Amazon Last-Mile Routing Research Project

Table des matières

Table des figures

Introduction

Le Amazon Last Mile Routing Research Challenge [1] est hébergé par Amazon.com et soutenu scientifiquement par une équipe de chercheurs du MIT Center for Transportation & Logistics. L'objectif principal du défi est de développer des approches innovantes tirant parti de l'intelligence artificielle, de l'apprentissage automatique, de l'apprentissage en profondeur, de la vision par ordinateur et d'autres méthodes non conventionnelles pour produire des solutions au problème de séquençage d'itinéraire qui surpassent l'optimisation traditionnelle.

En effet, malgré d'énormes progrès dans l'optimisation des itinéraires au cours des dernières décennies, il subsiste un écart important entre la planification d'itinéraire théorique et l'exécution réelle des itinéraires : dans les opérations réelles, la qualité d'un itinéraire n'est pas exclusivement définie par sa longueur théorique, sa durée ou son coût. Les chauffeurs-livreurs expérimentés ont une connaissance tacite de l'environnement opérationnel complexe dans lequel ils servent les clients au quotidien. Pour permettre une livraison plus sûre, plus efficace et plus durable, il est essentiel de tirer parti de ces informations tacites pour améliorer la planification des itinéraires.

1. `https://routingchallenge.mit.edu`

Chapitre 1

Définition du cadre du projet

1.1 Présentation du problème et des données fournies par Amazon

Les organisateurs du projets ont fournis des données (au format `json`) contenant des données historiques de 6112 routes empruntées par des livreurs en Amérique du Nord entre le 19 juillet et le 26 août 2018 (39 jours). Les livraisons se concentrent dans les alentours de 5 grandes villes : Boston, Los Angeles, Seatle, Chicago et Austin. Les données sont décrites dans le tableau A.2 et celles-ci sont réparties en trois ensembles :

- Informations au niveau de l'expédition : lieux de livraison et caractéristiques des colis à livrer ainsi que les constraintes pesant sur les modalités de livraisons.
- Informations sur le transit : distances et temps de déplacement entre les lieux à desservir.
- Informations au niveau de l'itinéraire : date de l'itinéraire, origine de l'itinéraire, caractéristiques du véhicule et liste des différents lieux à visiter.

Notre objectif est de pouvoir générer des séquences d'itinéraires qui prédisent de "bonnes séquences" sur un ensemble donné en apprenant à partir de séquences d'itinéraires réelles qui ont été exploitées par des conducteurs expérimentés réels. Le but est d'élaborer un modèle capable de prédire 1000 séquences en moins de deux heures. De plus, le modèle sera évalué à l'aide d'une métrique présentée en annexe E. Plus ce score est faible, plus la qualité de la solution est optimale.

1.2 Présentation du modèle préliminaire

1.2.1 Modélisation du problème

Le but du problème est de déterminer un plus court circuit qui visite chaque lieux de livraisons (les stops) une et une seule fois. Après avoir réalisé des recherches sur le vehicle routing problem (VRP), nous avons décidé de modéliser notre problème à l'aide du TSPTW (vehicle routing problem with time windows) dont la formulation mathèmatique est décrite en Annexe B. Nous nous sommes en particulier appuyé sur les articles [Pisinger, 2007] et [Mourid St-Pierre, 2012].

1.2.2 Implémentation

Afin de mettre en oeuvre ce programme linéaire, nous avons utilisé une bibliothéque python intitulé **PuLP** (cf [Mitchell et al., 2011]) qui permet de décrire des programmes mathématiques. Python est un langage de programmation de haut niveau bien établi et pris en charge qui met l'accent sur le développement rapide, la clarté du code

et de la syntaxe. PuLP fonctionne entièrement dans la syntaxe et les idiomes naturels du langage Python en fournissant des objets Python qui représentent des problèmes d'optimisation et des variables de décision, et en permettant aux contraintes d'être exprimées d'une manière très similaire à l'expression mathématique originale.

La librairie Pulp a l'avantage de pouvoir faire appel à different Solver. Nous avons décidé d'utiliser l'outil **"GNU Linear Programming Kit"** (voir [Romain, 2008]) qui est performant pour résoudre des problèmes d'optimisation linéaire de variables continues ou mixtes (entières et continues). Ce kit est composé d'un langage de modélisation GNU MathProg et d'une librairie de fonctions C (GLPK) utilisant le solveur Glpsol. L'extrême avantage de ce kit est d'être en libre accès et relativement facile à installer et à utiliser.

1.2.3 Étude du temps de calcul

De plus, afin d'accélérer la recherche d'un itinéraire, nous avons développé un modèle de Machine Learning (ML) présenté en annexe C. Son objectif est de déterminer quelles sont les itinéraires entre deux stops qui ont la probabilité la moins élevé d'être emprunté afin de pouvoir les retirer du problème et ainsi diminier le nombre de possiblilités.

Afin de savoir si ce programme linéaire est capable de prédire des séquences en un temps raisonnable, nous avons passé en paramètre de celui-ci des problèmes avec des nombres variables de stops afin de connaitre l'influcence de la taille du problème sur le temps de calcul. Or, comme nous supprimons certains itinéraires, le nombre de stops n'est pas suffisant pour caractériser la taille du problème. Pour cela, nous regardons également le nombre d'itinéraire reliant deux stops effectivement empruntables.

Les graphiques montrant les résultats de cette étude sont présentés sur le figure F.1. Nous pouvons observer une explosion combinatoire qui survient au alentour de 15 stops. Ainsi, pour pouvoir satisfaire aux exigences du projet, nous avons décidé de limiter à 15 le nombre de stops à mettre en paramètre du programme linéaire. Or, les séquences que nous devons prédire contiennent entre 100 et 200 points de livraisons. Ainsi, il a fallu trouver des solutions afin de diminuer le problème en sous-problèmes.

1.2.4 Présentation de l'algorithme

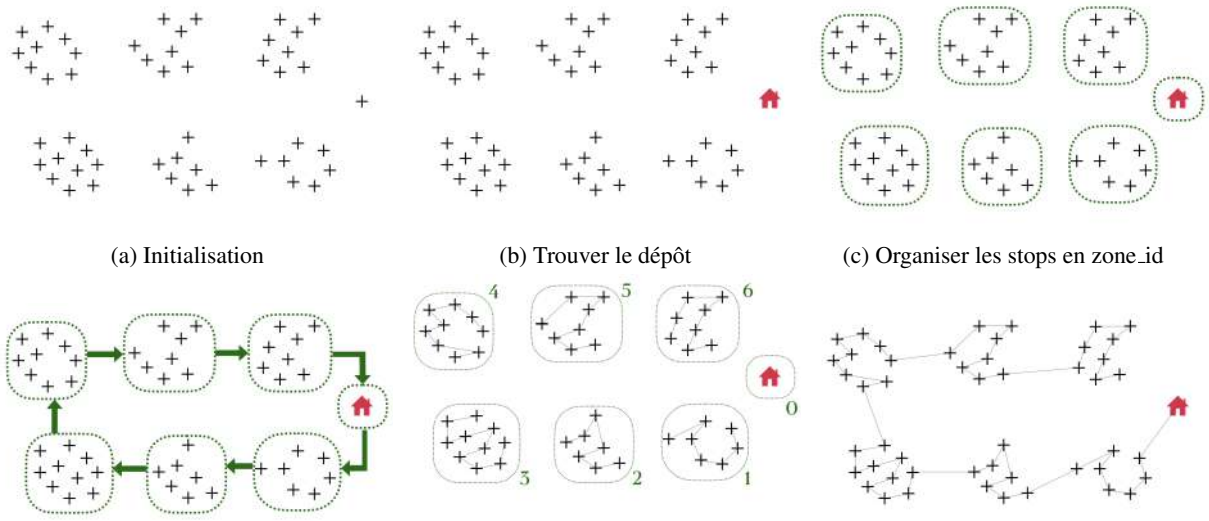
Afin de passer outre le problème d'explosion combinatoire, nous avons découpé le problème en sous-problèmes dont la solution peut être trouvé en un temps raisonnable. Les différentes étapes de l'algorithme son présentées sur la figure 1.1.

L'algorithme se compose de deux étapes majeurs : *i)* regrouper les stops par zone_id afin de trouver une route qui connecte toutes les zone_id et *ii)* trouver une route qui connecte les stops de chaque zone_id.

Néanmoins, si le nombre de zone_id est toujours trop grand pour le modèle alors une étape est ajoutée. Les zone_id sont réunis en clusters. Pour cela, une zone_id est choisie aléatoirement pour former un cluster avec ses plus proches voisins. En itérant, il est ainsi possible d'obtenir un nombre acceptable de clusters. Une fois ces clusters trouvées, il faut calculer un circuit reliant les clusters dans un premier temps, puis déterminer un chemin reliant les zone_id au sein de chaque cluster.

1.3 Résultats

Nous avons testé l'algorithme sur les routes du dataset d'entraînement fourni par Amazon et nous avons pris en compte deux paramètres pour évaluer ces solutions : Le score qui reflète la qualité de la solution et le temps de



(a) Initialisation (b) Trouver le dépôt (c) Organiser les stops en zone_id

(d) Trouver l'ordre de passage des stop_id dans chaque zone_id (e) Trouver l'ordre de passage des stops (f) Résultat

FIGURE 1.1 – Les grandes étapes de la recherche d'une séquence

calcul pour les impératifs du projet. Ainsi, sur un total de 100 routes testées, nous obtenons un score moyen de 0.098 pour un temps d'exécution total de 12 minutes. Ainsi, en moyenne, l'algorithme nécessite 7 secondes pour résoudre le problème sur une route.

Malheureusement, la solution que nous avons soumise ne nous a pas permis d'obtenir un score car le temps d'exécution dépassait les 2 heures. Cependant, en regardant les scores des autres participants, nous avons remarqué que les score qu'ils avaient obtenu était du même ordre de grandeur que le notre. Le modèle le plus performant avait un score de 0.03 et le dixième un score de 0.08.

Chapitre 2

Amélioration de la solution

L'objectif de ce chapitre est de présenterles améliorations que nous avons apportées afin de présenter un modèle plus performant. Nous avons travaillé autour de deux axes d'amélioration : augmenter la qualité de la solution en diminuant le score moyen et diminiuer le temps de calcul. En effet, l'objectif est de pouvoir prédire 3000 routes en moins de 4 heures.

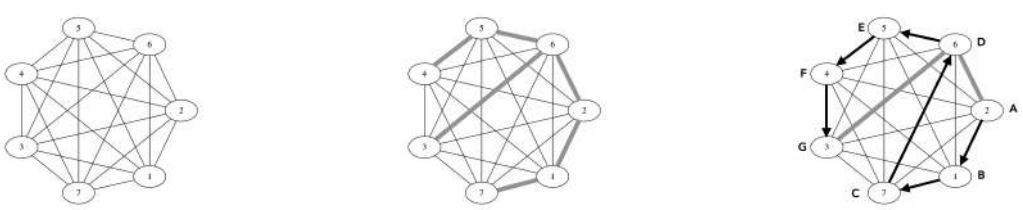
2.1 Solutions permettant de diminuer le temps de calcul

Dans un premier temps, nous avons cherché à diminuer le temps de calcul du modèle et de l'algotihme afin de cadrer avec les contraintes temporelles imposées par le projet. En effet, le temps de calcul du modèle ne doit pas excéder 4 heures pour prédire environ 3000 routes. Notre algorithme est capable de prédire en moyenne une route en 7 secondes. Pour prédire 3000 routes, il faudrait environ 12 heures, ce qui est beaucoup trop.

2.1.1 Ajout des bornes supérieurs et inférieurs au programme linéaire

Afin d'améliorer le temps de calcul du programme linéaire (cf annexe B), nous avons ajouté une borne supérieure et une borne inférieure à celui-ci. Ces deux bornes reposent sur le calcul d'un arbre couvrant de poids minimal. Pour trouver cette arbre couvrant, j'ai décidé d'utiliser l'algorithme de Kruskal [1].

Premièrement, la borne inférieure est simplement la somme des poids des arrêtes d'un arbre couvrant de poids minimal. Ensuite, la borne supérieure est la somme des poids des arrêtes d'un chemin trouvé en parcourant l'arbre couvrant. Pour trouvé ce chemin, il suffit de partir d'un sommet puis de parcourir les arcs de l'arbre couvrant sans revenir en arrière ni parcourir un sommet déjà visité. Si il n'est pas possible d'atteindre un sommet sans enfreindre cette régle alors il est possible d'emprunter des arcs ne faisant pas partie de l'arbre couvrant. Un exemple est présenté sur la figure 2.1



(a) un graphe décrivant le problème (b) Calculer l'arbre couvrant (c) Fabriquer la route

FIGURE 2.1 – Méthodologie pour trouver la borne supérieur

1. https://fr.wikipedia.org/wiki/Algorithme_de_Kruskal

Ces ajouts nous permettent de passer d'un temps de résolution moyen par problème de 0.087 secondes à 0.080 secondes.

2.1.2 Threading

Dans un premier temps, nous avons constaté que l'ordinateur utilisé par Amazon pour testé nos algorithmes était beacoup plus puissant que les ordinateurs portables que nous utilisions. En effet, l'ordinateur utilisé par Amazon comporte 16 coeurs et possède 64 GB de RAM. Les deux PC portables qui étaient à notre dispostion ne comportaient que deux coeurs et 8 GB de RAM.

Ainsi, afin de diminuer le temps de calcul, nous avons décidé de paralléliser la recherche des séquences à l'aide de Threads. Dans le code que nous avons soumis le 18 juin, le code comportaits 10 Threads. Chaque Threads calcul un dixième des séquences à prédire.

Pour savoir si le nombre de threads est suffisant, j'ai fait le calcul suivant. En moyenne, il faut 7 secondes pour prédire une séquence. Pour réaliser un calul pécimiste, je majore le temps de calcul à 30 secondes car c'est le pire temps que nous avons obtenue pour calculer une séquence. Ainsi, un thread peut calculer 120 séquences en 1 heures donc 480 séquences en 4 heures. Avec les 10 threads, l'algorithme peut calculer 4800 séquences en 4 heures. Ce qui est suffisant dans notre cas.

2.2 Solutions permettant d'augmenter la qualité de la solution

Dans un deuxième temps, nous avons voulu améliorer la qualité de la solution sans augmenter le temps de calcul. Nous avons ainsi développé de nouvelles méthodes dont un nouveau modèle de machine learning qui sont présentées dans cette parties. Pour avoir une idée de nos progrès, nous avons calculé à chaque étape sur un ensemble de 100 routes le score moyen des séquences prédites pour ces routes.

2.2.1 Modèle de la soumission

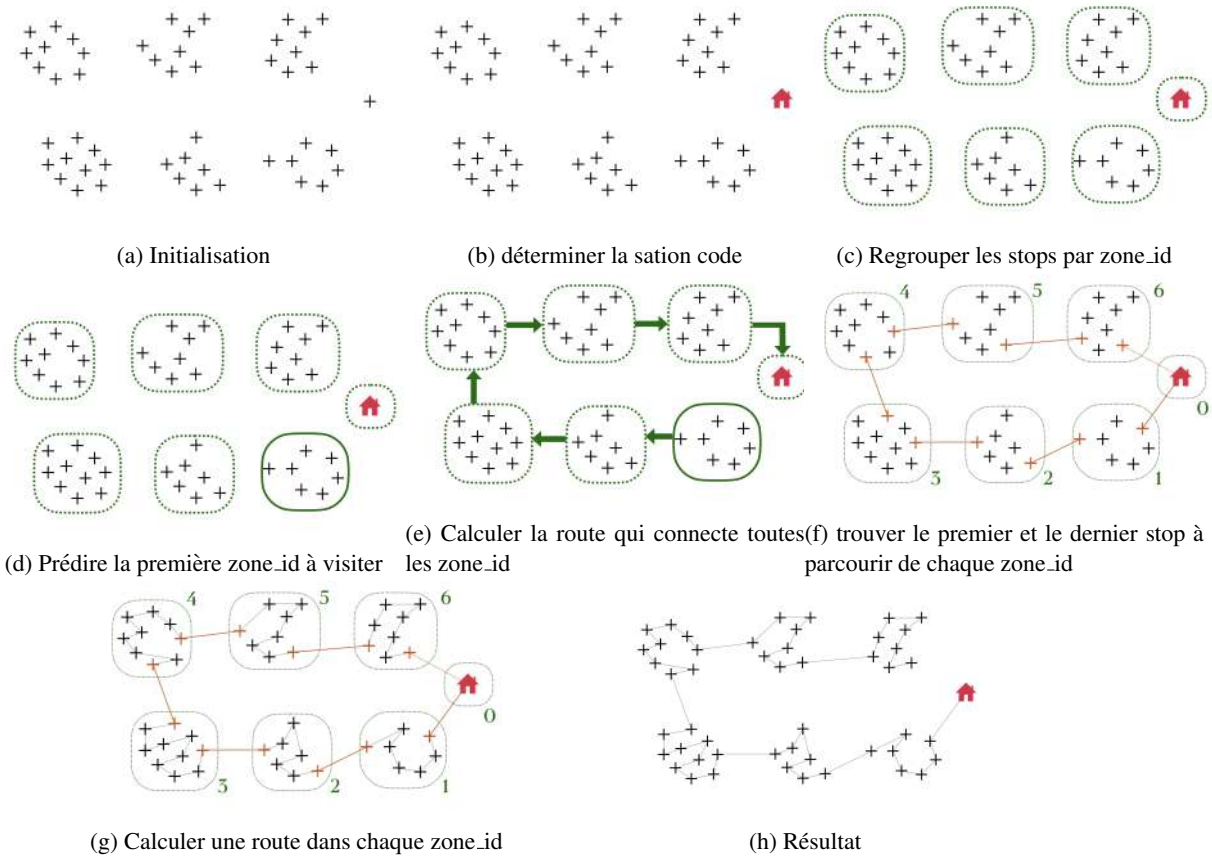
Le modèle que nous avons soumis le 18 juin reprend les grandes étapes de celui présenté dans la partie 1.2.4. Nous l'avons amélioré en changent quelques points de méthodologies qui vont être présenté tout au long de ce chapitre. La figure 2.2 récapitule le travail effectué et présente les grandes étapes de la recherche d'une séquence.

2.2.2 Imposer le point de départ et d'arrivé au sein de chaque zone_id

Comme décrit dans la partie 1.2.4, la seconde partie de l'algorithme est dédiée à la recherche d'une route reliant les stops d'une même zone_id. Pour cela, l'ancien modèle se contentait de trouver le plus court chemin reliant tout les stops sans ce soucier du stop de départ et du stop d'arrivé. Par conséquent, il est possible que le dernier stop parcouru dans une zone_id soit très éloigné du premier stop de la zone_id suivante. Ce qui peut amener l'algorithme à prédire des itinéraires inutilement long.

Pour amener plus de cohérence à l'algorithme, j'ai décidé d'imposer le point de départ et d'arrivé au sein de chaque zone_id. Pour trouver ces stops, j'ai décidé que le stop de départ d'une zone_id serait le stop le plus proche du stop d'arrivé de la zone_id précédente. De même, le stop d'arrivé d'une zone_id serait le stop le plus proche du stop de départ de la zone_id suivante. Ainsi, pour chaque paire successive de zone_id, il suffit de trouver les deux stops appartenant chacun à une zone_id qui soit les plus proche. Cette étape correspond à la figure 2.2f présentée dans la section 2.2.1

Le fait d'imposer le point de départ et le point d'arrivé au sein d'une zone_id a permis de passer d'un score moyen de 0.098 à 0.083.



(a) Initialisation (b) déterminer la sation code (c) Regrouper les stops par zone_id

(d) Prédire la première zone_id à visiter (e) Calculer la route qui connecte toutes les zone_id (f) trouver le premier et le dernier stop à parcourir de chaque zone_id

(g) Calculer une route dans chaque zone_id (h) Résultat

FIGURE 2.2 – Etapes majeurs de l'algorithme soumis le 18 juin

2.2.3 Gérer les sous-problèmes non résolus par le Programme Linéaire

Ensuite, nous avons voulu savoir si le solveur renvoyé une solution optimale pour tout les sous-problèmes. En effet, Le solveur que nous utilisons est capable de renvoyer 6 status :

- UNDEFINED= 0. La solution faisable n'a pas été trouvée (mais peut exister).
- INFEASIBLE = 1. Le problème n'a pas de solution réalisable.
- FEASIBLE = 2. Solution réalisable trouvée.
- OPTIMAL = 3. La solution optimale existe et a été trouvée.
- UNBOUNDED = 4. La fonction de coût est illimitée. Le Mip ou au moins le problème relaxé est illimité.
- NOT SOLVED = 5. Le paramètre par défaut avant qu'un problème ne soit résolu

Les status "FEASIBLE", "OPTIMAL" permettent d'obtenir une séquence optimale. C'est à dire que le solver trouve bien une solution. Les autres status ("UNBOUNDED","UNDEFINED et "INFEASIBLE") ne permettent pas d'obtenir une séquence. Pour résoudre ce problème, nous avions dans un premier temps décidé de retourner une séquence aléatoire dans l'un de ce cas.

Or, en réalisant des tests, j'ai remarqué que, sur 2074 tentatives de résolutions (de sous-problèmes), 12,5 % de ces tentatives retournent "UNBOUNDED" ou "INFEASIBLE" (les 87,5% restant était "OPTIMAL"). Cela pose un vrai problème pour notre modèle car si 12,5% des sous-problèmes ne sont pas correctement résolu alors la qualité de la solution globale ne peut être que amoindri.

Pour résoudre ce problème, j'ai essayé de modifier les contraintes du modèle (cf annexe B) afin de remplacer cette séquence aléatoire par une séquence qui se rapproche d'une séqeunce optimale. J'ai émis l'hypothèse que l'origine de ce problème est l'élagage des matrices (cf annexe C) couplé à la détermination des points de départ et d'arrivés à chaque sous-problème. En effet, les matrices que nous passons en paramètre possède un ou plusieurs

chemin hamiltonien [2] mais rien ne nous assure que l'un de ces chemins commence ou finit par un des points imposés.

Afin de traiter ces deux cas, j'ai décidé de relacher progressivement des contraintes. Dans un premier temps, je supprime les contraintes concernant les fenêtres temporelles. Ensuite, si la première solution ne fonctionne pas, je retire la contrainte du point d'arrivé et du point de départ en réintégrant les contraintes temporelles. En dernier recours, je retire les contraintes temporelles et les points de départ et d'arrivé.

Dans 95% des cas, Cette méthode permet de trouver une séquence optimale pour le problème relâché alors qu'il ne renvoyait pas "OPTIMAL" dans un premier temps. Ainsi, 99% des sous-problèmes retournent une solution étiquetée "OPTIMAL". Grâce à ce travail, le score est passé de 0.083 à 0.076. Ainsi, la qualité de la solution est grandement améliorée.

2.2.4 Changer la manière de Classifier les zone_id

Ensuite, j'ai décidé de modifier la manière dont les zone_id sont regroupées si leur nombre dépasse celui admissible par le solveur car je ne le trouvais pas satisfaisant.

En effet, nous choisissions au hasard une zone_id et nous la regroupions avec ces plus proches voisines (aus sens de la distance en temps qui les sépare). Le principale problème de cette solution est la dépendance de la qualité de la solution vis-à-vis de l'initialisation. En effet, si le premier stop choisi est très éloigné de tout les autres alors le regroupement n'a pas de sens.

Maintenant, nous utilisons un modèle totalement déterministe en plusieurs étapes. Dans un premier temps, nous trions les temps de parcours moyen entre deux zone_id (c'est à dire le temps de parcours moyen des trajets joignant deux stops contenus dans une zone_id différente) par ordre croissant. Ensuite, nous sélectionnons les deux zone_id qui ont la distance la plus petit. Ces deux zone_id vont être regroupé pour former un cluster et ainsi de suite jusqu'à temps que l'on ait le nombre de cluster souhaité. Le pseudo-code de cette méthode est présenté dans l'algorithme 1.

Algorithm 1: same_size_cluster

Data: an adjacency matrix **P**, a list of the zone_id with the stops contains in each zone_id **list_zi**, the maximal number of cluster **max**
Result: a list of cluster of zone_id
coeff ← a decreasing list of the elements of P ;
res ← [] (an empty list) ;
while *the number of cluster is larger than max* **do**
  zi_start,zi_arrival ← the zone_ids which correpond to the greater coefficient of the list coeff ;
  delete the greater coefficient of the list coeff;
  if *zi_start and zi_arrival are not already in a cluster* **then**
    adding a cluster formed of zi_start and zi_arrival in res
  else
    adding zi_start to the cluster of zi_arrival in res if zi_arrival is already in a cluster and vice versa
  end
end
return res

Cette nouvelle manière de regrouper les zone_id nous donne un score moyen sur 100 routes de 0,068 alors que l'ancienne méthode donait un score moyen de 0,076. Nous avons ainsi diminué drastiquement le score grâce à cette innovation.

2. chemin qui passe par tous les sommets une fois et une seule

2.2.5 Première tentative de Post-Processing

Afin d'amlériorer la qualité de la solution, nous avons pensé à ajouter une étape de Post-Processing à notre algorithme. Cette étape consite, une fois que l'algorithme a trouvé une séquence, à créer une matrice d'adjacence comportant les arcs effectivement emprunté (ceux de la séquence trouvé) et d'y ajouter des arcs dont la probabilité qu'ils soient empruntés sont la plus élevé. La probabilité qu'un arc soit emprunté ou non est déterminé à l'aide du modèle de Machine Leanging (cf annexe C). Ainsi, nous pouvons relancer une recherche de séquence sur le graphe complet avec un temps de calcul raisonnable (si on ajoute un nombre raisonnable d'arcs). Or, comme montré sur la figure F.2, le temps de calcul est élevé (plus de 20 seconde) et ne permet pas d'améliorer le score. Ainsi, cette idée a été abandonné.

2.2.6 Développer un modèle de Machine Learning pour trouver la premier zone_id à visiter

Dans cette section, nous allons présenter le développement du modèle de Machine Learning qui permet de trouver la première zone_id à parcourir comme montré sur les figure 2.2d et 2.2e

2.2.6.1 Motivation de cette recherche et hypothèses

J'ai voulu revenir à l'étude statistique afin de trouver des éléments qui nous permettraient de différencier les routes de label "High", "Low" et "Medium" et ainsi développper un modèle permettant de ne prédire que des routes label "High". Pour cela, je me suis intéressé au caractéristiques des paquets livrés. Sur la figure I.3, j'ai tracé la taille moyenne pour toutes les routes classées selon leur label de la taille maximale des colis déposés à un point de livraison selon l'ordre de passage de celui-ci. Nous pouvons remarquer que les livreurs ont tendance à livré les plus gros colis au début de la séquence de livraison. Le graphique présenté sur la figure I.3a ne présente la moyenne que sur 100 routes alors que les autres labels possèdent plus de 1000 routes.

En définitive, il semblerait que les livreurs ont, en moyenne, une préférence pour parcourir un certains type de stop en début de séquence. Pour essayer d'intégrer ces préférences dans l'algorithme, nous avons pensé à développer un modèle de machine learning capable de prédire le premier stop à visister. Or, nous avons pensé qu'il serait trop difficile de développer un modèle capable de trouver le premier stop. A la place, nous avons décidé de développer un modèle capable de prédire la première zone_id à parcourir et ensuite laisser le modèle déterminer le premier stop à livrer.

2.2.6.2 Développement de différents modèles

Dans un premier temps, notre but est de trouver un modéle capable de prédire la variable **Is_first_zi** qui vaut 1 si la zone_id est la première visitée après le point de départ et 0 sinon. Afin de prédire cette variable, j'ai calculé 12 varibales explicatives présentées en gras en annexe G. Pour trouver une relation, nous avons entrainé une régression logistique avec une pénalité l2 et un coefficient de 0.01. Les résultats présentés sur la figure I.2 ne nous ont pas satisfait. Ce qui nous a conduit à développer un second modèle.

Ensuite, afin de développer ce second modèle, j'ai créé des nouvelles variables explicatives. Celle ci sont présentées en annexes G et sont regroupées dans les 4 sections suivantes : Variables concernant la zone_id, Variables concernant le dépôt, Variables faisant le lien entre le barycentre de la route et la zone_id, Variables sur les enveloppes concaves et convexes. Les varibales qui ont demandé le plus de réflexions sont celles faisant intervenir le barycentre de la route, l'enveloppe convexe et l'enveloppe concave. En effet, nous avions l'intuition que la manière dont était agencées les zone_id pouvaient avoir un influence sur le choix de la première zone_id à visiter.

Les résultats sont présentés sur la figure I.2 en annexe. L'ajout de toutes ces varibles n'a pas amélioré les performances du modèle. Nous avons également entrainé un Random Forest [3] sans plus de résultats.

2.2.6.3 Intégration du modèle de ML dans l'algorithme

Malgrè la mauvaise qualité prédictive des modèles, nous avons tout de même désiré l'intégrer à l'algorithme pour savoir si il est capable d'améliorer la qualité de la solution en diminuant le score. Pour cela, j'ai intégré un modèle ML simplifié (régression logistique avec pénalité l2 et c :0.001) dans l'algorihtme avec uniquement 4 features : Tps_depot_to_first_zi, Km_depot_to_first_zi_ind_norm, distane_barycentre_ind_norm, convex, Nb_colis. J'ai décidé de prendre ces quatre variables car ce sont celle qui avait le plus d'importance dans le deuxième modèle comme le montre la figure H.3.

Le modèle donne la probablitié qu'une zone_id soit la première visité. Ainsi, celle qui a la plus grande probabilité est désignée comme celle qui sera parcouru en premier par le modèle. Sur les routes testées, le score à diminué de 0.1 (il est passé de 0.07 à 0.06). Ainsi, ce modèle amèliore la qualité de la solution. Nous avons donc décidé de la conserver pour notre modèle

2.3 Résultats

Notre modèle a obtenu un score de 0.065.

3. https://blog.ysance.com/algorithme-n2-comprendre-comment-fonctionne-un-random-forest-en-5-min

Bibliographie

[Applegate et al., 2011] Applegate, D. L., Bixby, R. E., Chvátal, V., and Cook, W. J. (2011). *The traveling salesman problem*. Princeton university press.

[Bayliss, 2021] Bayliss, C. (2021). Machine learning based simulation optimisation for urban routing problems. *Applied Soft Computing*, 105 :107269.

[Chen and Ng, 2004] Chen, L. and Ng, R. (2004). On the marriage of lp-norms and edit distance. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB '04, page 792–803. VLDB Endowment.

[Chen et al., 2019] Chen, X., Ulmer, M. W., and Thomas, B. W. (2019). Deep q-learning for same-day delivery with a heterogeneous fleet of vehicles and drones. *arXiv preprint arXiv :1910.11901*.

[Cordeau and Groupe d'études et de recherche en analyse des décisions (Montréal, 2000] Cordeau, J.-F. and Groupe d'études et de recherche en analyse des décisions (Montréal, Q. (2000). *The VRP with time windows*. Groupe d'études et de recherche en analyse des décisions Montréal.

[Dantzig and Ramser, 1959] Dantzig, G. B. and Ramser, J. H. (October 1959). The truck dispatching problem. *Management Science*, 6(1) :80–91.

[Dondo and Cerdá, 2007] Dondo, R. and Cerdá, J. (2007). A cluster-based optimization approach for the multi-depot heterogeneous fleet vehicle routing problem with time windows. *European Journal of Operational Research*, 176(3) :1478–1507.

[Feynman and Vernon Jr., 1963] Feynman, R. and Vernon Jr., F. (1963). The theory of a general quantum system interacting with a linear dissipative system. *Annals of Physics*, 24 :118–173.

[Fisher, 1995] Fisher, M. (1995). Vehicle routing. *Handbooks in operations research and management science*, 8 :1–33.

[Gutin and Punnen, 2006] Gutin, G. and Punnen, A. (2006). *The Traveling Salesman Problem and Its Variations*. Combinatorial Optimization. Springer US.

[King, 2008] King, J. E. (2008). Binary logistic regression. *Best practices in quantitative methods*, pages 358–384.

[Li, 2013] Li, H. (2013). Generalizations of dirac's theorem in hamiltonian graph theory—a survey. *Discrete Mathematics*, 313(19) :2034–2053.

[Mancel, 2004] Mancel, C. (2004). *Modelisation et resolution de problemes d'optimisation combinatoire issus d'applications spatiales*. Theses, INSA de Toulouse.

[Miller et al., 1960] Miller, C. E., Tucker, A. W., and Zemlin, R. A. (1960). Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7(4) :326–329.

[Mitchell et al., 2011] Mitchell, S., OSullivan, M., and Dunning, I. (2011). Pulp : a linear programming toolkit for python. *The University of Auckland, Auckland, New Zealand*, page 65.

[Mourid St-Pierre, 2012] Mourid St-Pierre, B. (2012). *Optimisation de distribution de biens et services : cas de Neslté pour la distribution des surgelées et des glaces*. PhD thesis, Haute école de gestion de Genève.

[Nalepa, 2019] Nalepa, J. (2019). *Smart Delivery Systems : Solving Complex Vehicle Routing Problems*. Intelligent Data-Centric Systems : Sensor Collected Intelligence. Elsevier Science.

[Nemhauser et al., 1989] Nemhauser, G. L., Kan, A. R., and Todd, M. (1989). *Handbooks in operations research and management science*. North-Holland Amsterdam.

[Pisinger, 2007] Pisinger, David ; Røpke, S. (2007). A general heuristic for vehicle routing problems.

[Poullet, 2020] Poullet, J. (2020). *Leveraging Machine Learning to Solve the Vehicle Routing Problem with Time Windows*. PhD thesis, Massachusetts Institute of Technology (MIT).

[Romain, 2008] Romain, A. (2008). Introduction au langage de modélisation gnu mathprog (glpk). *GDF-Suez, Direction de la Recherche et de l'Innovation*.

[Sakarovitch, 1984] Sakarovitch, M. (1984). *Optimisation combinatoire : Graphes et programmation linéaire*. Collection Enseignement des sciences. Hermann.

Annexes

Annexe A

Les dataset et les variables du projet

Dataset	Champs de données	Description et plage de valeurs
Route information	Route ID	une chaîne alphanumérique qui identifie de manière unique chaque route
	Station Code	une chaîne alphanumérique, identifie de manière unique la gare (ou le dépôt) où l'itinéraire a commencé
	Date	la date à laquelle le véhicule de livraison a quitté la gare
	Departure Time	l'heure de déepart du véehicule de livraison de la gare, indiqu ee en UT
	Executor Capacity	la capacité du véhicule de livraison, spécifiée en cm^3
	Stops	Arrêts sur l'itinéraire à desservir
	Observed sequence	Séquence observée dans laquelle les arrêts sur l'itinéraire ont été desservis
	Route Score	variable catégorique indiquant la qualité de la séquence d'arrêt observée (high, medium, low)
Stop information	Stop ID	un code identifiant chaque arrêt sur un parcours (unique au sein de chaque parcours)
	Latitude / Longitude	latitude et longitude de chaque arrêt spécifiées via le système de projection WGS 84
	Type	variable catégorielle désignant le type d'arrêt (gare ou livraison)
	Zone ID	identifiant unique de la zone de planification géographique dans laquelle se situe l'arrêt
	Packages	colis à servir à chaque arrêt
	Distances	distances à tout autre arrêt sur le même itinéraire, la moyenne des temps de trajet historiquement réalisés (en secondes)
Package information	Package ID	identifiant unique de chaque colis (unique sur l'ensemble des données)
	Time window	heures de début et de fin : contraintes de créneaux horaires de livraison sur certains colis

Dataset	Champs de données	Description et plage de valeurs
	Planned service time	temps que le service de ce paquet devrait prendre
	Dimensions	largeur, longueur, hauteur maximales du colis Colis

TABLE A.2 – tableau présentant les varaibles contenues dans les trois dataset fournis pour le projet

Annexe B

Modèlisation mathématique du TSPTW

L'ensemble des itinéraires entre les différents points de livraison du réseau est représenté par un graphe orienté connexe $G = (V, E)$ où l'ensemble des sommets V correspond aux points de livraison et l'ensemble des liens orientés E correspond aux chemins entre ces points.

Les variables du problème sont : la variable binaire $x_{i,j}$ pour tout (i, j) dans E est égale à 1 si l'arc (i, j) est choisi dans la solution et 0 sinon ; le temps de parcours de chaque arc est représenté par $p_{i,j}$; la variable t_i pour chaque sommet i dans V représente l'heure à laquelle le consommateur i a été livré, où $t_i \in [a_i, b_i]$ qui représente la fenêtre de temps dans laquelle le colis doit être livré au client. Le sommet de départ v_0 de l'itinéraire est connu ainsi que le sommet v_f final.

Le problème revient à minimiser la fonction objectif

$$\sum_{(i,j)\in E} x_{i,j} * p_{i,j} \tag{B.1}$$

soumis à

$$\begin{cases} \displaystyle\sum_{(i,j)\in E} x_{i,j} = 1 & \forall i \in V - \{v_f\} \quad \text{(B.2a)} \\ \displaystyle\sum_{(i,j)\in E} x_{i,j} = 1 & \forall j \in V - \{v_0\} \quad \text{(B.2b)} \\ \displaystyle\sum_{(i,j)\in E} x_{i,j} = \sum_{(i,j)\in E} x_{j,i} & \forall i \in V - \{v_0, v_f\} \quad \text{(B.2c)} \\ \displaystyle\sum_{(i,j)\in E} x_{i,j} - \sum_{(i,j)\in E} x_{j,i} = 1 & i = v_0 \quad \text{(B.2d)} \\ \displaystyle\sum_{(i,j)\in E} x_{i,j} - \sum_{(i,j)\in E} x_{j,i} = -1 & i = v_f \quad \text{(B.2e)} \\ x_{i,i} = 0 & \forall i \in V \quad \text{(B.2f)} \\ t_i + p_{i,j} - t_j \leq M * (1 - x_{i,j}) & \forall (i, j) \in E \quad \text{(B.2g)} \\ a_i \leq t_i \leq b_i & \forall (i, j) \in E \quad \text{(B.2h)} \\ x_{i,j} \in \{0, 1\} & \forall (i, j) \in E \quad \text{(B.2i)} \end{cases}$$

La fonction objectif (Eq. (B.1)) vise à minimiser le temps de parcours total d'une route. Les contraintes (B.2a) et (B.2b) exigent que chaque client soit servi une et une seule fois, Eqs (B.2c), (B.2d) et (B.2e) assurent la conservation du flux. Eqs (B.2f) et (B.2g) (M suffisamment grand, c'est-à-dire supérieur au maximum de $p_{(i,j)}$ + maximum

de t_i) imposent le problème de non avoir un cycle. Ces équations font partie de la formulation MTZ (voir aussi [Miller et al., 1960]). Eq. (B.2h) impose le respect des contraintes de temps. Enfin, Eq. (B.2i) requiert que la variable de décision soit un booléen.

Annexe C

Modèle d'élagage

C.1 Définition du modèle

En bref, nous utilisons un problème de classification binaire deS liens pertinents et non pertinents, chaque exemple traité par le modèle de Machine Learning correspondant à un lien dirigé de deux arrêts $s \to s'$. Pour ce faire, nous devons définir des variables explicatives pertinentes pour le modèle :

- `distance time` : time required to travel (the link) from stop s to stop s'
- `five closest1` : average time taken from stop s to go to the 5 stops closest to it (close in the sense of distance time)
- `same zone` : boolean variable which is equal to 1 if stop s and s' are in the same zone id, 0 otherwise
- `planned time service2` : time taken by the delivery man to deliver all the packages from stop s'

La variable à prédire est y, qui vaut 1 si le lien $s \to s'$ est utilisé par la séquence, 0 sinon. Nous entraînons notre modèle sur les routes de l'ensemble d'apprentissage qui sont dans la zone id de label "High" en utilisant un modèle de Régression Logistique avec une validation croisée 10 fois (avec pénalité L_2). Cette méthode est préférée aux autres modèles comme SVM, réseaux de neurones artificiels,... pour ses performances, ses temps d'exécution et son explicabilité.

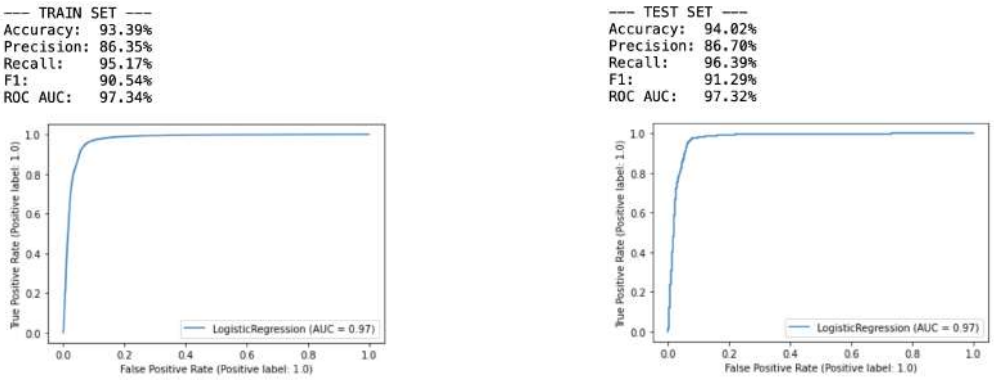
Le but est de construire un modèle capable de prédire la valeur de la variable cible y étant donné les valeurs de 4 co-variables $X_1, \ldots, X_4$. Le modèle de régression logistique est défini par : $\log(\frac{p}{1-p}) = a_1 X_1 + \cdots + a_4 X_4$ où $a_i \in \mathbb{R}$ sont les paramètres du modèle et $p = Prob(Y = 1|X = x)$. Les paramètres a_i sont estimés par la méthode du maximum de vraisemblance avec un ensemble de données historiques (voir aussi [King, 2008]).

Sur l'ensemble de test, ce modèle donne un score F1 de $0,91$ avec un rappel de 0,96$etuneprécisionde0,86$ (voir Fig. C.1). Ce modèle est considéré comme suffisamment bon pour être utilisé pour notre prédiction.

C.2 Utilisation du modèle pour diminuer le temps de calcul

L'étape suivante consiste en un processus qui utilise ce modèle de classification binaire pour élaguer (couper) les liens $s \to s'$ dans chaque cluster qui ne seront pas utilisés pour la séquence de route. Cependant, nous devons calibrer la tendance à l'élagage pour nous assurer qu'il y a toujours un chemin que nous pouvons emprunter après l'élagage des maillons. Dans ce cas, une condition suffisante qui assure l'existence d'un tel chemin est donnée par le théorème de Dirac (voir [Li, 2013]). Rappelons qu'un cycle hamiltonien dans un graphe $G = (V, E)$ est un cycle qui visite chaque sommet exactement une fois.

Theorem 1 *(Dirac's theorem) Let $G = (V, E)$ be a graph with n vertices in which each vertex has degree at least*



(a) The performance on training set

(b) The performance on testing set

FIGURE C.1 – The performance of our ML model to predict if a vertex is significant on routes label "High"

$n/2$. *Then* G *has a* ***Hamiltonian cycle***.

Nous avons donc utilisé cette condition pour la taille. L'idée est que, dans chaque cluster de taille n, nous élaguons de telle sorte que le degré de chaque stop soit d'au moins $\frac{n}{2} + 1$ (degrés comptés avec les liens non orientés). Pour un lien donné, plus la prédiction du modèle est proche de 0, plus la probabilité que le lien soit pertinent est faible, c'est-à-dire que ce lien ne sera pas utilisé pour la séquence de routage, et nous allons donc l'élaguer.

Cependant, le théorème de Dirac ne s'appliqe que dans le cas de graphe non orienté, il faut donc automatiquement élaguer le lien $s' \longrightarrow s$ si l'on sait que le lien $s \longrightarrow s'$ est élagué. Le pseudo code utilisé pour élaguer les liens dans un cluster est présenté dans l'algorithme 2.

Algorithm 2: Pruning links in a cluster

Result: A cluster in the form of a matrix by using the Machine Learning model
Input: road_id, ss_adj_matrix, model, scaler, stop_data, route_data, package_data
$n \leftarrow$ number of rows in the matrix;
for *s in stops of a cluster* **do**
  if *degree of* $s > n/2$ **then**
    number of pruning arcs remaining $\leftarrow n/2 -$ (degree of s) $- 1$;
    find the list of stop (arc) s' which is still linked to stop s;
    calculate the sum of (model.predict(arc $s \longrightarrow s'$) + model.predict(arc $s' \longrightarrow s$)) and then store them in a list l;
    sort elements in list l in ascending order;
    remove ($2 \times$ number of pruning arcs remaining) links [a] of the matrix which correspond to (number of pruning links remaining) first values of elements in list l;
  end
end
return An adjacency matrix

a. We take into account 2 times because it is necessary to count the outgoing arcs but also the incoming links of s

En général, en utilisant cette méthodologie, la majorité des liens dirigés dans les clusters sont élagués entre 30% et 50%. Cependant, certaines grappes ne sont pas élaguées ; celles-ci correspondent à de très petits clusters qui contiennent la minorité d'arrêts. Techniquement, on peut noter que la méthode des arcs d'élagage conserve souvent plus de 60% des arcs pertinents à utiliser plus tard pour la prochaine étape de la recherche opérationnelle

(RO).

Annexe D

Utilisation de la commande RC-CLI

Il existe trois phases d'analyse de l'utilisation du Routing Challenge RC-CLI : ***Model-Build, Model-Apply*** et ***Model-Score***. Après avoir créé une application sur **Docker image** en utilisant la commande [`rc-cli new-app app-name`] sur le terminal, nous pouvons tester les trois phases d'analyse. En effet, la Docker image de notre application exécutera le code source pour chacune de ces phases suivantes :

- La phase de **Model-Build** ([`rc-cli model-build`]) : Elle aura accès aux données historiques sur les itinéraires connus et produira un modèle présenté dans la partie 1.2 capable de prédire les séquences d'itinéraires
- La phase de **Model-Apply** ([`rc-cli model-apply`]) : Elle appliquera ce modèle à de nouvelles données et prédira les séquences d'itinéraires. En effet, le model-apply met en production le modèle construit dans la phase précédente
- La phase de **Model-Score** ([`rc-cli model-score`]) : Elle comparera les parcours proposés aux séquences réelles (actual sequences) et fournira un score. La métrique utilisée pour calculer ce score est présentée dans l'annexe E

Lorsqu'elle sera évaluée dans le cadre du concours par Amazon (le 15 mai), notre soumission passera par les 4 étapes suivantes :

1. Réinitialiser tous les données dans le dossier **data** : [`rc-cli reset-data`]
2. Construire un modèle à l'aide des données d'entraînement fournies par Amazon : [`rc-cli model-build`]
3. Appliquer notre modèle à un nouvel ensemble de données : [`rc-cli model-apply`]
4. Donner un score sur notre modèle : [`rc-cli model-score`]

Il est possible d'exécuter toutes ces étapes sur un modèle enregistré avec une seule commande en utilisant [`rc-cli production-test`] en notant que nous pouvons aussi vérifier avec cette commande en local afin de tester notre modèle avant de la soumettre.

Finalement, afin de créer une soumission pour ce challenge, il faut donc créer un snapshot de notre modèle pour sauvegarder notre application de Docker image en utilisant la commande [`rc-cli save-snapshot snapshot-name`]. Il faut noter aussi qu'après avoir enregistré l'image Docker, tous les participants sont fortement encouragés à valider leurs solutions à l'aide de la commande [`rc-cli production-test snapshot-name`] avant de soumettre les fichiers image Docker via la plate-forme d'Amazon.

Annexe E

Critère d'évaluation

$$route\_score\,^{1} = \frac{SD(A,B).ERP_{norm}(A,B)}{ERP_e(A,B)}$$

où A est la séquence réalisée historiquement et B est la séquence soumise par l'utilisateur. Une route_score de 0,0 signifie un score parfait et 1,2 est le score maximum possible.

Sequence deviation (SD)

Le SD mesure la différence entre la séquence proposée et la séquence de référence. Il est calculé grâce au processus décrit ci-dessous :

- Initialiser la liste vide a
- Parcourir tous les arrêts de la séquence B. A chaque arrêt, trouver le même arrêt dans la séquence A et ajouter son index i dans a
- Une fois tous les arrêts de la séquence B parcourus, SD est donné par

$$SD = \frac{2}{n(n-1)} \sum_{i=1} n|a_i - a_{i-1}| - 1$$

n est le nombre total d'arrêts dans la séquence B

Edit distance with Real Penalty (ERP)

Les distances ERP sont une variante de la distance d'édition, alias la distance de Levenshtein. Ils mesurent le nombre d'opérations à élément unique (insertions, suppressions et substitutions) nécessaires pour transformer la séquence proposée en séquence de référence.

Tout d'abord, ERP_{norm} (Normalized edit distance with real penalty) est calculé grâce à la procédure décrite ci-dessous.

Ensuite, ERP_e (ERP Edit Count) compte le nombre d'opérations d'édition (insertions, substitutions ou suppressions) nécessaires pour transformer la séquence B en séquence A lors de l'exécution de la formulation récursive ERP_{norm}.

Procédure pour calculer ERP_{norm}

- Calculer les temps de trajet normalisés entre tous les arrêts de l'itinéraire
- Calculer Edit Distance with Real Penalty(ERP) en fonction de la formule récursive suivante :

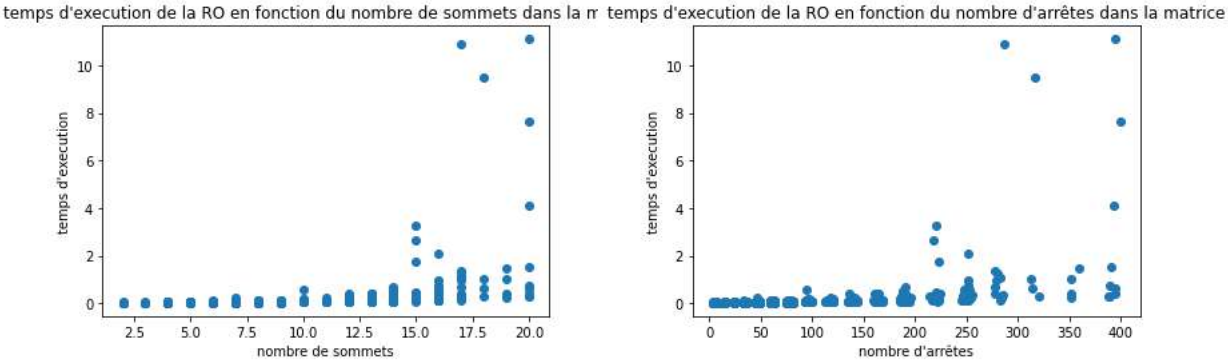
1. Lien vers la scoring metric github : `https://github.com/MIT-CAVE/rc-cli/tree/main/scoring`

$$ERP_{norm}(A,B) = \begin{cases} \sum_{i \in A} time_{norm}(A[i], g) & if B = \{\}, \\ \sum_{i \in B} time_{norm}(B[i], g) & if A = \{\}, \\ min \begin{cases} ERP_{norm}(tail(A), B) + time_{norm}(A[0], g), \\ ERP_{norm}(A, tail(B)) + time_{norm}(B[0], g), \\ ERP_{norm}(tail(A), tail(B)) + time_{norm}(A[0], B[0]), \end{cases} & otherwise, \end{cases}$$

où $time_{norm}(x, y)$ est le temps de trajet normalisé entre l'arrêt x et l'arrêt y, tail(X) est la séquence X sans son premier arrêt, X[0] est le premier arrêt dans séquence X, et g est un temps très long associé à tout arrêt présent dans une séquence mais pas dans l'autre (cela ne devrait pas être le cas) ;

Annexe F

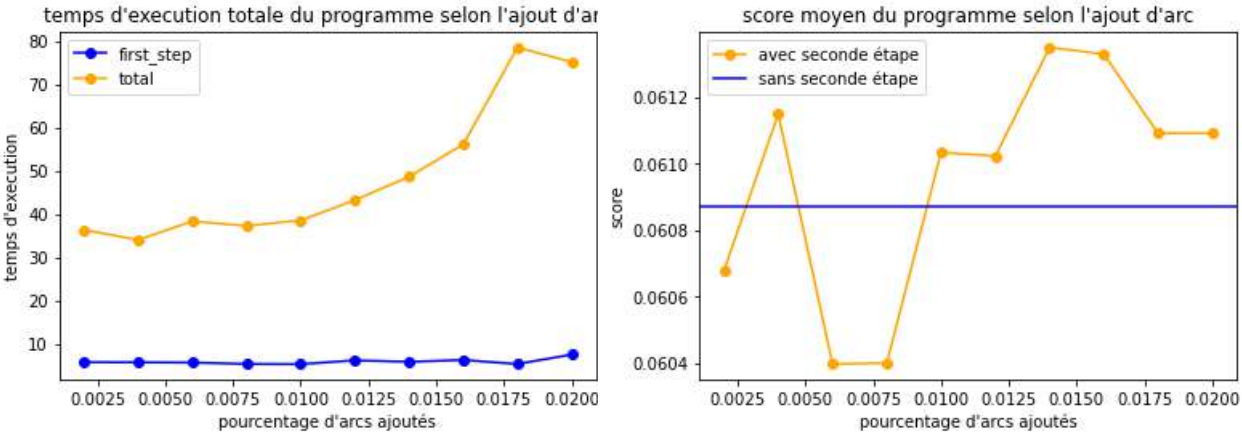
Etude du temps de calcul de l'algorithme et du score de la solution retournée



(a) Temps de calcul en fonction du nombre de sommets

(b) Temps de calcul en fonction du nombre d'arc

FIGURE F.1 – Étude du temps de calcul de l'alogirthme soumis le 15 juin



(a) étude du temps de calcul

(b) étude du score

FIGURE F.2 – Comparaison de la qualité de la solution et du temps de calcul avec et sans une étape de post-processing

Numéro du test	**% d'arcs ajouté**	**Écartement maximale de 2 stops**	**Dataset utilisé**	**Score avant**	**Score après**
1	0.01	4	100 routes high ayant eu le meilleur score avec l'algorihtme de la soumission	0.012	0.016
2	0.01	4	100 routes high ayant eu le pire score avec l'algorihtme de la soumission	0.188	0.183
3	0.02	les stops ne sont pas dans la même zone_id	40 routes high ayant eu le pire score avec l'algorihtme de la soumission	0.228	0.246
4	0.02	6	40 routes high ayant eu le pire score avec l'algorihtme de la soumission	0.228	0.273
5	0		200 routes high ayant eu le pire score avec l'algorihtme de la soumission	0.112	0.112

TABLE F.1 – Résultats des tests sur l'algorithme utilisant le post-Processing

Annexe G

Liste des variables explicatives utilisées dans les différents modèles de Machine Learning pour trouver la première zone_id à visiter

1 Variables concernant la zone_id

- **Nombre_stops_in_zi** : nombre de stops contenus dans la zone_id
- **Moyenne_tps_in_zi** : temps moyen mis pour passer d'un stop à un autre à l'intérieur de la zone_id
- **Max_vol_colis** : volume du plus gros colis à livrer dans la zone_id
- **Nb_colis** : nombre total de colis à livrer au différents stops contenus dans la zone_id

2 Variables concernant le dépôt

- **Day** : jour auquel à eu lieu la livraison
- **Hour** : heure à laquelle débute la livraison
- **Lat_pt_depart** : latitude du point de départ
- **Long_pt_depart** : longitude du point de départ
- executor_capacity : la capacité du camion de livraison
- **Tps_depot_to_first_zi** : temps (en seconde) de parcours entre le dépot (pt de départ) et le barycentre de la zone_id
- **Km_depot_to_first_zi** : distance (en Km) entre le dépot (pt de départ) et le barycentre de la zone_id
- Km_depot_to_first_zi_ind : numéro attribué à la zone_id en fonction de la valeur de Km_depot_to_first_zi. 0 pour la zone_id ayant la plus petite valeur, 1 pour la seconde zone_id avec la plus petite valeur et ainsi de suite
- Km_depot_to_first_zi_ind_norm : Km_depot_to_first_zi_ind diviser par le nb de zone_id dans la route
- shortest_dist : distance entre le stop le plus proche du dépôt (dans la zone_id) et celui-ci
- shortest_dist_ind : numéro attribué à la zone_id en fonction de shortest_dist. 0 pour la zone_id avec la plus petite valeur, 1 pour la seconde zone_id avec la plus petite valeur et ainsi de suite.
- shortest_dist_ind_norm : shortest_dist_ind diviser par le nombre de zone_id dans la route

3 Variables faisant le lien entre le barycentre de la route et la zone_id

— **distane_barycentre** : distance (en Km) entre le barycentre des stops du cluster et le baryencetre de tout les stops de la route
— **cosinus_barycentre_first_zi** : cosinus de l'angle formé par les deux droites reliant d'une part le barycentre de tout les stops et le point de départ et d'autre part la droite reliant ce même barycentre au barycentre du cluster
— sinus_baryecentre_first_zi : sinus de l'angle formé par le même que celui qui a permis de calculé cosinus_barycentre_first_zi
— distane_barycentre_ind : numéro attribué à la zone_id en fonction de la valeur de distane_barycentre. 0 pour la zone_id ayant la plus petite, 1 pour la seconde et ainsi de suite.
— distane_barycentre_ind_norm : distane_barycentre_ind divisé par le nombre de zone_id dans la route

4 Variables sur les enveloppes concaves et convexes

— convex : nombre de stops dans la zone_id et sur l'enveloppe convexe formée par les stops de la route
— concave : nombre de stops dans la zone_id et sur l'enveloppe concave formée par les stops de la route
— in_convex : 1 si la zone_id est dans l'enveloppe convexe 0 sinon
— in_concave : 1 si la zone_id est dans l'enveloppe concave 0 sinon
— res_nb_convex : nombre de colis à livrer sur l'enveloppe convex dans la zone_id
— res_nb_concave : nombre de colis à livrer sur l'enveloppe concave dans la zone_id
— res_vol_convex : volume totale de colis à livrer sur l'enveloppe convexe dans la zone_id
— res_vol_concave : volume totale de colis à livrer sur l'enveloppe concave dans la zone_id
— res_dim_convex : volume du plus gros colis à livrer sur l'enveloppe convexe dans la zone_id
— res_dim_concave : volume du plus gros colis à livrer sur l'enveloppe concave dans la zone_id

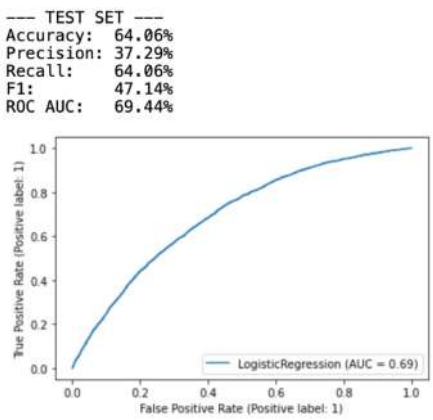
5 Variables sur les times windows

— time_first_window_inf : temps (en seconde) entre la plus petite borne inférieur des time windows imposées aux différents stops de la route et le departure time
— time_first_windows_sup : temps (en seconde entre) la plus petite borne supérieur des time windows imposés aux différents stops de la route et le departure time
— window_sup_km : distance (en Km) entre le baryencentre de la zone_id et le stops ayant la time windows avec la plus petite borne supérieur
— window_inf_km : distance (en Km) entre le baryencentre de la zone_id et le stops ayant la time windows avec la plus petite borne inférieur
— window_sup_time : temps (en seconde) entre le baryencentre de la zone_id et le stops ayant la time windows avec la plus petite borne supérieur
— window_inf_time : temps (en seconde) entre le baryencentre de la zone_id et le stops ayant la time windows avec la plus petite borne inférieur
— window_sup_km_ind : numéro attribué à chaque zone_id. 0 étant pour la zone_id ayant la plus petie window_sup_km, 1 la seconde plus petite et ainsi de suite
— window_inf_km_ind : numéro attribué à chaque zone_id. 0 étant pour la zone_id ayant la plus petie window_inf_km, 1 la seconde plus petite et ainsi de suite
— window_sup_time_ind : numéro attribué à chaque zone_id. 0 étant pour la zone_id ayant la plus petie window_sup_time, 1 la seconde plus petite et ainsi de suite
— window_inf_time_ind : numéro attribué à chaque zone_id. 0 étant pour la zone_id ayant la plus petie window_inf_time, 1 la seconde plus petite et ainsi de suite
— window_sup_km_ind_norm : window_sup_km_ind divisé par le nombre de zone_id dans la route

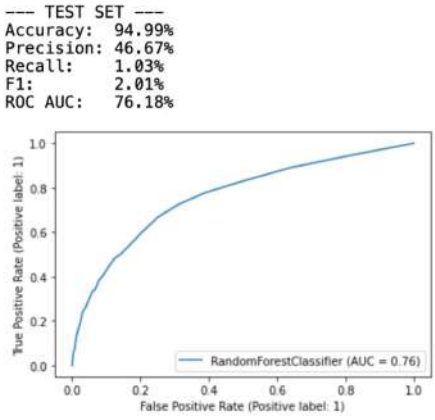
— window_inf_km_ind_norm : window_inf_km_ind_norm divisé par le nombre de zone_id dans la route
— window_sup_time_ind_norm : window_sup_time_ind_norm divisé par le nombre de zone_id dans la route
— window_inf_time_ind_norm : window_inf_time_ind_norm divisé par le nombre de zone_id dans la route

Annexe H

Performances de différents modèles de Machine Learning permettant de trouver la première zone_id à parcourir

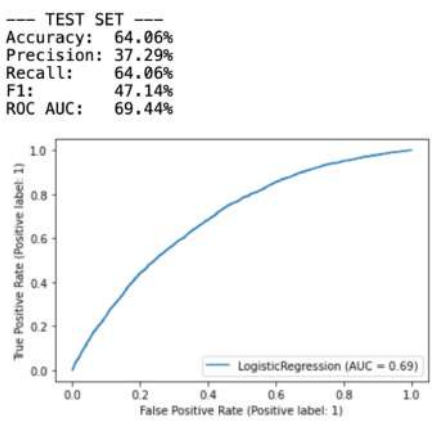


(a) Régression logistique pénalité L2 avec C :0.001

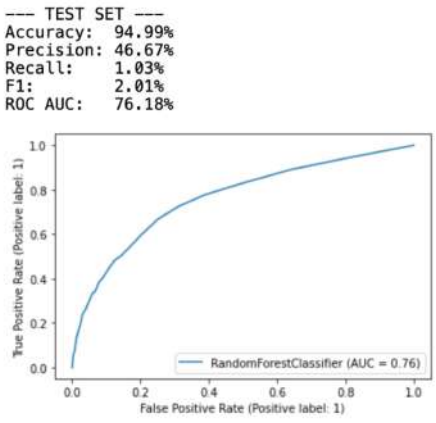


(b) Random Forest

FIGURE H.1 – Performances du premier modèle



(a) Régression logistique pénalité L2 avec C :0.001



(b) Random Forest

FIGURE H.2 – Performances du second modèle

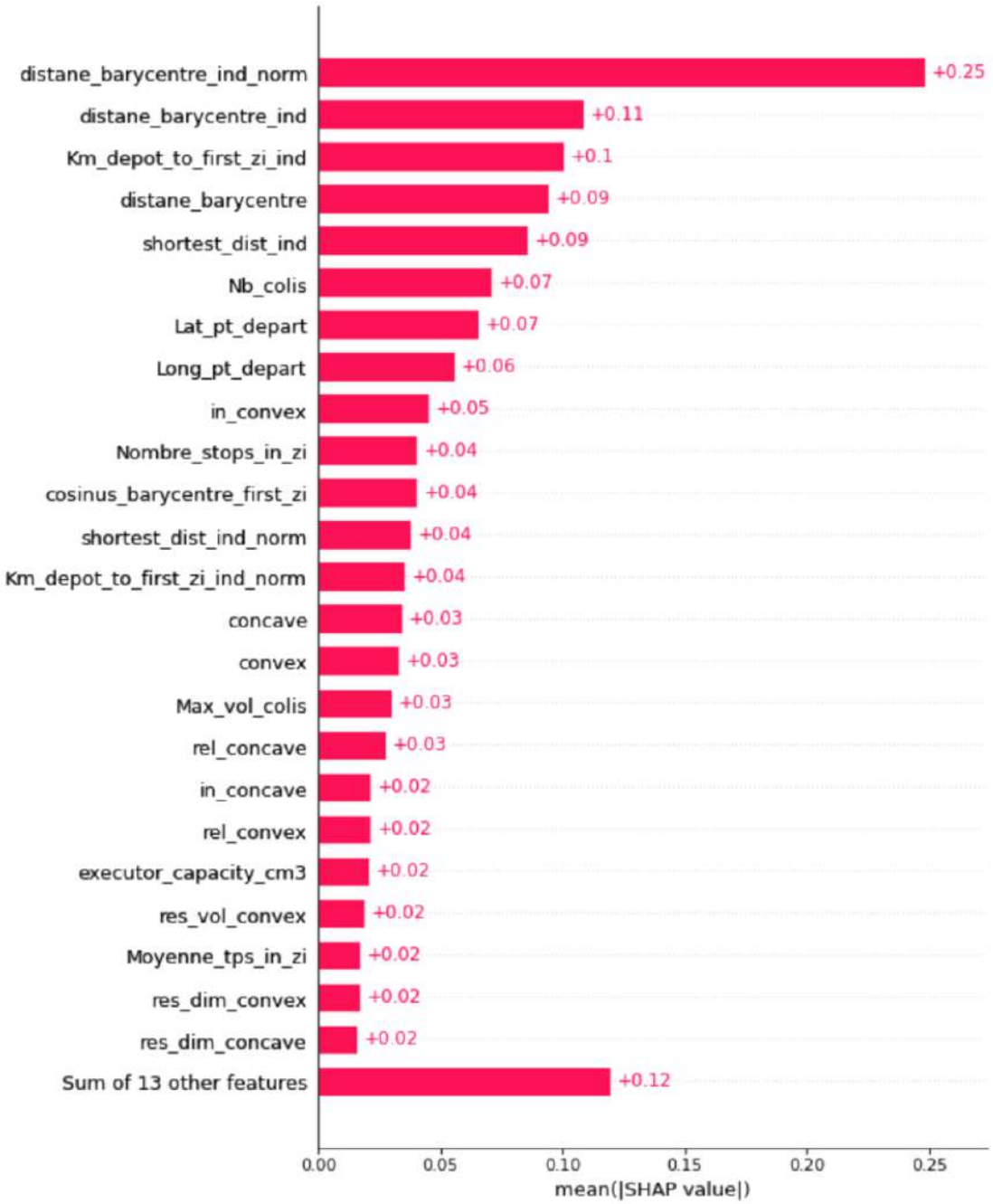


FIGURE H.3 – Features les plus importantes du second modèle pour le modèle avec régression logistique

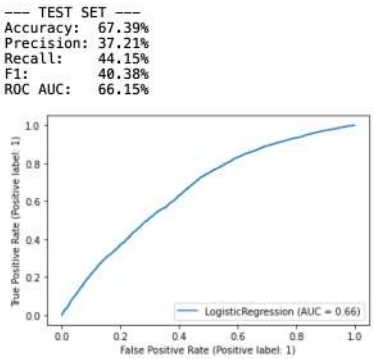
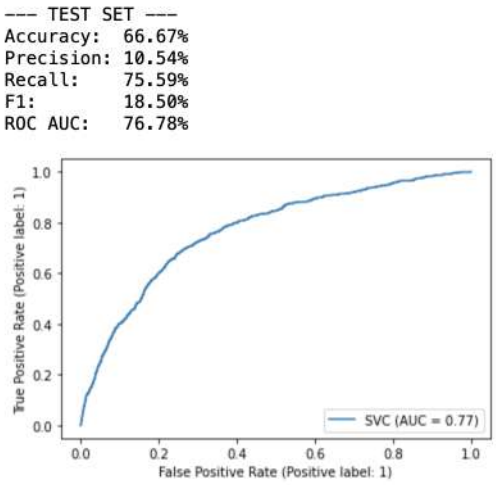
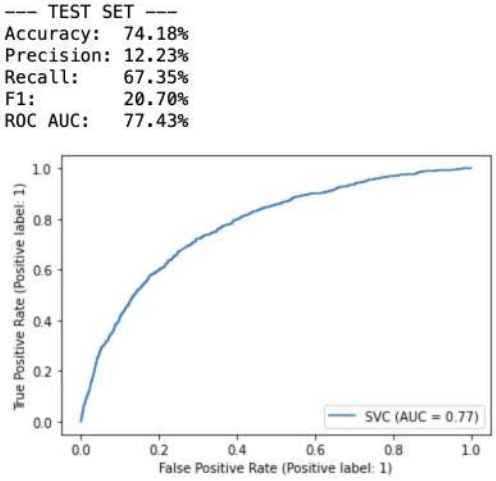


FIGURE H.4 – Performances du modèle utilisé pour la soumission du 18 juin

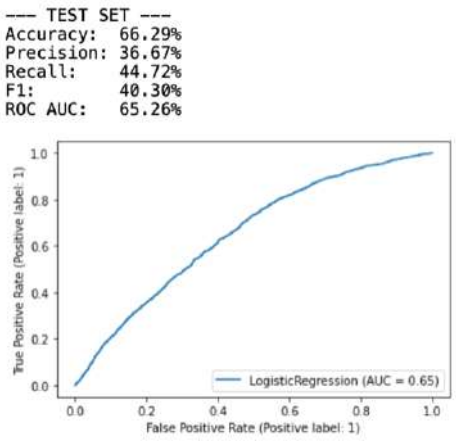


(a) Modèle avec 5 features (de la soumission)

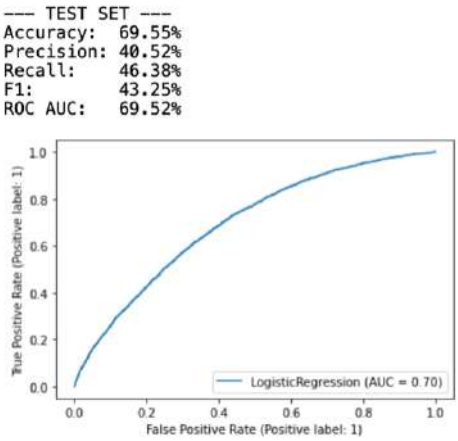


(b) Modèle avec toutes les varibales du second modèle

FIGURE H.5 – Résultat du modèle avec SVM pour trouver la première zone_id avec un nombre différents de varibales



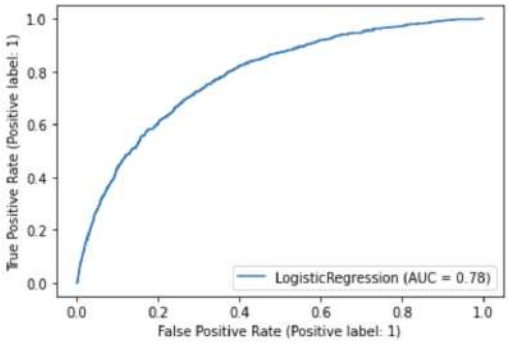
(a) Modèle avec toutes les features



(b) Modèle avec 5 features (de la soumission) et la feature window_inf_time_ind_norm

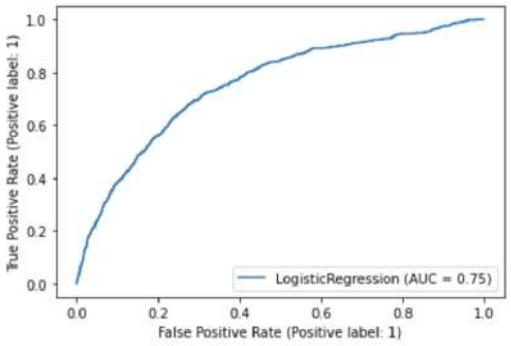
FIGURE H.6 – Résultat de différents modèles avec des features sur les time_windows pour trouver la première zone_id

```
--- TEST SET ---
Accuracy:  72.55%
Precision: 11.91%
Recall:    70.15%
F1:        20.37%
ROC AUC:   78.44%
```



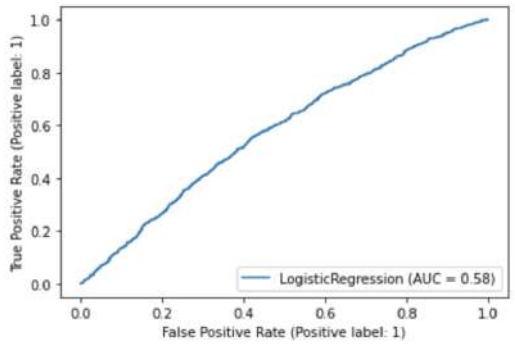
(a) La variable cible est la première zone_id

```
--- TEST SET ---
Accuracy:  70.87%
Precision: 11.16%
Recall:    69.26%
F1:        19.22%
ROC AUC:   75.45%
```



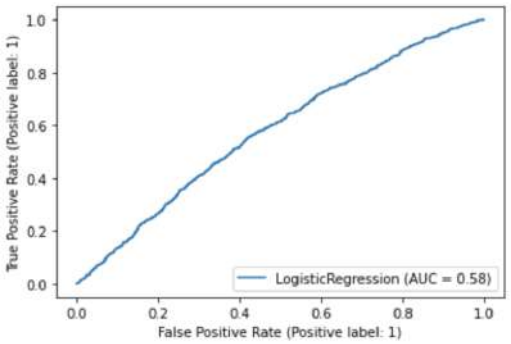
(b) La variable cible est la dernière zone_id

```
--- TEST SET ---
Accuracy:  54.23%
Precision: 6.25%
Recall:    58.24%
F1:        11.29%
ROC AUC:   58.00%
```



(c) La variable cible est la cinquième zone_id

```
--- TEST SET ---
Accuracy:  54.23%
Precision: 6.25%
Recall:    58.24%
F1:        11.29%
ROC AUC:   58.00%
```



(d) La variable cible est la première zone_id ou la dernière zone_id

FIGURE H.7 – Résultats de différents modèles de Machine Learning avec les même variables expliatives et des variables cibles différentes pour chaque modèle

Annexe I

Etude statistique

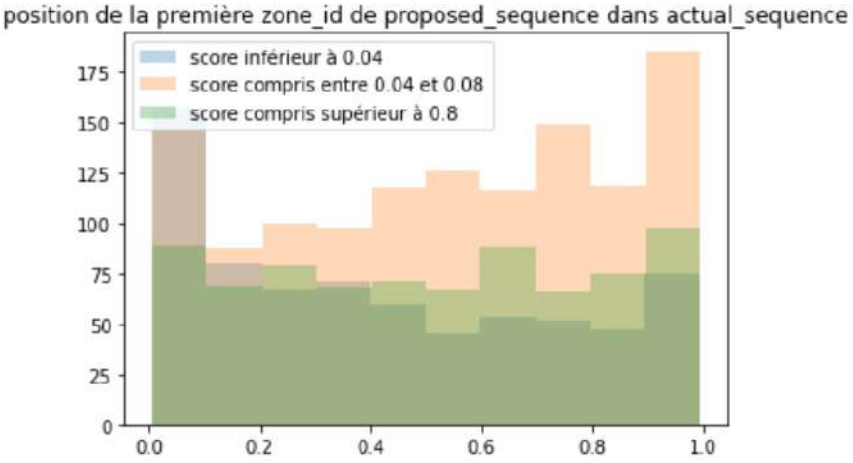
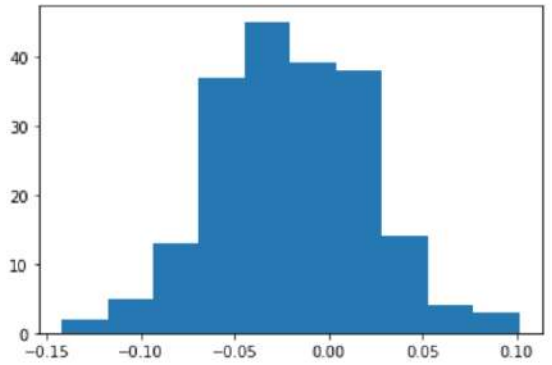
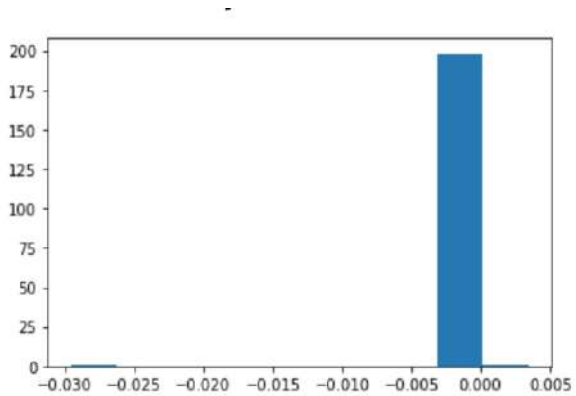


FIGURE I.1 – Histogramme de la position dans la séquence proposée par Amazon de la première zone_id visitée par notre algorihtme

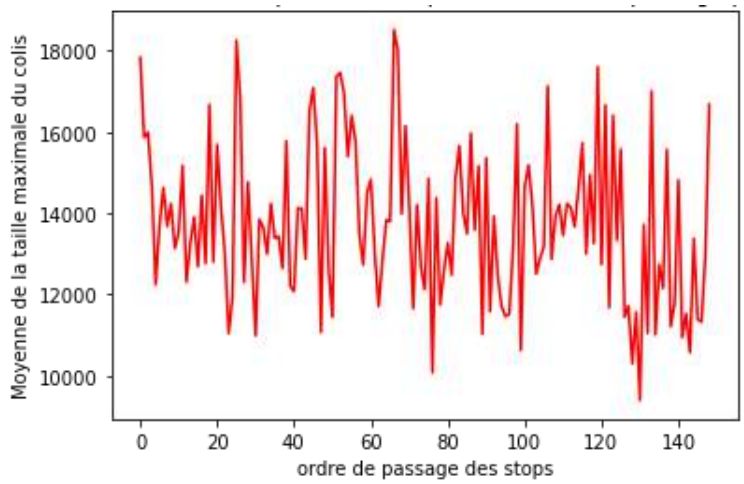


(a) Histogramme de la variable difference pour une inversion imposée, c'est à dire que la séqeunce retourné par l'algorithme est inversé par rapport à celle retrounée par le modèle

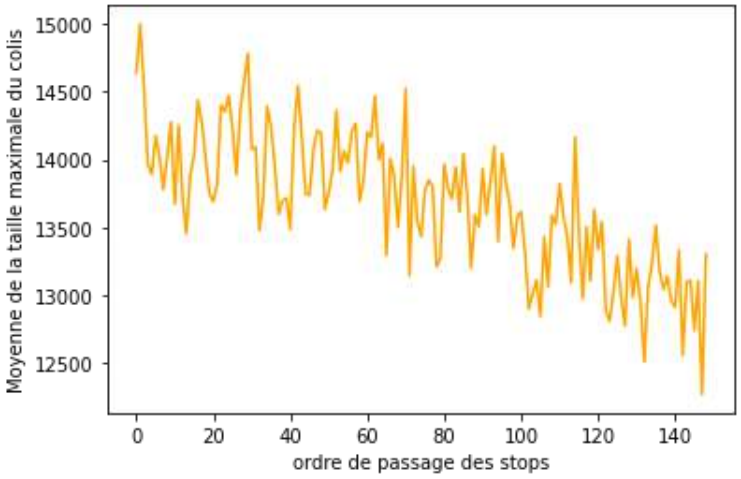


(b) Histogramme de la variable difference avec le post-processing

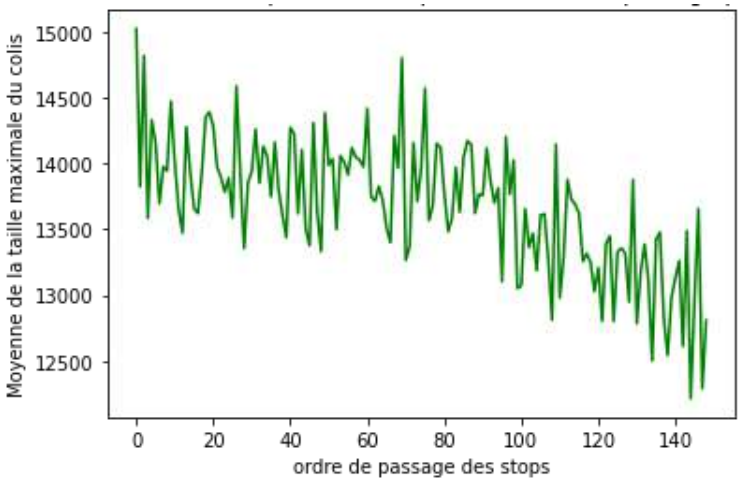
FIGURE I.2 – Histogrammes de la variables "différence" (elle mesure la différence entre le score fouri par l'agorithme avec différentes méthodes de post-processing et le score sans cette étape de post-processing) sur les 20% de routes label "High" ayant obtenue les pires scores



(a) Label ”Low”



(b) Label ”Medium”



(c) Label ”High”

FIGURE I.3 – Moyenne de la taille maximale du colis déposé à un stop selon l’ordre de passage de celui-ci pour des routes de différents label