiter* les donnée. Lors que l'on utilise donc un modèle d'apprentissage par arbre de décision et que l'on calcule la RMSE on obtient une valeur nulle. Ce qui signifie que le modèle *overfit* également les données d'entraînement.

3.1.1 Cross-validation

Afin de mieux évaluer les modèles testés, nous avons utilisé une *10-fold cross-validation*. En utilisant la cross validation sur le modèle linéaire nous obtenons toujours une erreur moyenne de 4.8 pour les dix *folds*. Le modèle d'apprentissage par arbre de décision obtient lui une valeur de 5.78, pire que le modèle de régression linéaire donc. Ce qui indique que l'arbre de décision *overfit* tellement qu'il fonctionne moins bien que le modèle de régression linéaire. Apres avoir entraîné un modèle de type *Random Forest* et du type *Support Vector Regression* sur 10 *folds* nous obtenons respectivement une erreur moyenne de 4.57 et de 4.84. Pour la suite, nous avons choisi le modèle *Random Forest*.

3.1.2 *Grid search*

Nous avons donné un liste de valeur pour chaque métaparamètres (`n_estimators` , `max_features` et `bootstrap`) à tester. *Grid search* nous indique que la meilleur combinaison de valeur est `{'max_features': 8, 'n_estimators': 30}` avec une erreur moyenne de 4.47. Ce résultat peut encore

1. Le code concernant cette section est disponible ici

être amélioré car la valeur des métaparamètres `n_estimators` et `max_features` sélectionnées ont tous deux la valeur maximale que nous avions donnée. Nous changeons donc la liste des valeurs de `n_estimators` et `max_features` pour des valeurs plus grandes que 30 et 8 respectivement et ainsi de suite. La meilleur combinaison était `{'max_features': 9, 'n_estimators': 120}` avec une erreur moyenne de 4.42 (de légères améliorations restait cependant possible mais les gains en précisions était négligeables).

3.1.3 *Randomized Search*

Nous avons test une *Randomized Search* avec des valeurs entre 1 et 200 pour `max_features` et entre 1 et 8 `n_estimators`. La meilleur combinaison de métaparamètres est `{'max_features': 7, ' n_estimators': 180}` avec une erreur moyenne de 4.719.

3.1.4 Intervale de confiance

L'intervale de confiance (95%) pour la RMSE est entre 4.42min et 4.68min

3.2 Partie 2 - Classification

3.2.1 Adapter les données pour la classification

premièrement nous avons adapter les donnée pour la classification car celles que nous avons collectées étaient faite pour la régression. Par exemple la *feature* `delay` a du être modifiée, pour ce faire nous avons créé un script Python. Ce script crée deux fichier l'un avec les `delay` divisé en quatre catégories et le second ou ils sont divisés en seulement trois catégories. Dans le premier cas les catégories sont les suivantes *big late, late, on time, early et big early* et le deuxième cas les trois catégories sont *late, on time et early.* Le script supprime également les valeurs nulles. Pour choisir le meilleurs nombre de classe de `delay`, nous avons affiché le nombre d'occurrence de chacune des classes (voir annexe E.1 et annexe E.2). On voit qu'il y a moins de `delay` dans les classes *big late et big early* et la division des classes est meilleure dans le deuxième cas.

3.2.2 Preparation des données pour le machine learning

Maintenant ce choix fait, nous pouvons aller plus loin, comme l'équipe de régression nous division les données en deux : les assets et les targets. Nous avons ensuite nettoyé les assets pour ne garder que les *features* utile.

Supprimer les attributes avec toutes les valeurs identiques

Nous pouvons voir que `transport_type, year, month, direction` ont toujours la même valeur pour chaque ligne. Les attributes de type string ont également été supprimés

Transformation des attributs catégoriels

Nous transformons les attributes `string` en catégorie afin d'aider l'algorithme de machine learning.

Extra feature

Nous avons également ajouté une colonne supplémentaires qui est la combinaison des colonnes `hour` et `minute`

3.2.3 Selection et entraînement des modèles

Maintenant que les données sont préparées nous pouvons commencer à choisir et entraîner un modèle. Nous avons pour se faire utilisé différent outils comme la cross-validation ou encore le grid search.

Decision tree

Le premier modèle que nous avons testé est l'arbre de décision. Nous avons divisé les données en deux datasets le premier est le test set et le train set. Nous avons choisi le critère d'entropie et le `min_sample_leaf = 100` comme hyper paramètres.

Ce qui nous donne un score d'entraînement de 54.4% et un score de test de 52.2% avec un 10-folds validation on a une moyenne de 52.4% and une deviation standard de 0.7%.

Random forest

Nous avons testé 2 random forest :

- RandomForestClassifier(n_estimators=100, criterion="entropy", bootstrap=True, min_samples_leaf=300 class_weight=class_weight)
- RandomForestClassifier(n_estimators=100, criterion="entropy", bootstrap=True, min_samples_leaf=100)

Le premier nous donne un score d'entraînement de 51.5% et un score test de 50.6% nous avons également fait une 10-fold cross validation qui nous donne une moyenne de 50.6% et une deviation standard de 0.9%

Le second nous donne un score d'entraînement de 54.0% et un score test de 52.4%, le 10-fold cross validation nous donne une moyenne de 52.4% et une deviation standard de 1.0%

KNN

Nous avons choisi un KNN avec 50 voisins et un `leaf_size` de 300. Ce qui nous donne un score d'entraînement de 55.9% et un score test de 52.6% avec la 10-fold cross validation on a une moyenne de 53.1% et une deviation standard de 0.8%.

Grid Search pour le random forest

Nous avons testé une combinaison de 72 hyperparameters

```
bootstrap= [False,True], n\_estimators = [5,10,100], max\_features = [2, 3, 4,8], min\_samples\_leaf
 = [100,200,500]
```

Le grid search nous indique que la meilleur combinaison est bootstrap = False, max_features= 8, min_samples_leaf =100, n_estimators =100.
Ce qui nous donne un score d'entraînement de 56.3% et un score test de 54.1%.

Si nous regardons de plus près les scores, nous pouvons voir que c'est toujours avec `min_sample_leaf = 100` que nous avons les meilleurs résultats donc nous avons testé des valeurs moins grande.

64 combinaisons d'hyperparameters supplémentaires on donc été testées.
bootstrap =[False,True], max_features= [2, 3, 4,8], criterion = ["entropy","gini"], min_samples_leaf = [10,30,50,100]
Selon le grid search les meilleurs paramètres sont : bootstrap = False, criterion = 'entropy', max_features = 8, min_samples_leaf = 10
Avec un score d'entraînement de 71.5% et un score test de 58.3%.

On voit que si on choisit 10 pour `min_sample_leaf`, on commence à avoir un petit overfit, selon nous, il faut choisir entre 30 et 50, 30 semblant être le meilleurs. on veut aussi voir que l'indice de gini est le meilleur on peux le supprimer de la grille de recherche.

Nous avons donc décidé de réduire le grid search à 6 combinaison d'hyperparameters.
bootstrap =[False,True], max_features= [4,8,"auto"]
La meilleur combinaison étant bootstrap = False and max_features = 8

Modèle Final

Notre modèle final est donc un random forest avec comme hyperparameters `n_estimator=10, min_sample_leaf=30, bootstrap=False, Max_features=8` Ce qui nous donne un score d'entraînement de 62.3% et un score test de 56.3%.
Nous savons que ce n'est pas le meilleur résultat mais ce problème est sûrement lié au manque de données, en ajouter des donnés sur l'état du réseau, les accidents, les travaux, ...

Chapitre 4

Conclusion

Les predictions sont malheureusement loin d'être parfaites nous pensons que cela est du au manque de données non n'avons en effet effectué la collecte des données sur dix jours. Les données ne contiennent pas d'informations sur l'état du réseau ce qui pourrait aider l'algorithme de machine learning. Nous pouvons cependant retenir différentes informations grâce aux graphiques que nous avons généré. Premièrement on remarque qu'il y a réellement très peu de retard, que le délais apres 19 heure si il y a un retard, il est toujours soit de -10 minutes ou de 10 minutes. On voit également qu'il y a une augmentation des retards aux heures de pointe, il y a également une augmentation à 21 heure à causse du confinement. En remarque que la visibilité est linéairement corrélée avec le retard. Sur le graphique disponible en annexe F.2, on voit bien qu'il y a moins de retard le dimanche mais qu'il y plus de problème le samedi. On remarque également sur le graphique des retards par stops (voir annexe F.3) que les stops n°5512 et n°6474F ont des retards plus important. Quand a notre algorithme de machine learning il permet de prévoir des retards mais avec une précisions relativement faible, il pourrait cependant être amélioré en utilisant des algorithmes plus puissant, mais surtout en améliorant la qualité et la quantité des données. Nos avons conclu que la classification, une fois améliorée, permettrait d'obtenir les informations les plus utiles et les plus précises. Ces informations permettrais de fournir à la STIB un outil pour prédire la vulnérabilité d'un stop du réseau à l'avance, des opérations préventives pourrait donc être mises en place pour réduire le retard.

Annexe A

Position des stops

FIGURE A.1 – Position des stops numéro 0089 et 6608G en jaune et noir respectivement.

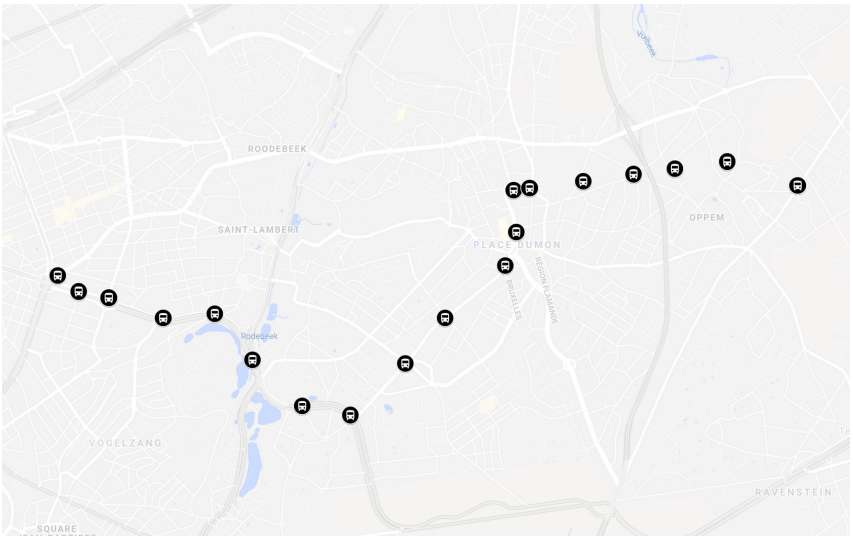

FIGURE A.2 – Position des stops présent sur la ligne 39

Annexe B

Première visualisation des données

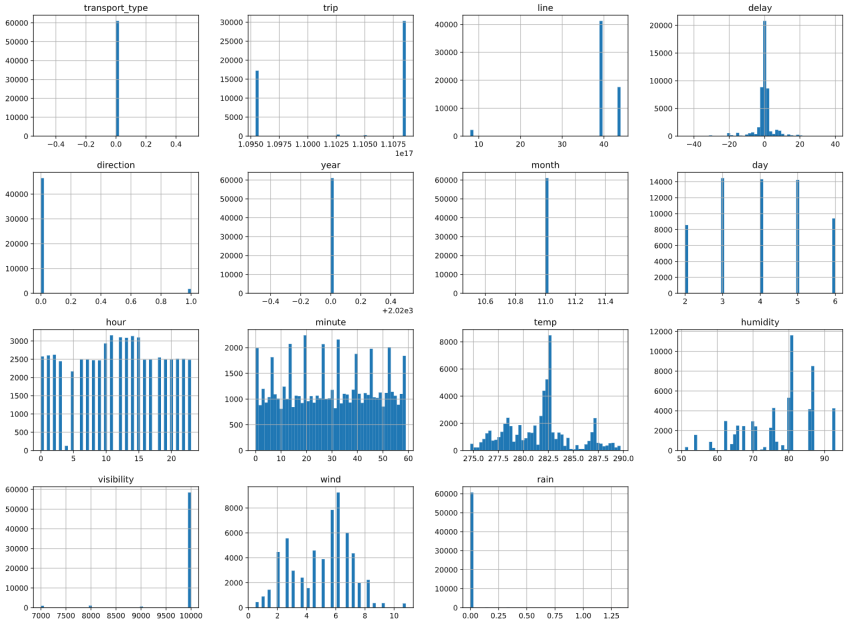

FIGURE B.1 – Plot des features par nombre d'occurrences

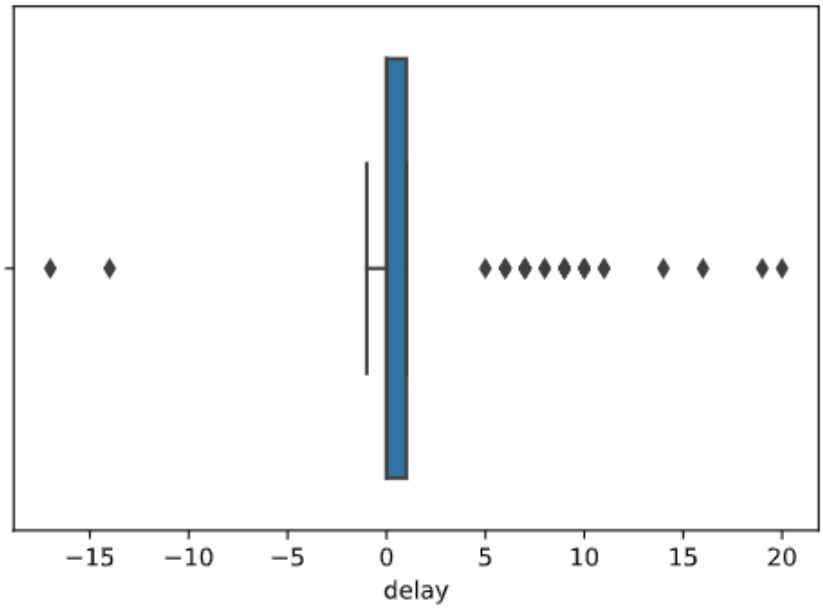

FIGURE B.2 – Boxplot des délais du 19 septembre sur la ligne 39 au stop 0089

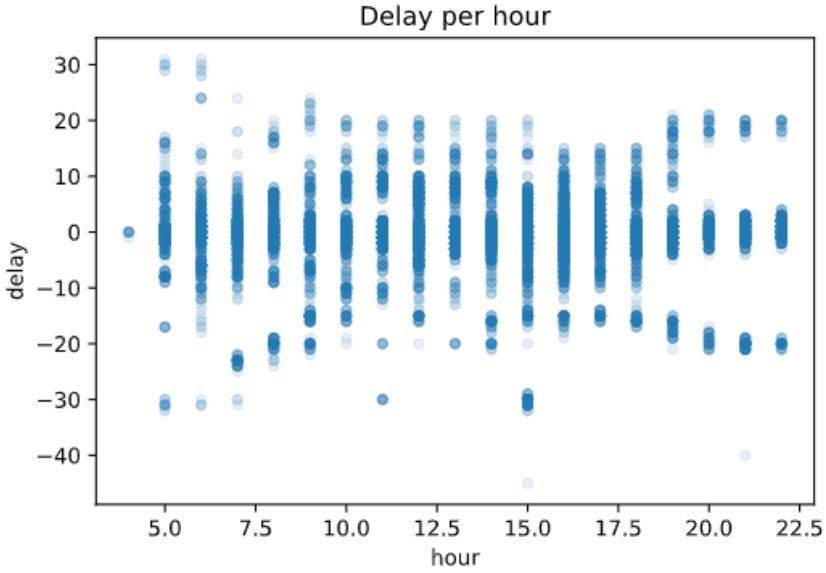

FIGURE B.3 – Délai par heure pour tout le dataset

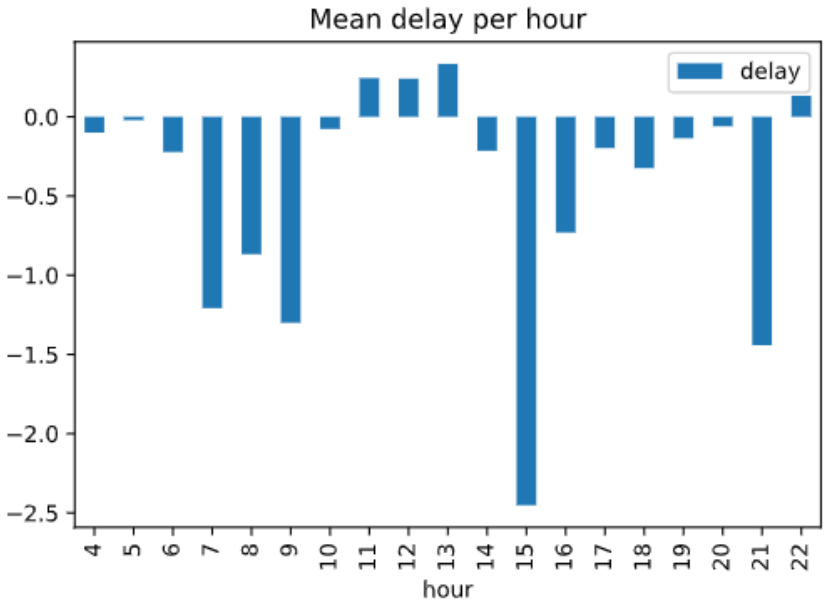

FIGURE B.4 – Délai moyen par heure pour tout le dataset

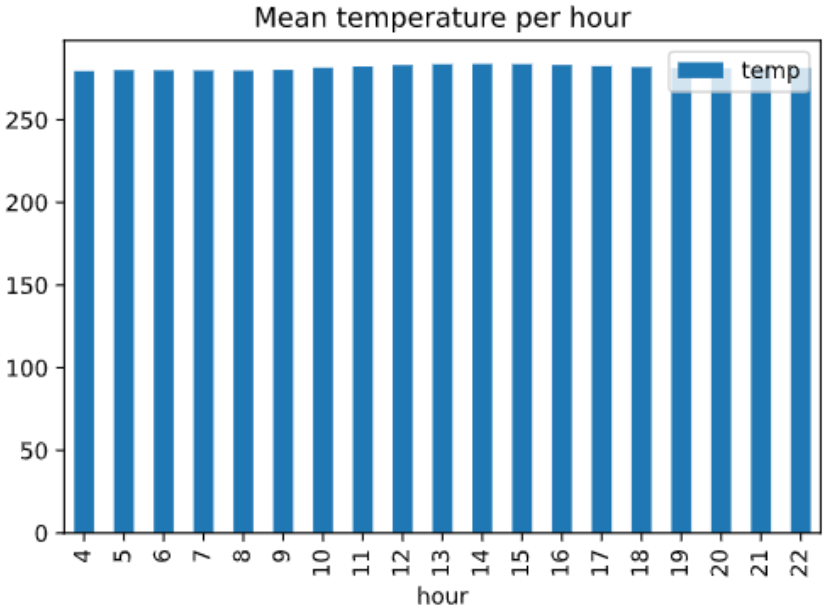

FIGURE B.5 – Température moyenne par heure pour tout le dataset

Annexe C

Recherche de corrélation

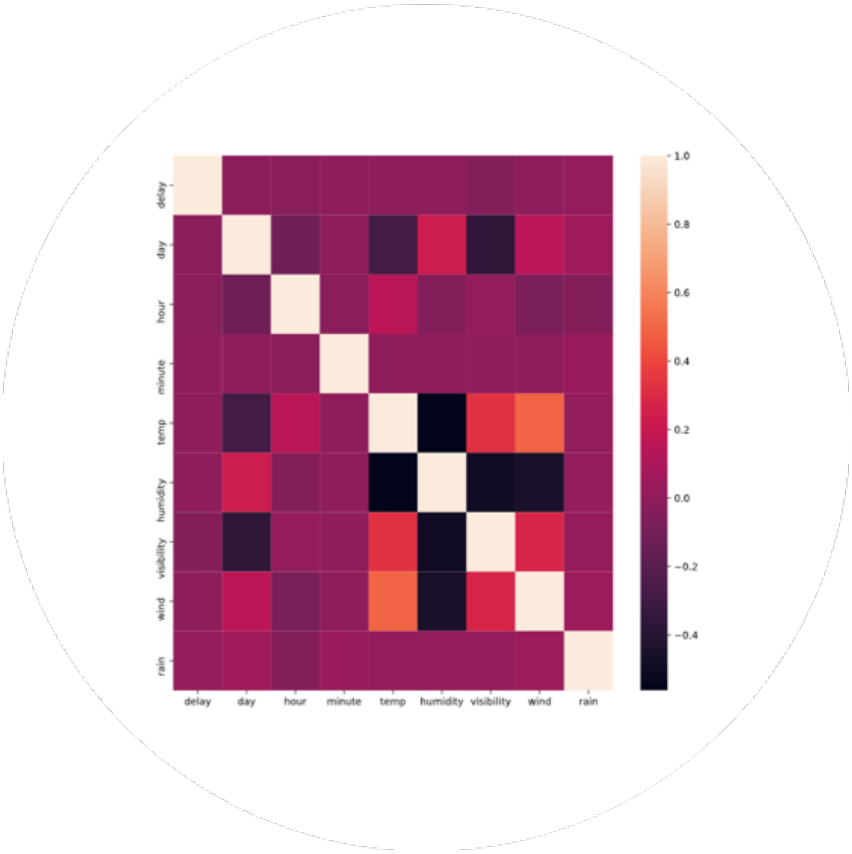

FIGURE C.1 – Matrice de corrélation linéaire entre les différentes *features*

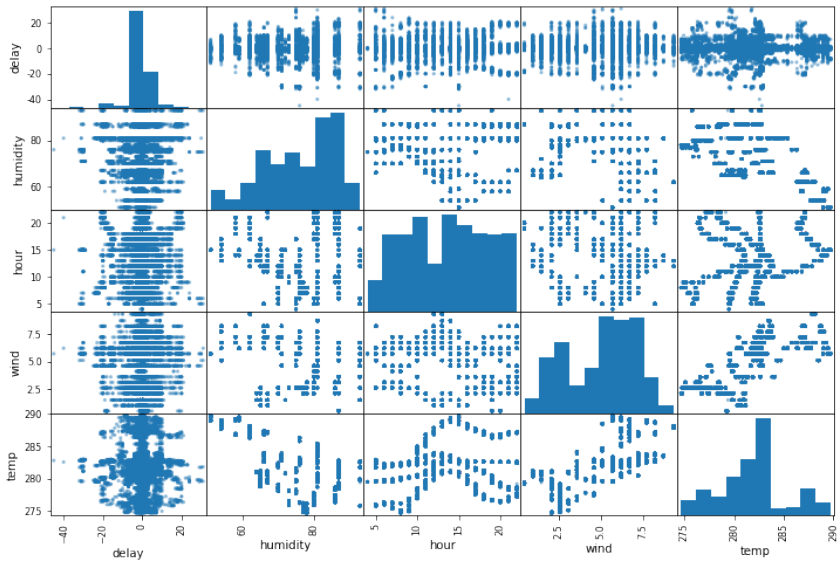

FIGURE C.2 – Plot de chaque *features* par rapport à chacune des autres

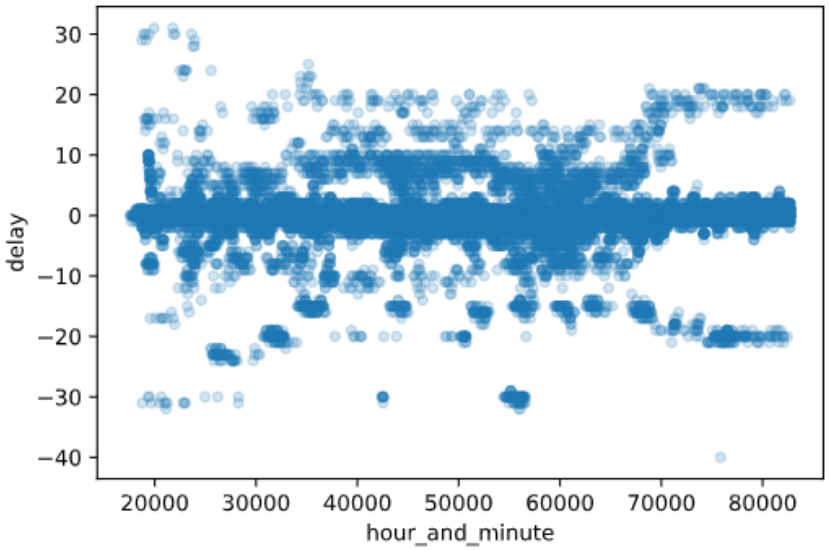

FIGURE C.3 – Délai par heure et minute pour tout le dataset

Annexe D

Regression

Valeur réelle	**Prédiction**	**Différence**
-1.0	1.02270508	-2,023
-1.0	-1.04394531	0,044
0.0	-1.015625	-1.016
-1.0	0.03173828	-1,032
-18.0	0.48291016	-18,483

TABLE D.1 – Échantillon de prédiction du modèle de régression linéaire

Annexe E

Classification

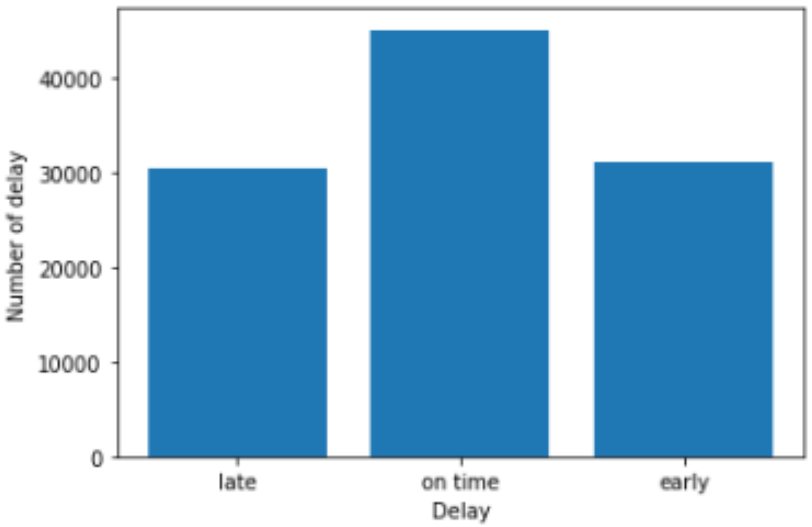

FIGURE E.1 – Division des délais en 3 catégories

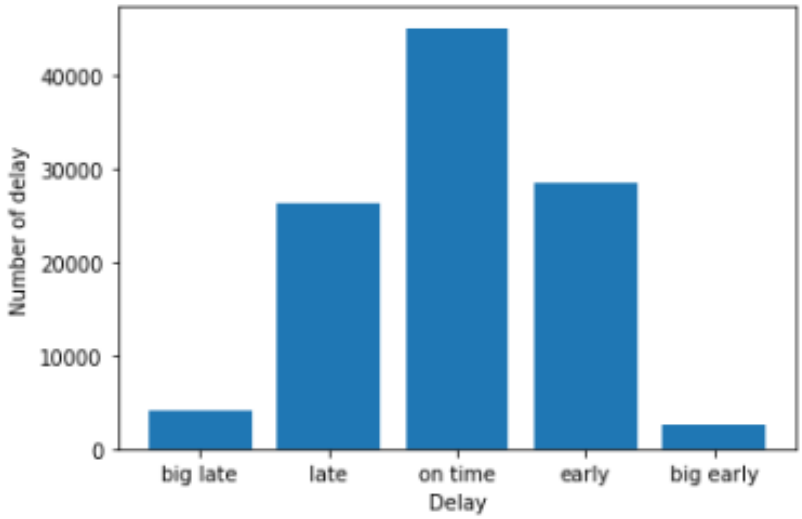

FIGURE E.2 – Division des délais en 5 catégories

Annexe F

Conclusion

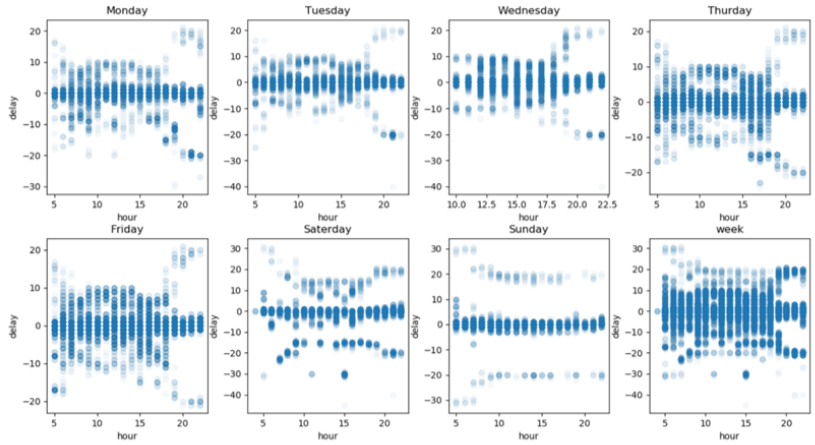

FIGURE F.1 – Le retard en fonction de l'heure par jours

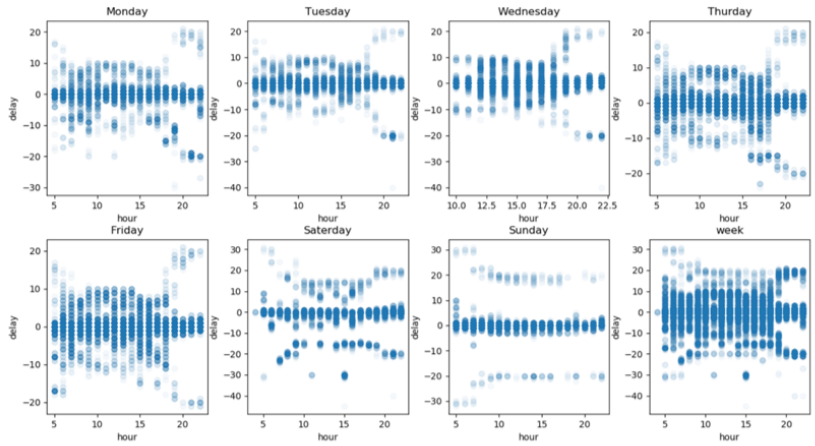

FIGURE F.2 – Le retard en fonction de l'heure par jours

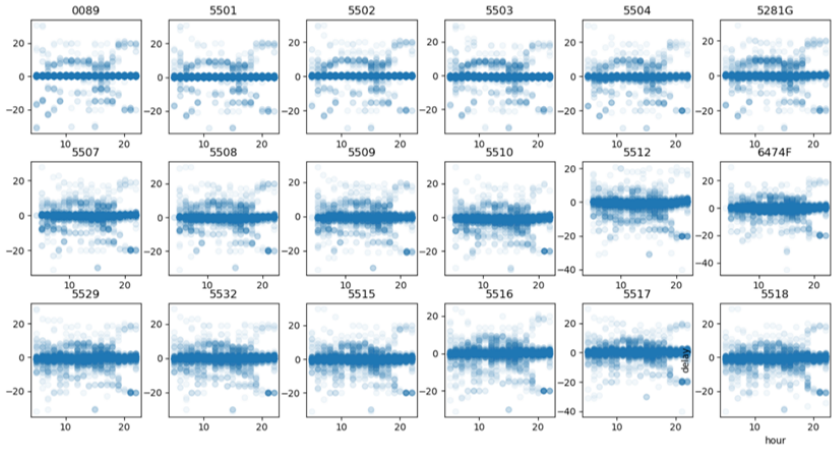

FIGURE F.3 – Le retard en fonction de l'heure par stop

Annexe G

Bonus

Figure G.1 – Bonus

Listings

Bibliographie

[1] Robert R. F. DeFilippi. Standardize or Normalize Examples in Python. Avr. 2018. URL : https://medium.com/@rrfd/standardize-or-normalize-examples-in-python-e3f174b65dfc.

[2] Aurélien Géron. Hands-on machine learning. Avr. 2017. URL : http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20%5C&path=ASIN/1491962291.

[3] Urvashi Jaitley. Why Data Normalization is necessary for Machine Learning models. Avr. 2019. URL : https://medium.com/@urvashilluniya/why-data-normalization-is-necessary-for-machine-learning-models-681b65a05029.

[4] machine learning in Python scikit-learn documentation. URL : https://scikit-learn.org/stable/.

[5] Python Data Analysis Library. URL : https://pandas.pydata.org/.

[6] Emre Rençberoğlu. Fundamental Techniques of Feature Engineering for Machine Learning. Avr. 2019. URL : https://towardsdatascience.com/feature-engineering-for-machine-learning-3a5e293a5114.

[7] Natasha Sharma. Ways to Detect and Remove the Outliers. Mai 2018. URL : https://towardsdatascience.com/ways-to-detect-and-remove-the-outliers-404d16608dba.

[8] statistical data visualization seaborn documentation. URL : https://seaborn.pydata.org/.

[9] Hadley Wickham. « Tidy data ». In : The Journal of Statistical Software 59 (10 2014). URL : http://www.jstatsoft.org/v59/i10/.

[10] Zixuan Zhang. Understand Data Normalization in Machine Learning. Août 2019. URL : https://towardsdatascience.com/understand-data-normalization-in-machine-learning-8ff3062101f0.