Prédiction des temps de trajet dans un réseau de bus à l'aide de données historiques

Nikita Marchant[1]
Superviseurs : Martine Labbé, Samuel Deleplanque

[1]Université Libre de Bruxelles, Département d'Informatique
nimarcha@ulb.ac.be

Abstract

Introduction

Le but de ce projet est de pouvoir prédire, en temps réel, l'heure d'arrivée d'un véhicule de transport en commun aux arrêts de son trajet. Dans le cadre de ce travail, la prédiction sera effectuée pour des lignes de bus, tram et métro de la STIB [1].

Un réseau de transport en commun étant très difficile à modéliser à cause de sa complexité et par ce qu'il est très influençable par des événements stochastiques, l'approche qui sera utilisée ici sera non pas de modéliser le réseau pour prédire son état futur mais d'extrapoler grâce à des algorithmes de machine learning les trajets des véhicules grâce à des données historiques récoltées au préalable.

Méthode développée

La méthode qui a été développée dans le cadre de ce travail est la méthode des k plus proches voisins.

L'implémentation a été découpées en n grandes parties : premièrement, la collection de données pour alimenter l'apprentissage du modèle, ensuite le traitement de ces données pour améliorer leur qualité et les transformer en un format utilisable par le modèle, l'étape suivante a été l'analyse et le choix de fonctions de score pour entraîner le modèle et pour finir, l'entraînement en lui même ainsi que l'analyse des résultats.

Collection des données

La STIB ne mettant pas à disposition ses données historiques, il a fallu commencer par récolter des informations grâce à un programme écrit pour l'occasion. Les deux seules sources de données sur l'état du réseau en temps réel disponibles sur internet sont : premièrement, pour chaque ligne, la position précise à un arrêt prêt de chaque véhicule et deuxièmement, pour chaque arrêt, une estimation du temps d'attente avant le prochain véhicule de chaque ligne.

La source de données qui a été choisie est la première pour deux raisons : le réseau de la STIB disposant de milliers d'arrêts, il aurait été difficile de récolter avec une fréquence suffisante les temps d'attente pour chaque arrêt. De plus, les temps d'attentes étant eux mêmes des prédictions, il serait un peu illogique de les utiliser comme des mesures.

La source choisie souffre tout de même d'un problème non-négligeable : les véhicules apparaissent sur la ligne comme de simples point non identifiés. Il n'est donc pas possible de récupérer le numéro du véhicule, sa plaque d'immatriculation ou tout autre identifiant unique. De plus, si deux véhicules d'une même ligne sont suffisamment proches, ils n'apparaissent plus que comme un seul point.

Le processus de collection des données est un programme *Python3* utilisant `aiohttp` pour permettre des requêtes http asynchrones au serveur web de la STIB et ainsi permettre une période d'échantillonnage de 20 secondes pour chaque ligne, dans chaque sens (ce qui fait 142 mesures toutes les 20 secondes dans l'état actuel du réseau). La période de 20 secondes a été choisie empiriquement car une fréquence plus élevée imposait une charge trop lourde au serveur de le STIB et une fréquence plus faible aurait été trop imprécise.

Cette collection de données a permis l'enregistrement d'approximativement 80 millions de mesures en 6 mois,réparties sur l'ensemble des lignes.

Traitement des données

Les données récupérées grâce à la méthode décrite dans la section précédente sont malheureusement de mauvaise qualité : il est courant qu'un véhicule cesse d'émettre sa position pendant plusieurs minutes, que la source de donnée tombe en panne ou que les positions de deux véhicules fort proches soient confondues en un seul point.

De plus, les mesures ne sont pas directement utilisables

1. Société de transports en commun à Bruxelles (Belgique)

par l'algorithme que nous avons choisi. En effet, nous ne disposons que d'une suite de positions (les sorties de notre programme de mesure sont des vecteurs horodatés de booléens, voir annexes) et ce dont nous avons besoin est des durées que chaque véhicule a pris pour parcourir les distances entre deux arrêts consécutifs.

Nous avons donc développé un algorithme qui assigne un identifiant à chaque nouveau point qui apparaît sur une ligne et qui essaye, dans la mesure du possible, de détecter lors des mesures suivantes si un point est déjà connu et qu'il s'est déplacé ou si c'est un nouveau point (par exemple un bus qui vient du dépôt).

Grâce à ces identifiants, nous pouvons donc maintenant reconstituer des trajets (une suite de paires [position, instant]) pour ces véhicules en parcourant les mesures et en enregistrant la position de chaque identifiant dans la ligne à chaque instant.

Dans l'optique de pouvoir proposer des prédictions en temps réel, l'algorithme supporte un mode dans lequel il a connaissance des trajets passés dont le véhicule n'est pas arrivé au terminus et qui permet de compléter ces trajets au fur et à mesure que de nouvelles mesures sont disponibles.

L'algorithme, bien qu'ayant pris du temps à développer et à paramétrer ne présente pas de grand intérêt scientifique. Son fonctionnement est donc décrit en annexe.

Modèle d'apprentissage : les k plus proches voisins

L'algorithme des k plus proches voisins [2] est une méthode d'intelligence artificielle qui peut être utilisée aussi bien pour de la classification que pour de la régression [Hastie et al. (2009)]. L'idée de cette méthode est de trouver les k trajets les plus similaires à la cible et d'utiliser ceux-ci pour extrapoler le temps de trajet futur.

Pour calculer la similarité entre deux trajets, nous devons extraire des *features*, des grandeurs qui caractérisent ces trajets. Les features que nous utiliserons ici sont les temps que le véhicule a pris pour voyager entre deux arrêts. Un trajet n'est évidemment pas caractérisé uniquement par ces grandeurs mais nous verrons qu'en première approximation ces grandeurs sont déjà suffisantes [3].

Pour cela, nous projetons chaque trajet dans un espace à n dimensions avec $n+1$ étant le nombre d'arrêts déjà effectués par le véhicule dont on cherche à prédire le temps de trajet (ce véhicule sera appelé α).

Les trajets sont donc représentés par le vecteur colonne $T_\alpha = \left(d_{1,\beta}, d_{2,\beta}, ..., d_{n,\beta}\right)^T$ avec $d_{i,\beta}$ étant la durée du trajet entre l'arrêt i et $i+1$ pour le véhicule β.

Maintenant que nous avons transformé nos trajets en vecteurs, nous pouvons utiliser tous les trajets du passés (l'ensemble d'apprentissage) et les placer dans cet espace à n dimensions pour entraîner notre algorithme.

Une fois que l'algorithme est entraîné, nous pouvons faire une prédiction pour un nouveau trajet ($\hat{d}$) dont on ne connaît pas la fin. Pour cela, nous le plaçons aussi dans cet espace et nous mesurons sa similarité par rapport à tous les autres trajets de l'ensemble d'apprentissage.

La similarité $s_{\alpha,\beta}$ (ou distance) entre deux trajets α et β est définie comme la distance euclidienne entre deux vecteurs :

$$s_{\alpha,\beta} = \sqrt{\sum_{j=0}^{n}(d_{j,\alpha} - d_{j,\beta})^2} \tag{1}$$

Nous sélectionnons ensuite les k vecteurs dont la similarité est la plus proche de $\hat{d}$ et nous nommons cet ensemble $\mathcal{V}$: les plus proches voisins.

Dans la version originale de kNN, la prédiction du temps de trajet $\hat{d}_{n+1,\alpha}$ est ensuite donnée par la moyenne des temps de trajets pour les vecteur appartenant à $\mathcal{V}$.

$$\hat{d}_{n+1,\alpha} = \frac{1}{k}\sum_{i\in\mathcal{V}} d_{n+1,i} \tag{2}$$

Il est aussi possible d'effectuer la régression à l'aide d'une moyenne pondérée. Chaque vecteur est pondéré par l'inverse de sa distance par rapport au vecteur cible :

$$\hat{d}_{n+1,\alpha} = \sum_{i\in\mathcal{V}} s_{\alpha,i} \cdot \frac{1}{k}\sum_{i\in\mathcal{V}}\left(\frac{d_{n+1,i}}{s_{\alpha,i}}\right) \tag{3}$$

La projection des trajets dans un espace à n dimension ainsi que l'implémentation de l'algorithme des plus proches voisin ont dans un premier temps été écrits en *Python3*, uniquement à l'aide de la librairie `numpy` pour faciliter les calculs vectoriels et un prototypage rapide. Dans un second temps, nous avons utilisé la librairie `scikit-learn` pour profiter de son implémentation optimisée de kNN.

Fonctions de score

Afin d'effectuer un bon choix de l'hyper-paramètre k ainsi que de choisir entre la moyenne pondérée ou non, il est important de pouvoir comparer deux modèles entre eux.

2. k-nearest neighbors ou encore k-NN.

3. Une autre feature très intéressante et fort utilisée dans la littérature est la taux de remplissage du véhicule mais nous n'avons pas accès à cette donnée dans notre cas. D'autres features qui pourraient être intéressantes sont énumérées dans les perspectives, en fin d'article

Pour cela, nous aurons aussi besoin d'assigner un score à une prédiction en fonction de sa performance par rapport à ce qu'il c'est vraiment passé.

Score d'une prédiction

Pour assigner un score à une prédiction, la littérature utilise souvent le RMSE [4] : l'erreur quadratique moyenne. Cependant, cette métrique souffre d'un problème : qu'un bus soit annoncé une minute à l'avance ou une minute en retard est pondéré de la même manière alors que dans un cas l'usager attend son bus une minute de plus et que dans l'autre il le rate.

Nous avons donc implémenté d'autres métriques qu'il nous semblait intéressant de pouvoir minimiser :

— Le RMSE double négatif : chaque erreur négative (quand un bus est annoncé plus tard qu'en vrai) est doublée et en suite on applique le RMSE.
— RMSE pondéré linéairement : l'erreur de prédiction pour l'arrêt suivant est pondéré avec un poids de 5, l'erreur pour le dernier arrêt est pondérée avec un poids de 1 et les arrêts entre les deux sont pondérés linéairement entre 1 et 5.
— RMSE pondéré exponentiellement : l'erreur de prédiction pour le $n$ème arrêt est pondéré avec un poids de $\frac{1}{n}$

La première métrique résout (ou en tout cas diminue) le problème du bus raté et les deux suivantes prennent en compte le fait qu'il est plus intéressant de bien prédire le futur proche et un peu moins bien le futur lointain que de bien prédire l'entièreté du trajet.

Avant de choisir la métrique que nous utiliserons comme fonction de score, nous avons décidé de les comparer. Pour cela, nous entraîné un kNN avec un k fixé arbitrairement à 130 à l'aide de 12000 trajets de la ligne de bus 95 vers Grand-Place en utilisant les 10 premiers arrêts comme donnée connue et les 13 suivants comme donnée à prédire. Ensuite, nous avons effectué 6000 prédictions pour d'autres trajets de la même ligne et nous avons calculé leur score avec chaque métrique.

Nous avons ensuite comparé graphiquement deux à deux chacune des fonctions de score sur l'ensemble des 6000 prédictions. Nous pouvons en voir un exemple dans la figure 1 : chaque point représente une prédiction avec, en l'abscisse son score selon le RMSE et en ordonnée son score selon le RMSE double négatif. Des graphes pour toutes les autres comparaisons se trouvent en annexe.

Cette première comparaison nous a déjà permis d'apercevoir une corrélation linéaire entre les différentes fonctions. En plus de cette corrélation déterminée visuellement,

4. Root Mean Squared Error

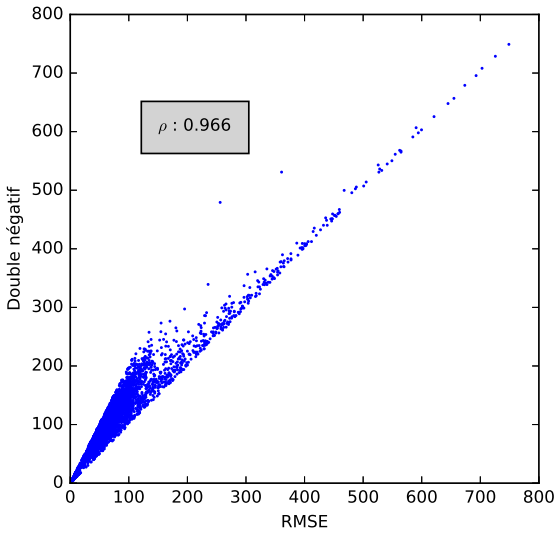

FIGURE 1 – Comparaison entre le RMSE et le double négatif pour chacune des 6000 prédictions.

nous avons comparé la corrélation de rang entre les fonctions grâce au coefficient de corrélation de Spearman (ou ρ). Celui-ci nous donne un indice sur le fait que deux fonctions ordonnent un ensemble de la même manière [5]. Celui-ci varie entre -1, quand l'ordre est parfaitement inversé à 1, quand l'ordre est identique.

Sa formule est définie comme ceci, d étant la différence entre le rang défini par la première fonction et celui défini par la seconde :

$$\rho = 1 - \frac{6 \sum_{i=1}^{n} d_i^2}{n^3 - n} \tag{4}$$

	RMSE	linéaire	exponentiel	double
RMSE	1.000	0.961	0.812	0.966
linéaire	0.961	1.000	0.896	0.926
exponentiel	0.812	0.896	1.000	0.793
double	0.966	0.926	0.793	1.000

TABLE 1 – Rho de Spearman entre chaque chaque paire de métrique de score.

Le RMSE double négatif semblant être la métrique la plus intéressante à minimiser et le RMSE simple étant quasiment

5. En effet, il nous importe peu que la corrélation soit linéaire ou même quadratique : tant qu'on peut classer les prédiction de la meilleure à la moins bonne et que ce classement ne change pas lors d'un changement de fonction de score, cela est suffisant pour nous.

identique (voir table 1 et figure 1), nous avons choisi d'utiliser par la suite le RMSE pour sa simplicité d'implémentation et sa rapidité (ce n'est que la norme du vecteur d'erreur, fonction qui est déjà implémentée rapidement dans `numpy`)

Score d'une modèle

References

Hastie, T. J., Tibshirani, R. J., and Friedman, J. H. (2009). *The elements of statistical learning : data mining, inference, and prediction*. Springer.

Appendices

Algorithmes de traitement et de collection de données

Source de données

Notre source primaire de donnés se présente comme ceci :

Chaque point rouge représentant au minimum un véhicule (mais parfois plus dans le cas ou ils seraient trop proches les uns des autres). La mesure correspondant à cette copie d'écran serait ceci (un vecteur de booléens) : `[vrai, faux, faux, faux, vrai, faux, faux, vrai, faux, faux]`

Problèmes rencontrés

Description du fonctionnement