NF26 - Data Warehouse et outils décisionnels

Printemps 2017 (P17)

Projet

# Modélisation, implémentation et analyse d'un service de taxis

**Maxime Churin GI04**

## Introduction

Le projet consiste, à partir d'un jeu de données conséquent : les trajets de taxis dans la ville de Porto au Portugal, à modéliser, concevoir, alimenter des tables et, enfin, de les analyser avec des algorithmes de *data-mining* comme les *k-means*. Pour se faire, on utilise un système de gestion de base de données (SGBD) de type NoSQL particulièrement adapté et performant pour des quantités de données importantes.

# 1 Le jeu de données

## 1.1 Description des données

Les trajets des taxis pour l'année 2015 à Porto nous sont donnés sous la forme d'un csv. Il s'agit d'un jeu de données réel ce qui est d'autant plus intéressant. Ce fichier regroupe plus de 1 700 000 de trajets décrient par un ensemble de 9 variables :

| Variable | Description |
|---|---|
| `TRIP_ID` | identifiant unique du trajet |
| `CALL_TYPE` | variable qualitative de la manière dont le taxi a été réservé |
| `ORIGIN_CALL` | identifiant du client du trajet |
| `ORIGIN_STAND` | identifiant de la station de taxi relative au point de départ du trajet |
| `TAXI_ID` | identifiant unique du chauffeur du taxi |
| `TIMESTAMP` | variable quantitative représentant la date où le trajet a commencé avec une précision à la seconde |
| `DAY_TYPE` | variable qualitative ajoutant de l'information sur le type de jour de départ du trajet |
| `MISSING_DATA` | variable qualitative pour décrire la présence/absence de données GPS manquantes |
| `POLYLINE` | liste de coordonnées GPS sous le format [LONGITUDE, LATITUDE], chaque paire de coordonnées représente 15 secondes de trajet |

**Table 1.1** – Tableau récapitulatif des différentes variables du jeu de données

## 1.2 Modélisation

Une fois les données assimilées, j'ai commencé à réfléchir à la modélisation. Le modèle en étoile, vu dans la première partie de l'UV, paraissait plutôt adapté. J'ai donc regroupé les différentes variables en 5 catégories/dimensions classiques : le Temps, le Vendeur, le Lieu, le Client et le Trajet. J'ai aussi réfléchi à des découpages hiérarchiques mais ils ne se justifiaient que pour la dimension Lieux et Temps. J'ai alors obtenu un UML qui se rapproche d'un UML classique en allant un petit peu plus loin. En effet, la suite du projet déjà en tête, il y a quelques modifications, comme la présence de redondance, et des implémentations pensées pour les requêtes et l'optimisation.

On observe que j'ai implémenté le type `geom` qui regroupe le couple latitude/longitude pour simplifier et clarifier la représentation mais aussi par souci de réutilisation car il se retrouve dans plusieurs classes. Pour la dimension Temps, je l'ai simplement découpée en hiérarchies classiques avec une précision de l'ordre de la seconde en rajoutant la variable `day_type}` au niveau du jour. Je souhaitais initialement faire une précision à la minute mais inclure les secondes m'a semblé intéressant en terme d'information considérant la faible différence de coût de stockage.

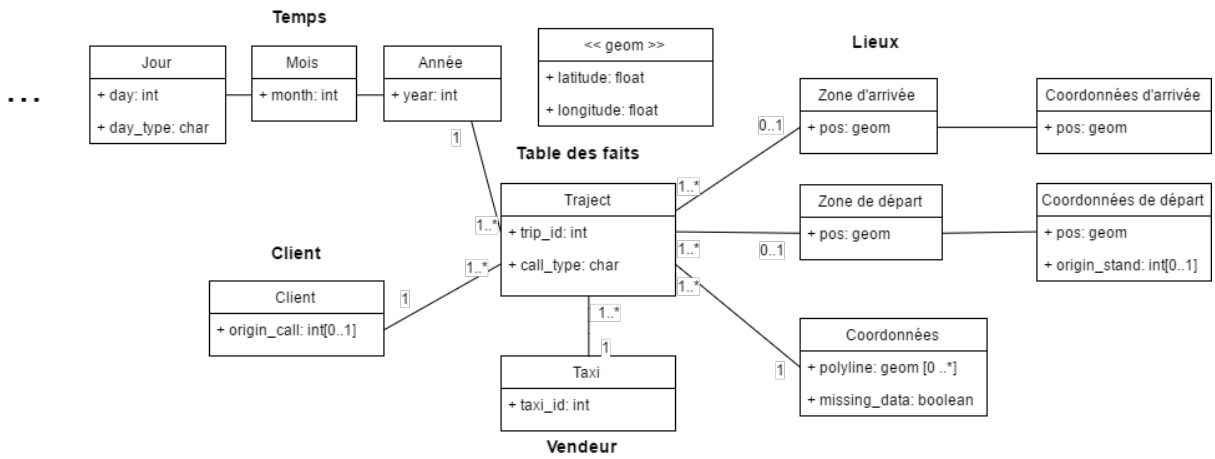



**Figure 1.1** – UML résultant de la modélisation des variables

Pendant cette modélisation, j'ai été confronté à deux grandes questions :

— Comment implémenter la dimension Lieu ?

— Où placer les attributs annexes : création d'une dimension ou stockage dans la table des faits ?

Pour la dimension Lieu, j'avais d'abord pensé à stocker uniquement les `polyline` et y faire des requêtes pour obtenir les positions de départ/arrivée. Cependant, cela ne m'a pas paru adéquat pour un jeu de données aussi grand. J'avais ensuite réfléchi à la possibilité de garder uniquement les positions de départ/arrivée, qui font sens à elles seules, mais on perd trop d'informations comme la distance du trajet ou sa durée. J'ai finalement opté pour une combinaison de ces deux solutions ce qui inclut de la redondance mais pour une meilleure performance. J'ai aussi créé la notion de zone de départ/arrivée pour pouvoir retrouver plus facilement des groupes de trajets et ajouté la variable `missing_data` dans ces coordonnées.

Pour les attributs restant, j'ai longuement hésité entre créer des dimensions et les inclure dans la table des faits. Mais, dans une logique de modélisation, les variables `origin_call` et `taxi_id` pouvant faire référence à plusieurs trajets j'ai créé de nouvelles classes. Finalement il ne restait plus que le `trip_id` et le `call_type` pour la table des faits.

# 2 Conception de l'architecture

## 2.1 Matérialisation

On s'intéresse maintenant aux attributs que l'on souhaite garder en ayant en tête les requêtes que nous souhaitons optimiser. J'ai donc décidé de ne pas stocker : `call_type, origin_stand`, `origin_call` et `missing_data`. Ces attributs, souvent null, n'apportent que peu d'informations et plutôt de type secondaire. Après une longue hésitation, j'ai décidé de tout de même matérialiser le type `geom`. Cela m'a permis d'apprendre la syntaxe de création et l'utilisation d'un type pour, entre autre, rendre la table plus lisible. J'ai aussi choisi de revenir sur mon implémentation UML en ne gardant finalement que la position et la zone de départ/arrivée de chaque trajet et non plus toutes les coordonnées. Une fois ce tri réalisé, il ne nous restait plus que les dimensions Temps et Lieu. On peut alors choisir de les matérialiser ou de les fusionner dans la table des faits. Le coût par requête avec des `JOIN` étant non négligeable, j'ai opté pour une fusion complète avec un seul modèle de table.

Dans `Cassandra`, et plus généralement en NoSQL, on duplique les tables en modifiant les clés de clustering et partitionnement qui oriente et optimise la table pour un type de requête spécifique. La classe Trajet constitue donc le modèle qui sera présent dans toutes les tables `Cassandra` car je n'ai pas jugé utile de retirer des attributs selon le type de requête. Les trois petits points représentent le reste des attributs hérités de la dimension Temps pour atteindre une précision à la seconde.

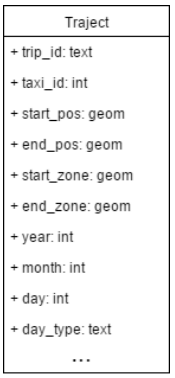



**Figure 2.1** – Patron de ma table modèle

J'ai ensuite listé les requêtes que je souhaitais optimiser en priorité :

— une requête sur la date du trajet
— une requête sur la zone de départ en fonction de l'année
— une requête sur la zone d'arrivée en fonction de l'année
— une requête sur la zone de départ et d'arrivée en fonction de l'année

J'insiste sur la notion d'année pour permettre une meilleure évolution de l'architecture au cours du temps en limitant le remplissage trop important des partitions correspondant aux zones. Il en découle donc 4 tables avec leur *primary_key*, celles-ci sont visibles dans le fichier *sql.txt* joint au rapport. Je précise à chaque fois le `trip_id` comme dernière clé de *clustering* car c'est le seul attribut unique du trajet.

## 2.2 Création et alimentation des tables

Une fois cette partie terminée, il a fallu s'attaquer au code permettant de se connecter à `Cassandra`, créer les tables et les alimenter à partir de notre csv. Cette partie a été réalisée en `Python`, un langage haut niveau très performant que je connaissais déjà. Les fonctions que je

vais brièvement aborder ici sont fournies dans le fichier annexe *functions.py*. J'ai donc créé des fonctions réutilisables me générant les requêtes de création et d'insertion à partir d'un dictionnaire. J'ai ajouté une subtilité pour gérer l'insertion du type `geom` à partir de ce dictionnaire. J'ai ensuite codé la fonction de lecture du fichier csv qui ne traite que les lignes cohérentes : `missing_data = False` et un vecteur de coordonnées non vide. Je réalise un traitement spécifique aux données stockées sous la forme de texte et aux données du type `geom` : dans le cas des zones j'arrondis au 1000ème. Cela représente des zones d'environ 100m² ce qui me semble un bon compromis et correspond à l'échelle d'une rue, d'un petit pavé de maison. Pour finir, j'utilise le fichier *script.py* pour appeler mes fonctions et réaliser mon alimentation. Comme j'aime avoir des retours lorsque je lance un script j'ai laissé de nombreux *print* d'informations dans le code.

## 2.3 Interrogation des tables

Pour réaliser les requêtes précédemment listées, j'ai codé un mécanisme d'interrogation fonctionnel. Il retourne la requête à lancer pour obtenir les données sélectionnées par les différents paramètres. Dans ces fonctions, il est obligatoire de préciser toutes les parties de la clé de partitionnement ce qui est représentatif de la réalité car, sans ça, la requête ne peut aboutir. On peut donc y préciser la liste des attributs à afficher ainsi qu'un dictionnaire pour les arguments facultatifs comme les éléments de la clé de *clustering* ou les autres éléments non clé de la table. Cependant, ces fonctions ne sont pas optimales, elles sont juste là pour montrer des exemples basiques d'interrogation. Les pistes d'améliorations seraient d'y inclure les différents types d'opérateur, des options comme `GROUP_BY, ORDER_BY` et, surtout, des systèmes de contrôle de validité des arguments et des attributs. On remarque aussi que de nombreuses choses pourraient se gérer à partir d'une interface et des formulaires adaptés. En effet, dans la réalité, il est rare et peu productif de laisser un utilisateur non initié faire des requêtes sur un SGBD. On lui fournit alors une interface qui va nous servir de filtre en passant les bons arguments pour générer la requête sans intervention de l'utilisateur.

# 3 Analyse des données

## 3.1 Méthode d'analyse

Dans cette dernière partie, nous allons essayer d'extraire de l'information de ce jeu de données. Il a alors fallu choisir quelles variables nous souhaitions analyser. Notre choix c'est naturellement porté sur les trajets et notamment les types de trajets les plus fréquents dans l'optique de les cartographier. Pour ce faire, nous avons utilisé des techniques d'analyses statistiques non-supervisées. J'ai choisi de réaliser un *k-means offline* adapté pour de grand jeux de données. Ce choix a été motivé par la nature de nos données qui sont en réalité des trajets et correspond à notre objectif de regrouper les trajets en K clusters plus ou moins homogène avec une distance minimale entre le cluster et ses trajets.

Pour implémenter cette algorithme des *k-means*, j'ai créé un objet `Centroid` en`Python` qui correspond au cluster. Il est composé d'un entier `K` permettant d'identifier la ligne de la table qui a servi à son initialisation, ses coordonnées de départ/arrivée et une liste des coordonnées qui correspondent aux différents trajets de distance minimale avec ce *cluster*. Le code est disponible dans le fichier annexe *functions.py* qui se lance à l'aide du script *exec_kmeans.py*. Mon algorithme est basé sur la logique suivante :

- on sélectionne au hasard `K` points parmi nos données
- on initialise nos `Centroid` avec ses `K` points
- Tant que la convergence n'est pas atteinte on répète :
  - sur toute la table affecter chaque trajet à son `Centroid` le plus proche
  - sur chaque `Centroid` recalculer les nouvelles coordonnées en faisant la moyenne de tous les trajets associés au `Centroid` puis vider la liste des points

Il ne restait plus qu'à choisir un nombre de cluster pour faire tourner l'algorithme sur notre table. Cependant, détermination n'est pas évidente. En effet, il existe différentes techniques qui permettent à partir du jeu de données de déterminer une valeur optimale, mais, dans notre cas, nous souhaitons orienter notre analyse en fonction du choix de l'utilisateur. J'ai donc codé ma fonction de manière à ce que ce `K` soit passé en paramètre de l'appel de l'algorithme.

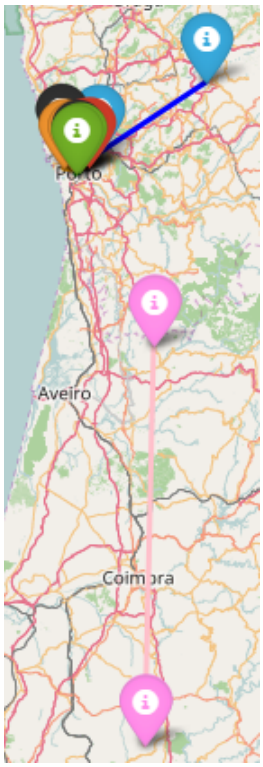
Pour finir, souhaitant visualiser le résultat de mon *k-means*, j'ai recherché des librairies de cartographie `Python`. Je suis alors tombé sur des documentations d'API comme Google Maps ou Here mais mon choix s'est plutôt porté sur Folium utilisant OpenStreetMap. Cette librairie ma séduite par sa simplicité, ses options de couleurs et la présence de marqueurs.

## 3.2 Présentation des résultats

J'ai donc commencé par faire tourner mon algorithme sur ma table *facts_by_zone* en prenant un nombre de clusters égale à 6. J'ai alors obtenu convergence après seulement 17 itérations. On observe ci-dessous que ma représentation est polluée par 2 clusters : le trajet bleue et le trajet rose. En effet, même si ils sont composés d'un petit nombre de trajets : moins de 3000 chacun soit 0.1% de l'effectif ; ils sont tellement différents qu'ils traduisent quand même un cluster. Le trajet rose, ne partant pas de Porto, est tout de même surprenant pour un service de taxis basé à Porto. On peut supposer la présence de données incohérentes. Pour le trajet bleue, c'est plus plausible car il correspond à un trajet de Porto à Felgueiras, une ville voisine plutôt importante.

```
Itération 17
Début analyse de la table
Centroid 479692 :
        Start position (latitude, longitude): 41.162659 -8.655098
        End position (latitude, longitude): 41.157375 -8.630948
        Number of points 318348

Centroid 1444128 :
        Start position (latitude, longitude): 41.195511 -8.554585
        End position (latitude, longitude): 41.360277 -8.221708
        Number of points 2415

Centroid 1060869 :
        Start position (latitude, longitude): 41.160770 -8.600302
        End position (latitude, longitude): 41.167368 -8.582263
        Number of points 362241

Centroid 1149141 :
        Start position (latitude, longitude): 41.152859 -8.609392
        End position (latitude, longitude): 41.151049 -8.618718
        Number of points 766712

Centroid 1431537 :
        Start position (latitude, longitude): 41.157782 -8.618667
        End position (latitude, longitude): 41.195148 -8.668316
        Number of points 254700

Centroid 197755 :
        Start position (latitude, longitude): 40.752726 -8.384290
        End position (latitude, longitude): 39.815244 -8.410438
        Number of points 329

Fin itération sur la table avec convergence = True
```

**(a)** Caractéristiques finales des clusters

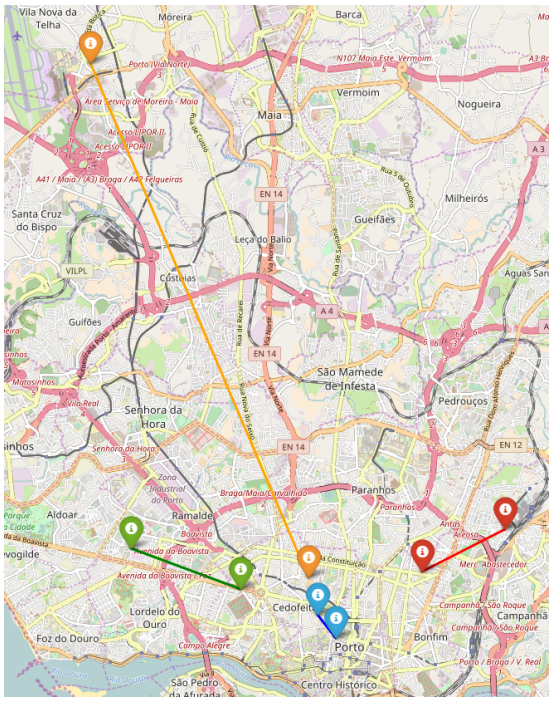



**(b)** Représentation cartographique des trajets

**Figure 3.1** – Résultats de l'algorithme pour $K = 6$

```
Itération 90
Début analyse de la table
Centroid 35215 :
        Start position (latitude, longitude): 41.160313 -8.619255
        End position (latitude, longitude): 41.240319 -8.664351
        Number of points 92859

Centroid 195166 :
        Start position (latitude, longitude): 41.154544 -8.617361
        End position (latitude, longitude): 41.150855 -8.613674
        Number of points 875066

Centroid 1319498 :
        Start position (latitude, longitude): 41.161256 -8.595747
        End position (latitude, longitude): 41.167786 -8.578614
        Number of points 323145

Centroid 1665207 :
        Start position (latitude, longitude): 41.158493 -8.633406
        End position (latitude, longitude): 41.164814 -8.656017
        Number of points 413675

Fin itération sur la table avec convergence = True
```

**(a)** Caractéristiques finales des clusters




**(b)** Représentation cartographique des trajets

**Figure 3.2** – Résultats de l'algorithme pour $K = 4$

Une solution consiste à ne plus considérer les *outliers* et donc retenter une analyse avec $K = 4$. L'algorithme fait alors plus de 90 itérations. On peut supposer qu'il arrive avec difficulté à convergence à cause de ces deux *outliers* et la présence de trajets concentrés dans une zone plus petite. On retrouve approximativement les 4 mêmes trajets dans Porto que sur la Figure 3.1 mais les *outliers* en moins. On remarque que le trajet orange mène à l'aéroport et que le trajet rouge mène à la gare. Ces résultat paraissent donc cohérent et, $K = 4$, un bon nombre de clusters.

## Conclusion

Ce projet, au départ un peu flou s'est peu à peu éclairci pour devenir très enrichissant. La première partie de modélisation est plutôt classique mais couplé à la matérialisation et au NoSQL elle en est bouleversé. Ça a été une réelle nouveauté pour moi de travailler avec cette technologie qui casse les codes classiques. Adepte du code propre et efficace, j'ai aussi aimé travailler en `Python` et réfléchir à l'implémentation d'algorithmes comme le *k-means*. Son analyse et la présentation des résultats m'a ensuite permis de vérifier la cohérence de mon travail mais, surtout, de prendre conscience de l'étendu du possible. On aurait pu réaliser des recherches sur les conducteurs de taxi, les durées des trajets ... qui sont toutes aussi intéressantes mais, par manque de temps, nous n'avons pas pu aller plus loin.