Apprentissage statistique : application au trafic routier à partir de données structurées et aux données massives

Brendan Guillouet

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)*

Présentée et soutenue le *18/11/2016* **par :**

BRENDAN GUILLOUET

Apprentissage statistique: application au trafic routier à partir de données structurées et aux données massives.

JURY

PHILIPPE BESSE	Professeur d'Université	Directeur de Thèse
CHRISTOPHE BIERNACKI	Professeur d'Université	Rapporteur
JEAN-MICHEL LOUBES	Professeur d'Université	Directeur de Thèse
FABIEN MOUTARDE	Professeur d'Université	Rapporteur
BRUNO PELLETIER	Professeur d'Université	Président du Jury
FRANÇOIS ROYER	CEO Datasio	Membre du Jury

École doctorale et spécialité :
MITT : Domaine Mathématiques : Mathématiques appliquées

Unité de Recherche :
Institut de Matématiques de Toulouse (UMR 5219)

Directeur(s) de Thèse :
Jean-Michel LOUBES et *Philippe BESSE*

Rapporteurs :
Christophe BIERNACKI et *Fabien MOUTARDE*

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)*

Présentée et soutenue le *18/11/2016* **par :**

BRENDAN GUILLOUET

Apprentissage statistique: application au trafic routier à partir de données structurées et aux données massives.

JURY

PHILIPPE BESSE	Professeur d'Université	Directeur de Thèse
CHRISTOPHE BIERNACKI	Professeur d'Université	Rapporteur
JEAN-MICHEL LOUBES	Professeur d'Université	Directeur de Thèse
FABIEN MOUTARDE	Professeur d'Université	Rapporteur
BRUNO PELLETIER	Professeur d'Université	Président du Jury
FRANÇOIS ROYER	CEO Datasio	Membre du Jury

École doctorale et spécialité :
MITT : Domaine Mathématiques : Mathématiques appliquées

Unité de Recherche :
Institut de Matématiques de Toulouse (UMR 5219)

Directeur(s) de Thèse :
Jean-Michel LOUBES et *Philippe BESSE*

Rapporteurs :
Christophe BIERNACKI et *Fabien MOUTARDE*

À Henri

À Simon

Remerciements

Je tiens à remercier, en premier lieu, mes directeurs de thèse, Jean-Michel et Philippe, pour avoir accepté de m'encadrer dans ce projet. Je vous remercie pour votre disponibilité. Je me suis senti parfaitement accompagné et soutenu pendant ces trois ans, le tout dans une ambiance de travail conviviale. On dit les bretons têtus, je crois que vous pouvez le confirmer, je vous remercie donc aussi pour votre patience et votre capacité à déchiffrer mes explications. Merci à Jean-Michel pour ton enthousiasme, tes nombreuses idées et pour m'avoir régulièrement (re)motivé et permis de croire en la qualité de mon travail. Merci à Philippe pour ton écoute, ta pédagogie, et ta capacité à être toujours à la page concernant les nouvelles technologies.

Je remercie très sincèrement, Fabien Moutarde et Christophe Biernacki, pour avoir accepté d'être les rapporteurs de ma thèse et pour leurs précieux conseils. Je remercie Bruno Pelletier pour avoir accepté de présider ce jury.

Je remercie François, pour m'avoir proposé ce sujet. Je te remercie pour la confiance que tu m'as accordée et la liberté d'action que tu m'as laissée. Chaque thèse est unique, mais celle-ci a été particulièrement riche en enseignements, de par la diversité des situations rencontrées.

Je souhaite exprimer ma très sincère gratitude à Susan, pour ces centaines d'heures passées à corriger un anglais parfois approximatif. Merci aussi à Frédéric pour cela. Je ne sais pas où en seraient mes papiers et ma thèse sans cette aide. Je te remercie également pour m'avoir permis de dépasser mes préjugés en ce qui concerne mon rapport à la race canine.

J'en profite aussi pour remercier les autres membres des bureaux que j'ai pu côtoyer dans les différents open-spaces. Merci à Julien pour sa bonne humeur, ses conseils et ses trolls, merci également à Maxime, Arnaud, Guillaume, Lionel, Julien, Aurélien, Xavier et Yohan. J'ai toujours eu la chance de travailler dans un environnement à la fois agréable et productif, et je vous en remercie. Je n'oublie pas non plus les collègues du foot, qui ont grandement contribué à lâcher la pression et à une aération d'esprit, très souvent nécessaire.

Étant thésard CIFRE, ma présence à la fac a été pour le moins sporadique. Cependant je tiens à remercier très sincèrement tous les thésards que j'ai croisés. En tant qu' "expat", j'ai toujours eu un très bon accueil et une aide à chaque fois apportée avec

plaisir. Dans le désordre, Sylvain, Clément, Anne-Claire, Mélanie, Anton, Waly, Tatiana, Fanny, Stéphane, Claire, Malika, Benoit, et tous ceux que j'oublie. Je tiens à remercier Claire, ma première co-bureau, pour m'avoir soutenu dans mes premières années, pour ses nombreux conseils et sa franchise. Et enfin, un grand merci à Sofiane, pour m'avoir très souvent écouté. Tes nombreux conseils saupoudrés de (petites) vanneries offrent un savoureux cocktail que je te souhaite de continuer à entretenir. J'en profite pour remercier Anna et Aurore pour leurs encouragements respectivement, en début de thèse et lors de ma première conférence.

Je remercie Anne et Nathalie, pour m'avoir accompagné lors de mes premiers pas dans la recherche, et la vision positive et stimulante de celle-ci qu'elles m'ont transmise.

Trois années de thèse, c'est avant tout trois années de vie. J'ai eu la chance de pouvoir compter sur de nombreuses personnes qui m'ont permis de trouver un équilibre durant cette période et de garder la tête sur les épaules. Je tiens à les remercier ici.

Je me devais de commencer par ces bon vieux gypsies. Merci Nico pour les heures de debrief, ton hyperactivité et tes Dufillages. Merci Yannig pour tous ces talents que tu nous fais partager, tes corrections syntaxiques et tes beats enflammés. Merci Aude pour toutes ces séances à refaire le monde et cette volonté que tu as de trouver un (bon) sens à chaque chose. Et bien sur, merci à toi Simon pour toutes ces histoires insensées et tous ces souvenirs et enseignements que tu nous laisses. Le bon vieux GS sait qu'il vous doit énormément et vous remercie d'avoir rendu cette thèse un peu plus... piquante. Ces 3 années de thèse n'ont pas toujours été faciles, mais je retiens les magnifiques moments passés en votre présence. En somme, je me permettrai de les résumer ici à de l'excellent boulot. Un merci tout particulier à Aude pour ton inestimable aide à la relecture finale ! J'en profite pour remercier ici Betty, Laura, Jean-Lou, Carmen, Robin, Servan, Tristan, Thomas, Laetitia, Victor et Juliette pour les nombreux beaux moments passés ensemble et ceux, tout aussi nombreux, à venir.

Cette thèse à aussi été une histoire de coloc' ! Merci à Quentin pour tous ses jeux de mots, à Alexandre car j'entends encore son onde sensuelle, à Julien pour la touche hypsterisante et à Marion pour tout ce qu'elle préfère et ce qu'elle ne préfère pas. Merci aux 'cooloks', Méderic pour ton état d'esprit en général et tous nos débats et Caro, pour avoir tant enrichi notre culture, particulièrement en ce qui concerne notre vocabulaire. Merci Antoine, mon double coloc', pour ta sérénité à toute épreuve et pour l'ambiance de travail, disons décontractée, à défaut d'être toujours productive et merci à Latufa pour ses attentions, ses rires et sa grande capacité à ne pas juger. Enfin, merci à toi Henri, pour tout ce que tu m'as apporté, et pour ton éternelle gentillesse. Merci à vous tous de m'avoir supporté au jour le jour.

Merci à toi Anaïs, mon al-thésard-égo, pour le timing de la rencontre et pour avoir partagé ma vision de la vie doctorante. Merci pour les séances "défouloirs", les après-midis thé(s)-banofee(s), tes névroses et le reste. Nul doute que tout cela est parti pour durée, une fois ces thèses finies. Merci également à Virginie, Florine et Silvia pour toutes ces

soirées insensées.

Un petit mot pour les anciens INSAiens, qui, pour une raison que je ne m'explique toujours pas, ont préféré le stress parisien à la vie du sud. Merci Victor pour ces échanges toujours si inspirants. Merci Tiphaine pour tes innombrables petites attentions. Merci Sam pour ta joie si communicative. Même si la distance réduit forcément la communication, je sais que je peux compter sur vous, et c'est un bonheur de vous savoir près de moi.

Merci aux Docteurs Quentin & Morgane d'être un si beau couple, pour leurs bienveillances, pour votre accueil hebdomadaire lors des vidéo-conférences GOT et les débats hautement philosophiques et essentiels qui ont suivis. Merci également à Audrey, Dorine, Bertrand et Sourinia pour ces moments d'apaisement passés avec vous.

Un grand merci à Christophe car rares sont les personnes qui savent vous bousculer et vous mettre face à vous même. #notroll. J'espère juste que tu sauras améliorer un peu tes goûts musicaux et ta culture underground dans le futur.

Merci Mathilde, d'être toujours si disponible, pour ton écoute et pour le repère que tu es pour moi. Merci Clémence, pour ton authenticité et pour toutes tes blagues d'un niveau encore inégalées. Merci à vous deux être aussi présentes depuis tant d'années.

Un grand merci à mes partenaires de planches. Dans l'ordre des numéros : Hassan, Estelle, Matthieu, Magalie, Laurie, Aurélie, Marion, Michèle, Cédric, Samy et Émeline. Merci pour m'avoir permis de m'évader, au moins une fois par semaine, merci pour cette belle aventure, qui n'en est qu'à son commencement. Je n'oublie pas notre mentore Dédeine pour son inspiration, qui ne se limite pas qu'à l'univers théâtral.

Merci à Solène pour ton inépuisable énergie que je n'abdique pas à vaincre un jour, à Inès pour ce qu'elle me pousse à accomplir et pour ces cinq secondes qu'elle m'a laissées à moi et à mon égo, à Gaby pour son talent d'ubiquité et pour cette dette qu'il a réglée avec brio, à Jessica pour ses précieux conseils, à Baptiste qui me montre avec tant d'investissement la voie à suivre, à Cyril pour l'exemple de volonté qu'il démontre chaque jour, à Oriane pour avoir fait le bon choix de venir à Toulouse. Thank you at Oli and Meli and Tami and Oh ! Merci pour ces trop rares mais si précieux moments passés avec vous. Merci à Camille et Simon pour les valeurs qu'ils transmettent et pour leurs talents. Merci à Pierre, Thomas, Tony, Anthony et Dimitri pour être toujours fidèles au poste. Je remercie également Fanny, Allan, Xavier, Vanessa, Edith, Morgan, Julien et Paul pour les moments partagés.

Merci à vous pour tout ça, et pour tout ce que je n'ai pas su transposer par écrit.

Merci à mon extraordinaire famille, pour cette solidarité sans faille et pour tout l'amour qu'ils m'ont donné depuis tant d'années. Merci d'avoir été là et d'avoir cru en moi. Merci Papa, merci Maman, pour toutes ces bonnes valeurs que vous m'avez transmises et tout ce que vous avez accompli, merci Gwennoline d'être venue me soutenir à Toulouse, pour ton humour mais aussi, pour tous ces bons moments autour d'un café, d'une (d'une ?) bière ou d'une assiette veggie, et merci Matilin, pour tous ces échanges inspirants et pour toutes tes anecdotes si enrichissantes. Vous êtes formidables. Je n'ai pas la place pour

nommer tous mes cousins, oncles et tantes ici, mais j'ai aussi une pensée pour eux, ainsi que pour mes grand-parents et notamment mes deux mamies qui, j'en suis sûr, sont très fières de moi, où qu'elles soient.

Enfin, merci à toi Trini d'être arrivée dans ma vie et tout ce que tu m'apportes depuis. Merci pour ta sincérité, ton affection, ta philosophie de vie et ta façon de voir les choses. Merci d'être une si belle personne et de tout rendre si facile. Merci de m'avoir à ce point soutenu et cru en moi. Gracias por cuidarme. Merci pour ton humour. Enfin, merci de me permettre de regarder vers l'avant avec sérénité.

Table des matières

Table des figures

Liste des tableaux

Introduction

This thesis is the result of a Ph.D. study carried out in the framework of a CIFRE industrial training and research agreement between the Datasio company and the University of Toulouse, Laboratory of Statistics and Probabilities. The aim of this work, naturally influenced by these dual auspices, is divided into two studies focusing on machine learning. The first study deals with the development of urban traffic prediction methods using taxi GPS data, with the purpose of improving road traffic flow monitoring. The fundamental results achieved led to the publication of a scientific paper in the *IEEE Transactions on Intelligent Transportation Systems* journal (Chapter 1), and a second article is under review by this same journal (Chapter 2). In the future, it would be interesting to explore possible industrial applications of this work. The second study (Chapter 3) focuses on a comparison of different technologies enabling the application of machine learning methods to massive volumes of industrial data. This comparison is based on lessons drawn from contracted client studies realised for Datasio. As the company's original client data is strictly confidential, the comparisons were made on datasets whose size and characteristics correspond to those of the original. This work will be published as part of the book [Besse P., 2016]. This thesis is written in the form of publications and is organised as follows :

In Chapter 1, we tackle the issue of finding the main paths taken by road network users. By looking for the most suitable method for clustering vehicle trajectories. We first provide a thorough review of the major existing clustering methods, before studying the relevant literature dealing with their application to trajectories. This investigation leads us to use distance-based clustering methods, which appear to be the most appropriate for reaching our objectives. We extend this approach by analyzing the different existing distances between trajectories and we propose a new distance which addresses the limitations of these distances. A hierarchical clustering is then applied on this distance, producing the partitioning of a set of trajectories and thus enabling the capture of the main behaviour pattern of the drivers. Finally, we produced experimental results comparing the different distances showing that our distance produces the best clustering.

In Chapter 2, we present the model we developed to capture user movement, and the different applications we derived from this model. We modeled each cluster of trajectories obtained in the previous chapter with a Gaussian mixture model. This enables us to

describe the space with a set of Gaussian densities that can be interpreted as states. We can therefore describe a trajectory as a succession of states instead of a succession of locations. Moreover, the evaluation of a Gaussian density function is simple and thus enables quick conversion of the trajectory to its state representation, as well as quick attribution for a new trajectory to the trajectory cluster where it most likely belongs. Hence this model enables us to learn the way users are moving in space and to extract characteristics of trajectories that can be used as features for predictive application. To prove that, we developed three methods in order to predict the final time and location of a trip, and its next location. The experimental results confirmed that our model enables us to describe the way the users are moving within each cluster of trajectories and that it is an excellent tool for predictive application.

The second problem was to determine the best environment for the application of machine learning methods on massive amounts of data. Today, technology produces an enormous amount of data. Accordingly, numerous tools have been developed to manage and analyse this volume of data. In the context of machine learning, the objective is to use a large amount of data in order to furnish the algorithm with a maximum amount of information, and all this, within a reasonable execution time. The model and the predictive applications presented within the first chapters were developed using the *Python* programming language and especially the machine learning algorithms within the *Scikit-Learn* library. We wanted to compare the performance of these machine learning algorithm implementations, to those implemented with technologies especially designed to handle massive amounts of data, such as the cluster computing framework, *Spark*. However, we decided against attempting this performance comparison on the machine learning algorithms we established within the predictive application presented in the first chapter. The data transformations required of the original taxi trajectory data were too complex to make the sought after comparison reliable, and thus we decided to retain the use of the most well known machine learning algorithms already implemented and optimised in the python libraries that we used. We did however, have the opportunity to pursue the comparison study as part of several industrial projects undertaken for *Datasio* clients on data other than our taxi data. In line with the goals of our second problem, it is of major importance for companies to find the best solution for applying machine learning algorithms on large data volumes, in a reasonable amount of time and with good performance. We therefore present in Chapter 3 the lessons drawn from three different use cases encountered during Datasio client projects. These three use cases concern the application of algorithms (Linear Regression, Random Forest, NMF) on various type of data (textual, images, ratings of movies). For confidentiality reasons, equivalent neutral datasets were used in our research. We then compared the implementations of the studied algorithms in three different technologies, *R*, *Python* and *Spark*, in terms of computation time and performance, leading to the conclusion that the machine learning tools in Spark are currently immature and their use should be undertaken with caution.

Résumé

La *découverte de connaissances géographiques* (*Geographic knowledge discovery*) a émergé ces dernières années comme un des axes centraux de la recherche scientifique. L'étude et la compréhension de comportements liés à la position géographique et le déplacement dans l'espace d'individus sont des enjeux majeurs dans de nombreux domaines d'applications : compréhension des flux de migration humaine ou animale, détection des comportements anormaux pour les systèmes de vidéo surveillance, évitement des collisions en robotique ou encore compréhension du comportement des consommateurs dans des centres commerciaux. Cette liste est non exhaustive et il existe de nombreuses autres applications pour lesquelles l'étude de données géolocalisées est nécessaire.

Nous nous concentrons dans cette thèse sur un sujet spécifique car basé sur des données géolocalisées publiques donc facilement accessibles et publiables. L'étude porte sur des trajets de taxis de différentes métropoles. L'étude et la compréhension du comportement des utilisateurs du réseau routier urbain recèlent de multiples enjeux tels que l'organisation du système de transports en commun d'une ville, la localisation des zones où des nouveaux services sont nécessaires, l'anticipation de problèmes de congestion du trafic, *etc...* En ce qui concerne les taxis et autres services de transports personnalisés, la prédiction de la destination et du temps d'arrivée des véhicules lors d'une course représente une information centrale. En effet, il est primordial de pouvoir anticiper où et quand un véhicule va finir sa course afin d'optimiser la distribution des futures courses. Ce problème a notamment été le sujet de concours Kaggle : "ECML/PKDD 15 : Taxi Trajectory Prediction (I)" et "ECML/PKDD 15 : Taxi Trip Time Prediction (II)" [kaggle, 2015]

Ces problèmes existent depuis un certain temps. Cependant l'émergence des nouvelles technologies apporte aujourd'hui des solutions nouvelles afin de les résoudre, notamment grâce à l'explosion du volume de données numériques. A titre d'exemple, le nombre de smartphones utilisés dans le monde était estimé à 2 milliards à la fin de l'année 2015. La plupart des ces appareils étant équipés de GPS ou autres systèmes de localisation, ils représentent une base de données considérable sur les déplacements humains, et sont le socle de nombreux travaux de recherche visant à la résolution des problématiques décrites ci-dessus. Ensuite, le coût de stockage des ces données à considérablement chuté ces dernières années. Si le prix d'un gigabyte était estimé à 100$ en 1997, il était dix

fois plus faible en 2000 et mille fois plus faible en 2010, facilitant le stockage, le partage et l'exploitation des données. Enfin la puissance de calcul est désormais accessible au grand public, permettant l'exécution d'algorithmes complexes sur les données, en particulier en ce qui concerne la classification supervisée et non-supervisée de ces données. Ces algorithmes sont aujourd'hui divers et variés et permettent de répondre à de nombreuses problématiques. Dans la dernière partie de cette thèse nous abordons le problème suivant : comment pouvons-nous utiliser ces avancées, à la fois en termes technologiques de puissance de calcul et algorithmiques à travers la diversité des solutions existantes pour traiter et analyser des données géolocalisées ? Nous faisons face à deux problèmes pour résoudre cette question.

- Premièrement, la plupart des algorithmes appliqués à la classification (supervisée ou non) nécessitent des données dans un format particulier : les individus doivent être de même dimension pour pouvoir être comparés les uns aux autres. Cette condition est limitante dans le cas des données géolocalisées. En effet, et nous le verrons plus en détail par la suite, ces données n'ont de valeur que si l'on considère leur aspect temporel. Lorsqu'un trajet est en cours de réalisation, il fournit des informations sur sa localisation de manière progressive, et en nombre indéterminé à l'avance. Comment pouvons nous alors appliquer des algorithmes sur ce type de données ? Quelles transformations doivent être appliquées sur ces données, issues d'observations d'un véhicule en mouvement, afin de les rendre exploitables et d'en extraire les bonnes informations ?

- Secondement, nous traitons, tout au long de cette thèse, des volumes de données considérables. Quelles sont alors les outils technologiques les plus adaptés pour la gestion de tels volumes de données ?

Le but principal de cette thèse est de construire un outil capable de résoudre la première problématique. Comment structurer les données géolocalisées issues de trajectoires afin de pouvoir en extraire les informations utiles à la résolution de problèmes de prédiction et/ou d'analyse ? Dans le challenge Kaggle [kaggle, 2015], par exemple, de nombreux algorithmes ont été proposés afin de résoudre un problème de prédiction de la destination d'un taxi au cours de son trajet. La solution victorieuse, [de Brébisson et al., 2015] utilise des réseaux de neurones dont l'apprentissage est effectué sur des vecteurs de tailles fixes. Chaque vecteur correspond à une trajectoire du jeu d'apprentissage et est constitué par les premières localisations du trajet et par des variables contextuelles. Cependant, cette solution possède une limitation majeure. Elle a été développée dans le but de prédire les destinations finales de véhicules à un instant t de leurs trajets où ils avaient atteint un point précis de leur parcours. Les vecteurs correspondant aux trajectoires du jeu d'apprentissage ont été construits en ce sens. Ainsi, si de nouvelles informations sur le trajet apparaissent au cours du temps, cela implique de reconstruire les vecteurs, et d'effectuer

un nouvel apprentissage afin d'intégrer ces informations à la prédiction. Ceci empêche d'étendre l'application aux cas pour lesquels les informations doivent être intégrées en temps réel afin d'améliorer la qualité de la prévision.

Pour pallier à ce problème, nous avons établi une méthode *data-driven*, construite à partir des caractéristiques des données de géolocalisation, qui permettent de capturer les informations sur la façon dont se déplacent les véhicules. Cette méthode a été développée à partir de jeux de données géolocalisées issues de trajets de taxis. Cependant, comme cette méthode est *data-driven*, elle ne requiert pas d'informations sur le réseau routier sous-jacent et peut ainsi être appliquée à tout type de trajectoire (bateaux, avions, tortues, *etc*...) . Cette méthode permet de structurer les données de façon à les rendre aisément manipulables et d'en extraire les informations nécessaires à la résolution de différentes problématiques et ce, indépendamment des caractéristiques des données à prédire. Dans la Section I nous expliquons comment nous avons obtenu cette nouvelle structure. Nous présentons également des applications de prédiction, définies à partir de cette structure, et les résultats de ces applications sur différents jeux de données.

Le second problème consiste à déterminer les technologies les plus adaptées pour traiter ces volumes de données massives et y appliquer les meilleures techniques d'apprentissage. Les applications de prévision établies dans la première partie de cette thèse ont été développées avec le langage de programmation *Python* et, en particulier, à l'aide des algorithmes d'apprentissage statistique de la librairie *Scikit-Learn*. Nous avons souhaité comparer la performance de ces algorithmes aux performances d'algorithmes implémentés avec des technologies spécialement conçues pour gérer de grands volumes de données, tels que *Spark*. Cependant, cette comparaison n'a pas été effectuée directement sur nos applications de prévision. Afin de rendre cette comparaison plus fiable, nous avons dans un premier temps décidé de nous concentrer sur des algorithmes d'apprentissage déjà implémentés et bien optimisés dans les différentes librairies de science des données. De plus, dans le cadre de prestations contractuelles au sein de l'entreprise Datasio pour différents clients, nous avons eu à appliquer diverses méthodes d'apprentissage connues. Ces expériences, effectuées sur des types de données variées, nous ont permis de tirer des conclusions sur l'état de l'art actuel des différentes mises en oeuvres de techniques d'apprentissage face à des données massives. Il est en effet très important, pour une entreprise, de sélectionner les solutions qui produisent les meilleurs résultats avec un temps de calcul acceptable. Pour des raisons évidentes de confidentialité, nous ne présenterons pas les cas d'applications exactes traités dans le cadre des prestations contractuelles en Datasio et ses clients. Cependant, nous avons effectué une comparaison sur trois cas d'applications à l'échelle industrielle, nécessitant la mise en œuvre de trois techniques d'apprentissage différentes (*Régressions linéaires*, *forêts aléatoire*, *NMF*), entre trois environnements technologiques : deux langages de programmation, *Python* et *R* et une structure logicielle de calculs distribués, *Spark*. Les résultats de cette comparaison sont présentés Section II.

I Données géolocalisées

Par commodité, les données analysées dans cette thèse sont des localisations GPS de taxis mais les méthodes développées s'applique à tout type de données.

I.1 La structure des données

Des données GPS sont enregistrées sous forme de vecteurs dans $\mathbb{R}^4$, (lon, lat, t, i), où t est le temps auquel l'observation a été réalisée, lon et lat sont respectivement la longitude et la latitude auxquelles le taxi i a été observé au temps t.

Pré-traitement des données

Le pré-traitement des données, ou *preprocessing*, est une étape primordiale de l'exploitation des données, qui consiste à transformer des données dans une forme appropriée à leur exploitation. Dans le cas des données géolocalisées, chaque observation prise individuellement ne contient que peu d'informations. Au contraire, le niveau d'information apportée par ces données augmente si l'on sait pour chaque observation par quel taxi elle a été produite et quelles sont ses observations précédentes et suivantes. L'étape de pré-traitement des données consiste alors à convertir les données brutes, un ensemble de vecteurs dans $\mathbb{R}^4$, en ensemble de trajectoires.

Definition 1. *Trajectoire Continue. Une trajectoire continue, T_c, est une fonction définie sur un intervalle de temps fixe, $\Delta^j = [t_s^j, t_e^j]$. Pour chaque temps t au sein de cet intervalle, cette fonction retourne la localisation correspondante. Elle est définie ainsi :*

$$\begin{array}{rccc} T_c & : & \mathbb{R}^+ & \rightarrow & \mathbb{R}^n \\ & & t & \mapsto & T_c(t) \end{array} .$$

Notation : $|\Delta^j| = t_e^j - t_s^j$ est la durée de la trajectoire et $n_c^j = \int_{t=t_s^j}^{t_e^j} T_c^j(t)dt$ est la longueur de la trajectoire continue T^c.

Ici, la localisation est $T_c(t) \in \mathbb{R}^2$ et correspond à la longitude et la latitude du taxi observé au temps t. Les jeux de données étudiés ne contiennent pas les localisations des taxis à chaque temps t car les observations sont produites à intervalle de temps régulier. La localisation exacte des taxis entre chaque intervalle de temps est donc inconnue. Nous considérons ainsi les trajectoires comme une séquence d'observation pour un même taxi i dans un intervalle de temps fixé.

Definition 2. *Trajectoire Discrète. Une trajectoire discrète, T^i, est définie ainsi :*

$$T^i : ((T_1^i, t_1^i), \ldots, (T_{n^i}^i, T_{n^i}^i)),$$

où $T_k^i \in \mathbb{R}^n$ est la taille de la trajectoire T^j, et n^i est le nombre d'observations dans T^i.

Dans la suite de nos travaux, nous considérons également la notion de trajectoire linéaire par morceaux (*piece wise linear trajectory.*) Dans le cas des trajectoires discrètes, la localisation exacte de la trajectoire entre les temps t_j^i et t_{k+1}^i est inconnue. La représentation linéaire par morceaux de cette trajectoire permet d'estimer les observations par un segment entre deux observations consécutives. Dans cette représentation, aucune hypothèse n'est faite sur l'indexation temporelle des segments.

Definition 3. *Trajectoire Linéaire par morceaux Une trajectoire linéaire par morceaux,* T_{pl}^i *est définie comme :*

$$T_{pl} \ : \ ((s_1), \ldots, (s_{n-1})),$$

où $s_k \in \mathbb{R}^4$ *est la taille de la trajectoire et* n_{pl} *est la somme des longueurs des segments qui la composent* $n_{pl} = \sum_{i \in [1 \ldots n-1]} \|p_i p_{i+1}\|_2$.

Une autre étape de pré-traitement communément appliquée aux données géolocalisées sur réseaux routiers consiste à utiliser les informations de ce réseau routier pour améliorer la qualité des observations. Cette étape, appelée *map-matching*, présente deux avantages. Il permet tout d'abord de réduire les erreurs de mesure du GPS et donc améliorer la précision de l'estimation de la localisation réelle du taxi en utilisant la localisation exacte du réseau routier. Le *map-matching* peut également servir à estimer les localisations d'un véhicule entre deux localisations effectivement observées. Cette tâche peut-être effectuée en estimant le chemin qui a été le plus probablement emprunté par un véhicule entre ces deux localisations.

Cependant, nous avons fait le choix de ne pas appliquer ces transformations à nos données car elles requièrent de posséder la carte et la géolocalisation précise du réseau routier. Or, nous souhaitons que notre modèle soit facilement duplicable, et ne nécessite pas d'autres données que les données brutes issues de parcours de véhicules. De plus, ne pas considérer le réseau routier nous permet d'appliquer notre modèle à différents types de données qui ne sont pas soumises à ce type de réseau. Nous n'utiliserons donc que les définitions des trajectoires définies ci-dessus dans la suite de ce document.

Classification non-supervisée de trajectoires

Après avoir défini la notion de trajectoire, nous souhaitons déterminer quels sont les principaux profils de trajectoires, *i.e.*, les principales routes empruntées par les utilisateurs du réseau routier. La connaissance de ces routes représente une information précieuse pour l'analyse du trafic. Dans cette optique, nous avons alors appliqué une méthode de classification non-supervisée sur un ensemble de trajectoires. La classification non-supervisée consiste à regrouper des individus similaires par groupe, tout en veillant à ce que chaque groupe soit bien différent les uns des autres. Il existe de multiples façons d'évaluer la similarité entre deux individus, et donc de multiples algorithmes de classification, pouvant produire des partitions très différentes. De plus, la complexité des trajectoires, de tailles

et d'indexations temporelles différentes, en fait une tâche compliquée. Afin de trouver la méthode de clustering la plus adaptée à notre objectif : relever les principales routes empruntées par les utilisateurs du réseau routier , nous avons mené une étude approfondie des algorithmes de classification présentés dans la littérature, qu'ils soient spécifiques aux trajectoires ou non. Cependant, nous n'avons pas trouvé de méthodes dans la littérature correspondante, permettant de produire la classification désirée. De cette revue, nous tirons néanmoins deux conclusions :

- Définir une méthode de classification spécifique aux trajectoires apparaît comme une tâche complexe pour atteindre notre objectif. Nous préférons les méthodes de classification basées sur des mesures de distance entre trajectoire. Cela implique de définir une nouvelle distance capable de comparer deux trajectoires en fonction de la route qu'elles ont parcouru.

- Pour obtenir la classification désirée, nous ne devons pas prendre en compte l'indexation temporelle des trajectoires. En effet, les groupes obtenus doivent être représentatifs des principales routes empruntées par les utilisateurs. L"indexation temporelle n'étant pas significative pour cette tâche, seules la la forme et la position géographique des trajectoires seront considérées.

Afin de satisfaire cet objectif, nous proposons une nouvelle distance, la *Symmetrized Segment-Path Distance* (*SSPD*).

Definition 4. *La distance SPD est définie ainsi :*

$$D_{SPD}(T^1, T^2) = \frac{1}{n_1} \sum_{i_1=1}^{n_1} D_{pt}(p_{i_1}^1, T^2),$$

où $D_{pt}(p_{i_1}^1, T^2) = \min_{i_2 \in [0,\dots,n_2-1]} D_{ps}(p_{i_1}^1, s_{i_2}^2)$.

Sur la Figure 1, nous pouvons observer un exemple de la distance *SPD* entre une première trajectoire, T^1 comparée à une seconde, T^2. À différentes observations de la localisation de la trajectoire T^1, nous mesurons la distance entre cette observation et la représentation linéaire par morceaux de la seconde trajectoire. En évaluant la moyenne de la distance *SPD* d'une distance à une autre, et sa réciproque, nous obtenons la distance *"Symmetrized Segment-Path Distance"*, SSPD.

Definition 5. *Symmetrized Segment-Path Distance*

$$D_{SSPD}(T^1, T^2) = \frac{D_{SPD}(T^1, T^2) + D_{SPD}(T^2, T^1)}{2}.$$

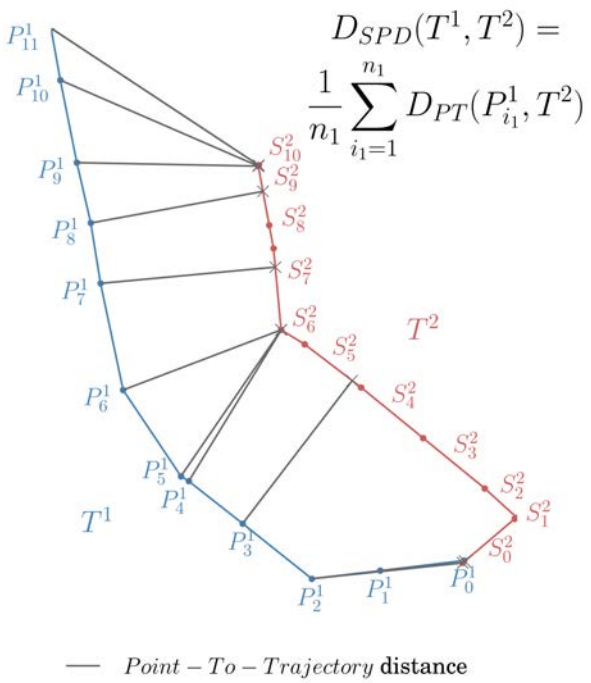


FIGURE 1 – Distance SPD de la trajectoire T^1 à la trajectoire T^2

La distance *SSPD* permet de comparer des trajectoires de tailles différentes. Elle ne compare pas des trajectoires uniquement à partir des localisations observées, mais leur attribue un poids plus important, permettant ainsi de considérer comme similaires deux trajectoires empruntant le même chemin, mais avec des indexations temporelles différentes. Sur la Figure 1, nous pouvons observer que la distance *SSPD* discerne à la fois des trajectoires qui sont physiquement éloignées l'une de l'autre mais également celles de formes différentes. La distance *SSPD* permet également de comparer deux trajectoires dans leur globalité, évitant de considérer comme similaires une trajectoire, et une de ses sous-trajectoires. Enfin, elle ne nécessite aucun paramètre additionnel, et peut donc être appliquée sans étude préalable sur n'importe quel couple de trajectoires.

Une fois ces distances calculées pour chaque couple de trajectoires, il reste à choisir l'algorithme de clustering le plus adapté. Ce choix est limité par les caractéristiques propre aux trajectoires et à la distance *SSPD*. Les algorithmes retenus sont la *classification ascendante hiérarchique (CAH)* et la *Propagation d'affinité (AP)* car ils prennent en entrée des matrices de distance et/ou de similarité qui ne sont pas nécessairement soumises à l'inégalité triangulaire, ce qui est le cas de la distance *SSPD*. Nous retiendrons finalement la *classification ascendante hiérarchique*, utilisant la distance ou saut de *Ward* comme mesure de dissimilarité inter-classe. Cet algorithme est préféré à la *propagation d'affinité* pour sa capacité à produire des partitions plus parcimonieuses. Il permet en outre de

produire facilement un nombre de classe désiré, ce qui n'est pas le cas de la *propagation d'affinité*.

La partition de l'espace obtenue par la méthode *CAH* avec la distance, ou saut, de *Ward* sur la matrice de distance *SSPD* produit ainsi des groupes de trajectoires décrivant les principaux flux de déplacements. Il révèle ainsi les principales routes empruntées par les utilisateurs du réseau routier.

La prochaine étape consiste à structurer les données au sein de chacun de ces groupes de trajectoires afin de comprendre comment les utilisateurs se déplacent sur chacun des chemins décrits par ces groupes.

Modèle de mélange gaussien.

Nous souhaitons, dans un premier temps, modéliser et comprendre le mouvement des utilisateurs au sein de chacun des groupes de trajectoires. Afin de pouvoir utiliser les informations extraites de cette modélisation, nous souhaitons également pouvoir associer une nouvelle trajectoire n'ayant pas servi à la classification, au groupe de trajectoires auquel elle appartient le plus probablement.

Pour satisfaire ces objectifs, nous utilisons le modèle de mélange gaussien, que nous appliquons sur chacun des ensembles de points constitués de tous les points des trajectoires d'un même groupe de trajectoire. Ainsi, ces différentes partitions vont permettre de créer des zones *data-driven* qui prennent en compte le mouvement des utilisateurs.

Definition 6. *Un modèle de mélange gaussien est un modèle probabiliste dont la densité est une somme pondérée de K densités de loi gaussienne. Elle est donnée par cette équation :*

$$\Phi(p) = \Phi(p|\Theta) = \sum_{k=1}^{K} \omega_k \cdot \phi_k(p),$$

ou ω_k est le poids de la composante k, i.e. la probabilité a priori pour chaque point p d'appartenir à la $k^{ème}$ composante du mélange , tel que $\sum_{k=1}^{K} \omega_k = 1$, et $\phi_k(p)$, $i = 1, \ldots, k$ est une fonction de densité de loi gaussienne.

Le modèle de mélange gaussien peut être considéré à la fois comme un modèle de classification supervisée ou non. Interprété comme un modèle de classification non-supervisée, il permet de modéliser l'ensemble des points issus des trajectoires d'un même groupe de trajectoire, par un ensemble d'états, chacun de ces états est modélisé par l'une des densités du mélange gaussien. Cette modélisation permet d'associer chaque observation à l'un de ces états, et ainsi d'introduire une nouvelle définition de trajectoire. En effet, les trajectoires peuvent désormais être interprétées comme une suite d'états, plutôt que comme une suite d'observations. Ce qui nous amène à la définition de trajectoire d'états :

Definition 7. *Trajectoire d'états. Une trajectoire d'états TS^i, correspondant à la trajectoire discrète, T^i, est définie comme suit :*

$$TS^i[(l_1^i, t_1^i), \ldots, (l_{n^i}^i, t_{n^i}^i)],$$

où l_j^i est l'état associé à l'observation p_j^i, n^i est la taille de la trajectoire TS^i.

Cette nouvelle représentation d'une trajectoire en trajectoire d'états possède plusieurs avantages. En effet, elle permet :

- une comparaison rapide entre trajectoires en considérant, par exemple, deux localisations appartenant à un même état comme similaires.
- de révéler des profils de déplacement en comparant des séquences d'enchaînement d'états.
- d'extraire facilement différentes statistiques en comparant notamment les observations appartenant aux mêmes états.

Interprété comme un modèle de classification, le modèle de mélange gaussien permet d'associer à chaque nouvelle observation une probabilité d'appartenir à chacun des états des différents mélanges gaussiens. Ces probabilités permettent alors d'assigner à chaque nouvelle trajectoire une probabilité d'appartenir à chaque groupe de trajectoires. Nous pouvons alors utiliser les informations et caractéristiques de chacun de ces groupes de trajectoires pour prédire le comportement de cette nouvelle trajectoire. Ces probabilités d'appartenance sont évaluées à partir d'estimations des densités de loi gaussienne, ce qui permet des calculs rapides, et ainsi, des évaluations en temps réel. Nous expliquons plus en détail, Section I.2, comment ces probabilités sont calculées.

Nous avons ainsi établi un modèle qui permet de décrire l'espace en plusieurs zones appelées états, chacune d'entre elles étant modélisée par une loi gaussienne. Cette discrétisation de l'espace peut s'apparenter à une grille obtenue de façon *data-driven*. En effet, chacun de ces états est issu d'un partitionnement d'un ensemble de points, tous issus de trajectoires empruntant la même route, contrairement à un partitionnement de l'espace en grille orthogonale. Ce partitionnement, qui permet donc de modéliser le déplacement des véhicules sans avoir besoin des informations sur la structure réseau routier, s'applique donc aux enregistrements GPS de tout type de mobiles. Il permet également de modéliser le déplacement des véhicules sans avoir besoin des informations du réseau routier utilisé par les véhicules. Nous présentons, dans la partie suivante, le résultat de cette modélisation sur deux jeux de données publiques.

Exemple de modélisation des données sur deux jeux de données publiques.

Tout au long de cette thèse, nous avons utilisé deux jeux de données différents pour tester notre méthodologie.

- Le premier jeu de données est composé d'observations issues de Taxi à San-Francisco aux États-Unis [Piorkowski et al., 2009]. Il contient plus de 11 millions de coordonnées GPS auxquelles s'ajoutent l'heure à laquelle les coordonnées ont été observées ainsi que l'identité du taxi qui a produit ces observations. Cinq cents taxis ont produit ces observations durant 30 jours avec un intervalle de 60 secondes entre chaque observation. Nous avons extrait deux sous-ensembles de ces jeu de données. Ces nouveaux jeux de données sont composés de trajectoires ayant en commun leur point de départ : la gare Caltrain de San-Francisco. Le premier contient 2.574 trajectoires de taxis terminant tous leurs courses dans le centre ville de San-Francisco. Ces données sont notamment utilisées dans le Chapitre 1 de cette thèse. Le second dataset contient 4.127 trajectoires dont le point de départ est également la gare Caltrain de San-Francisco, mais dont la zone d'arrivée est élargie. Dans le chapitre 2, ainsi que dans les prochaines sections, nous appellerons ce jeu de données, le *jeu de données Caltrain* (*Caltrain dataset*).

- Le second jeu de données a été fourni dans le cadre de la compétition Kaggle "ECML/PKDD 15 : Taxi Trajectory Prediction (I)"[kaggle, 2015]. Ce jeu de données est composé de plus de 83 millions d'observations GPS sur une année complète (Du 01/07/2013 au 30/06/2014) provenant de 442 taxis circulant dans la ville de Porto au Portugal. L'intervalle de temps entre chacune de ces observations est de 15 secondes. Nous étudions dans cette thèse un sous-ensemble de ce jeu de données que nous appelons *jeu de données Sao Bento* (*Sao Bento dataset*). Ce jeu de données est composé de 19423 trajectoires, ayant en commun le même point de départ : la gare *Sao Bento* de Porto.

Ces deux jeux de données diffèrent l'un de l'autre par l'intervalle de temps entre chaque observation : 60 secondes pour le jeu de données *Caltrain* et 15 secondes pour le jeu de données *Sao Bento*. La densité des points disponibles est beaucoup plus élevée pour le second jeu de données et contient donc des informations plus précises. Ensuite le réseau routier des villes de San Francisco et de Porto sont également très différents. À San Francisco, la plupart des routes sont parallèles ou perpendiculaires entre elles, tandis qu'à Porto le réseau routier est beaucoup plus désordonné. Ces différences vont nous permettre de tester la capacité de notre modèle à s'adapter à des jeux de données aux caractéristiques différentes, de façon *data-driven* et sans nécessiter de paramètres spécifiques sur les réseaux routiers.

Sur les Figures 2 et 3, nous pouvons observer des illustrations de la procédure mise en place pour structurer les donnés appliquées respectivement aux jeu de données *Caltrain* et *Sao Bento*.

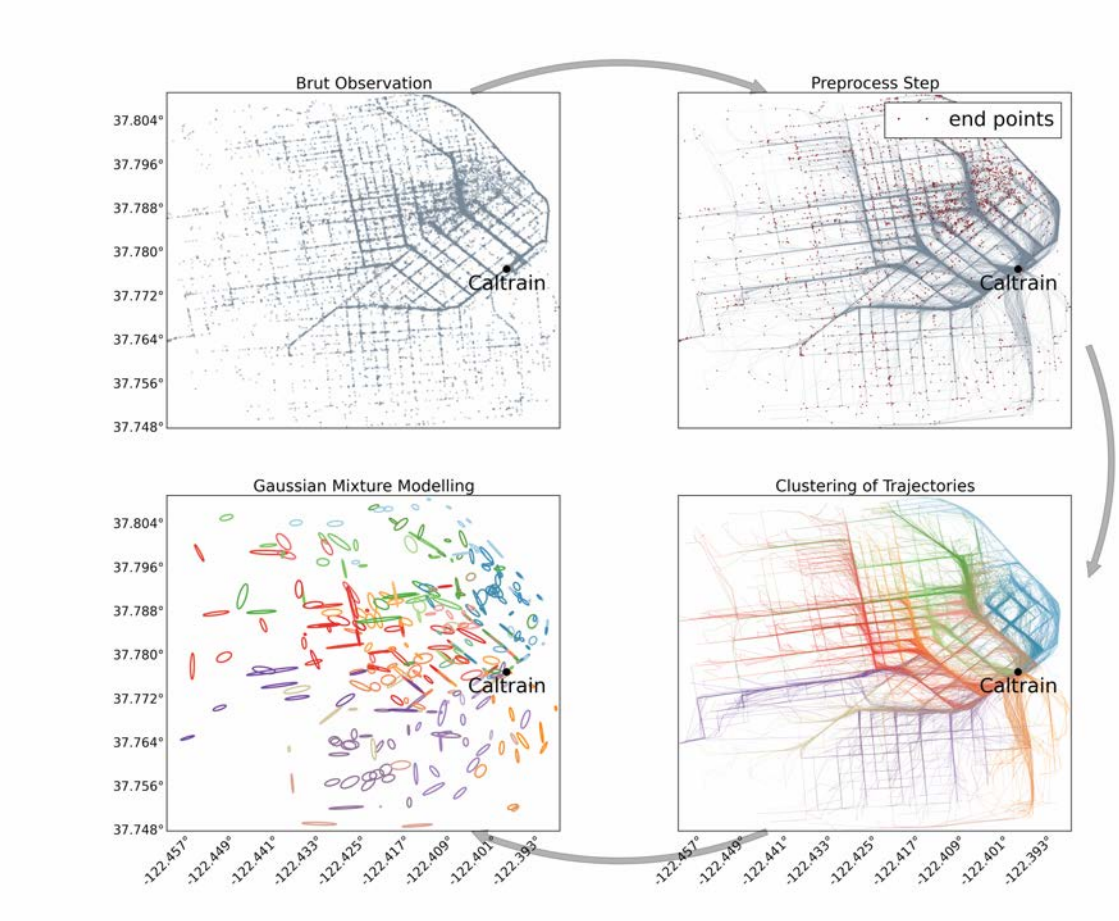


FIGURE 2 – Caltrain Dataset - Structured Data

I.2 Application et prédictions

De nombreux problèmes sont liés aux données géoréférencées. Nous traitons ici des données GPS issues de trajets de taxis. On peut imaginer que pour une société de taxi, savoir où et quand ses taxis vont finir leur courses en cours, lui permettra de sélectionner le meilleur taxi pour une nouvelle demande de trajet en fonction de la position du futur client et de l'heure à laquelle il souhaite effectuer cette course. La prédiction de la destination de l'utilisateur est également essentielle pour certaines applications de téléphone permettant de recommander des sites à visiter où encore d'améliorer le ciblage publicitaire. En effet, ces applications ont pour la plupart accès aux données GPS du smartphone de l'utilisateur mais n'ont pas forcément accès à l'information sur sa destination lorsqu'il est en mouvement. Certaines recherches sont également effectuées afin d'initialiser automatiquement la destination souhaitée dans le système de navigation de certains modèles de voitures. La prédiction est alors effectuée à partir des habitudes du conducteur et des

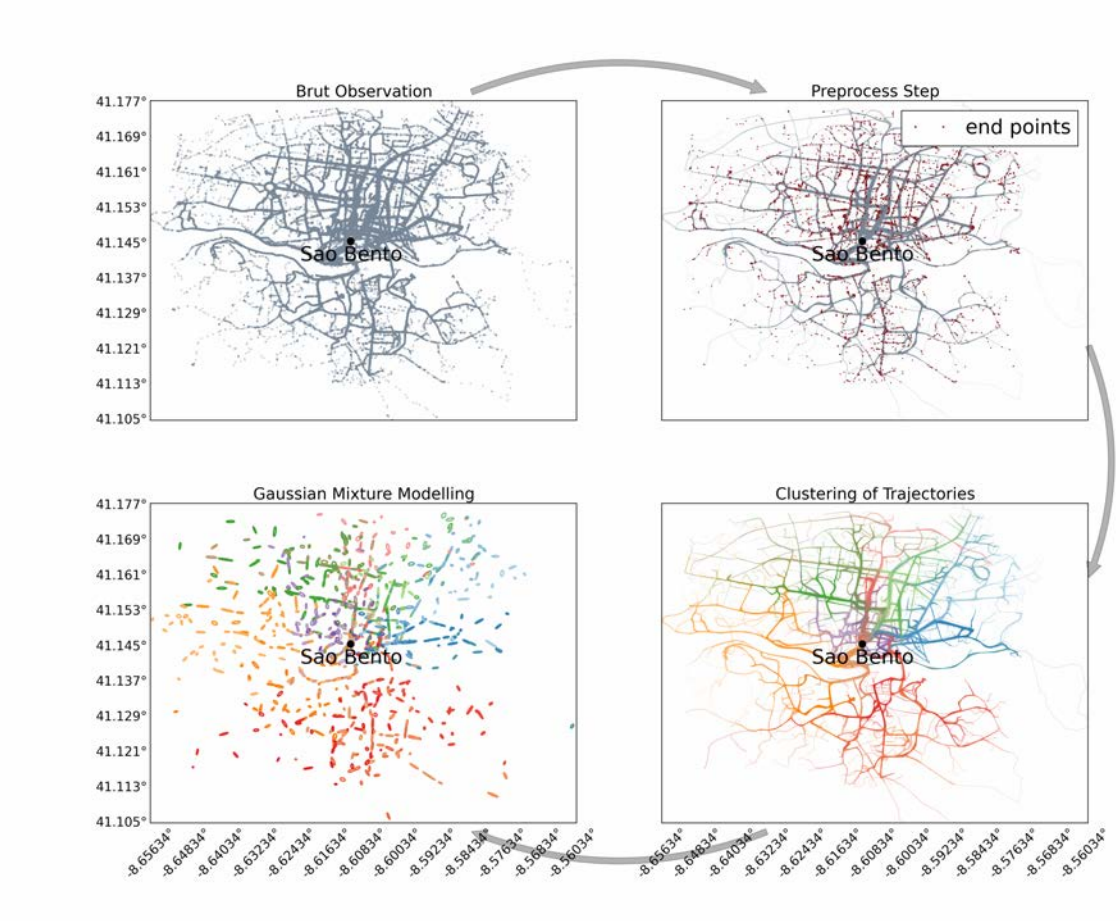


FIGURE 3 – Sao Bento Dataset - Structured Data

différents trajets qu'il a emprunté. Ces applications sont nombreuses et variées.

Dans cette section, nous présentons tout d'abord la manière dont nouvelles trajectoires peuvent être intégrées au modèle afin d'obtenir leur représentation en trajectoires d'états. Nous étudions ensuite différents types d'applications de prédiction réalisables à partir de ce modèle.

Classification de Trajectoires

Nous souhaitons associer à une nouvelle trajectoire T^c son profil de mouvement le plus approprié. Le profil est choisi parmi ceux obtenus par la classification non supervisée de trajectoires. Cette tâche peut-être effectuée facilement en calculant la vraisemblance de chacune des observations qui composent la trajectoire selon les paramètres des différents modèles de mélanges gaussiens associés au différents groupes de trajectoires. Ces vraisemblances peuvent être interprétées comme des scores, ou, une fois normées, comme des

probabilités d'appartenir à chacun des groupes de trajectoires. Ainsi, le score de la trajectoire T^c associée au $m^{\text{ème}}$ groupe de trajectoires, modélisé par le mélange gaussien de densité $\Phi^m(p_j^c|\Theta_{ML}^m)$, est calculé de la manière suivante :

$$\begin{aligned} s^m(T^c) = \quad & \mathcal{L}(\Theta_{ML}^m|T^c) = P(T^c|\Theta_{ML}^m) \\ & = \prod_{p_j^c \in T^c} \Phi^m(p_j^c|\Theta_{ML}^m), \end{aligned} \tag{1}$$

avec $\Phi^m(p_j^c|\Theta_{ML}^m)$ la densité du mélange gaussien définie Equation (6). Ce score présente l'avantage d'être rapidement mis à jour. En effet, lorsqu'une nouvelle observation d'une trajectoire apparaît en cours de réalisation, il suffit de calculer la fonction densité de chaque mélange gaussien pour cette observation, et de l'ajouter au calcul du score. Ces évaluations sont rapides et permettent de faire évoluer la prédiction de la classification de trajectoire en temps réel.

La trajectoire, T^c est alors associée au groupe de trajectoires, m, pour lequel son score, $s^m(T^c)$, est le plus grand. Chaque observation, p_j^c, de cette trajectoire peut alors être assignée à un état (m, n) correspondant à l'une des densités de loi gaussienne, $\phi_n^m(p)$, qui compose le mélange gaussien. Une observation est alors associée à la densité pour laquelle le calcul de sa vraisemblance est la plus forte. Ainsi on note, $l_j^c = (m, n)$ l'état auquel est assigné le point p_j^c, où n est estimé de la façon suivante :

$$n = \max_{k \in [1 \ldots k^m]} \phi_k^m(p_j^i). \tag{2}$$

En appliquant cette équation à chaque observation de la trajectoire T^c, on obtient sa représentation en trajectoire d'états, TS^c. Cette conversion va permettre de comparer cette trajectoire aux autres trajectoires ayant servi à la construction du modèle.

Nous avons testé cette méthode de classification sur les deux jeux de données *Caltrain* et *Sao Bento*. Nous avons pour cela utilisé la méthode de validation croisée. Les jeux de données sont d'abord divisés en 10 échantillons de taille égale. Le modèle est ensuite généré sur un ensemble de données composé de 9 des 10 échantillons, et nous avons effectué divers tests sur l'échantillon restant. Cette procédure est répétée 10 fois, de manière à ce que chaque échantillon serve d'échantillon test au moins une fois.

Résultats Expérimentaux : Les résultats de ces expériences démontrent l'efficacité de notre méthode de classification. Les tests ont été effectués sur différents partitionnements de l'ensemble des trajectoires des différents échantillons test. Le nombre de groupes produits pour chacun de ces partitionnements varie de 5 à 100 et par intervalle de 5. Pour chacun de ces tests, nous avons pu démontrer que le pourcentage de trajectoires bien classées est toujours supérieur à 85% pour le jeu de données *Caltrain* et supérieur à 91% pour le jeu de données *Sao Bento*. De plus, cette méthode permet de ne pas assigner une trajectoire à un groupe unique de trajectoires mais une probabilité d'appartenir à chacun

des groupes. Ainsi si l'on regarde les 3 groupes de trajectoires auxquels les trajectoires testées appartiennent le plus probablement, le pourcentage de trajectoire pour lequel le groupe correct fait partie de ce top 3 est supérieur à 96% pour *Caltrain* et supérieur à 99% pour *Sao Bento*, et ce quelque soit le nombre de groupes produits par la classification non-supervisée de trajectoire.

Applications

A partir du modèle défini Section I.1, et de la méthode de classification définie dans la Section précédente, nous avons défini 3 applications, permettant de résoudre différents problèmes de prédiction. Nous présentons, de manière succincte, dans cette partie, ces différentes méthodes et les principaux résultats obtenus suite à l'application de ces méthodes sur les jeux de données *Caltrain* et *Sao Bento* en suivant systématiquement la procédure de validation croisée.

- **Prédiction de la destination Finale :** Nous avons défini deux méthodes de prédiction de la destination finale d'une trajectoire en cours de réalisation. La première méthode consiste à assigner la nouvelle trajectoire à un groupe de trajectoires selon la méthode de classification décrite précédemment. Nous pouvons alors prédire la localisation du point d'arrivée de cette trajectoire en calculant la moyenne des localisations des arrivées des points d'arrivée des trajectoires composant ce groupe. La seconde méthode prend en compte la moyenne des localisations des arrivées des trajectoires de chaque groupe. La localisation d'arrivée de la trajectoire en cours est alors prédite en calculant la somme pondérée de ces différentes moyennes de localisation d'arrivée de chaque groupe. Les poids servant à la pondération sont les scores calculés grâce à la méthode de classification supervisée.

  Résultats Expérimentaux : La seconde méthode de prédiction de la destination finale de trajectoires donne les meilleurs résultats pour les jeux de données *Caltrain* et *Sao Bento*. Par ailleurs, cette méthode Cette peut également aider à choisir le nombre optimal de groupe pour le partitionnement. Il existe en effet une limite après laquelle l'ajout de nouveaux groupes de trajectoires n'améliore pas la qualité de la prévision. Ainsi, pour un partitionnement en 25 groupes de trajectoires du jeux de données *Caltrain*, nous pouvons prédire la localisation de la destination finale des trajectoires avec une erreur de $1km$ en moyenne lorsqu'elles ont accompli 50% de leur parcours et avec une erreur de $0.6km$ lorsqu'elles ont accompli 75%. Ces valeurs sont d'environ $0.75km$ et $0.45km$ pour le jeu de données *San Francisco*.

- **Prédiction du temps d'arrivée :** Nous avons établi une méthode permettant la prédiction du temps d'arrivée des trajectoires à partir de modèles de régression linéaire avec pénalisation *Lasso*. Pour chaque état obtenu grâce aux modèles de mélanges gaussiens, nous construisons un modèle de régression où le jeu de données

d'apprentissage est constitué des observations appartenant à cet état. La valeur à prédire est le temps d'arrivée de la trajectoire à laquelle appartiennent les observations. Les variables sont : le temps associé à l'observation, le nombre d'observations d'une même trajectoire déjà observée dans ce même état, l'heure de la journée, la journée de la semaine, *etc...*

Résultats Expérimentaux : Avec cette méthode, nous sommes capables de prédire le temps d'arrivée des trajectoires avec une erreur de 110 secondes en moyenne lorsque la trajectoire est à la moitié de sa réalisation et avec une erreur moyenne de 70 secondes lorsque la trajectoire est à 75% de sa réalisation pour le jeu de données *Caltrain*. Ces erreurs sont de respectivement 90 et 50 secondes pour le jeu de données *Sao Bento*.

- **Prédiction de la prochaine localisation :** Nous avons défini deux méthodes pour prédire la prochaine localisation en cours de réalisation. Ces deux méthodes sont basées sur des modèles de Markov (Chaîne de Markov du premier ordre et modèle de Markov Cachés) pour lesquelles les états sont issus des modèles de mélanges gaussiens. Une fois le prochain état prédit, à l'aide de ces modèles, la localisation est prédite soit en prenant la moyenne des localisations des observations de ces états, ou en considérant cette localisation moyenne comme la direction vers laquelle la trajectoire va se déplacer.

  Résultats Expérimentaux : Les meilleurs résultats sont obtenus à l'aide du modèle de chaîne de Markov du premier ordre et avec la seconde méthode d'estimation de la localisation. Avec ces méthodes nous pouvons prédire la prochaine localisation observée d'une trajectoire, 60 secondes plus tard, avec une erreur comprise entre 450 et 325 mètres pour le jeu de données *Caltrain*. Pour le jeu de données *Sao Bento*, la prédiction est réalisée 15 secondes plus tard. L'erreur de prévision est alors comprise entre 50 et 75 mètres.

Ces méthodes et les résultats de leurs applications sur les jeux de données tests sont développés plus en détails dans le Chapitre 2.

Le principal enseignement délivré par résultats, est la simplicité avec laquelle nous pouvons établir de nouvelles méthodes à partir du modèle défini. En effet, la nouvelle structure des trajectoires permet de rapidement en extraire différentes caractéristiques.

Les différentes méthodes de prédiction sont définies de façon générique. Par exemple, les résultats exprimés ici assument que nous ne connaissons pas le vrai groupe auquel appartiennent les trajectoires lorsque sont effectuées les différentes prédictions. Or, cette information est parfois connue à l'avance. Ces informations peuvent facilement s'intégrer à la prédiction, sans nécessiter de ré-apprentissage. Les différentes prédictions peuvent également être réalisées en temps réel, et les modèles appris ne nécessitent pas de ré-apprentissage à mesure que la trajectoire progresse dans sa réalisation.

Nous avons également pu tester l'influence de différentes variables contextuelles (l'heure de la journée, le jour de la semaine) sur la qualité de ces différentes prédictions pour des résultats variables. Ainsi nous avons pu constater que ces variables semblaient avoir une influence positive sur la qualité de la prédiction de la destination finale, mais une influence inexistante sur la prédiction du temps d'arrivée. De même cette influence varie d'un jeu de données à l'autre.

Ce type d'information peut-être très intéressant pour un analyste du trafic routier, qui pourra s'appuyer sur notre modèle pour définir des méthodes de prédiction et d'analyse du trafic.

Nous soulignons également le fait que nous avons cherché à développer des applications liées au trafic routier du fait que nous possédions deux jeux de données publiques de taxis. Ces applications sont cependant transposables à d'autres types de données, telles que les trajectoires de bateau, d'avion ou d'animaux. Des applications spécifiques à ce type de trajectoires peuvent en découler, notamment pour l'étude de comportement anormaux de port à port pour les flottes commerciales.

Future direction de travail

Sur la base de cette thèse, deux directions de recherche, en particulier, peuvent faire l'objet de futurs travaux.

- Nous avons analysé dans cette thèse deux sous-jeux de données, issus de deux jeux de données différents, que nous avons extraits de façon *data-driven*. Nous avons repéré pour cela les zones de forte densité des points de départs des trajectoires et extrait différents groupes de trajectoires en fonction de ces zones. Ainsi, on peut diviser l'ensemble des trajectoires et appliquer notre méthode sur chacun des sous-jeux de données obtenus. Ce partitionnement peut être amélioré en recherchant les couples principaux d'origines-destinations, et en appliquant notre méthode sur des jeux de données encore plus précis.

- Le second objectif consiste à intégrer des informations de l'environnement au modèle, comme des commerces, des services ou des lieux d'interaction. Ces points d'intérêt peuvent facilement être intégrés au modèle grâce à leur localisation en les associant à un ou plusieurs états. Ainsi, à partir des habitudes des individus vis-à-vis de ces points d'intérêt, de leurs caractéristiques et de leur popularité, nous pouvons conseiller à ces individus une place où s'arrêter ou acheter ce dont ils ont besoin en fonction, par exemple, de l'heure de la journée.

Outre ces deux objectifs, de nombreuses applications peuvent être imaginées, telles que la détection de trajets anormaux dans le temps et/ou l'espace ou la recommandation de parcours. Le modèle peut également être d'utiliser pour aider à la compréhension du trafic à l'intérieur d'une ville en mettant en évidence les principaux flux de circulation, les zones de congestion, et aider à régler ces différents problèmes.

II Apprentissage sur données massives

La seconde partie de cette thèse traite du choix des meilleurs environnements technologiques pour l'application de techniques d'apprentissage sur des données massives.

Dans la première partie, nous avons utilisé le langage de programmation *Python* et en grande partie la librairie *Scikit-Learn* pour l'utilisation de différents algorithmes de classification supervisée et non-supervisée (*propagations d'affinité*, *modèles de mélanges gaussiens*, *régression linéaire*) mais également pour implémenter nos méthodes originales (distance *SSPD*, méthodes de prédictions) ainsi que pour organiser l'architecture de nos modèles.

Or, comme énoncé précédemment, nous produisons une quantité considérable de données. En conséquence, de nombreux produits et services technologiques sont développés aujourd'hui afin de pouvoir exploiter ces volumes de données en les stockant et en les traitant de façon distribuée. Nous souhaitons donc étudier les performances de ces nouvelles technologies afin de choisir l'environnement le mieux adapté à notre modèle. Si les avantages de l'utilisation de ces technologies sont évidents pour certaines opérations simples, notamment pour le pré-traitement des données où les transformations sont appliquées sur les individus indépendamment les uns des autres, l'adaptation d'algorithmes plus complexes peut poser des problèmes. Ainsi, avant d'adapter directement le modèle et ses applications de prédictions développés dans la partie précédente à ces nouvelles technologies, nous avons souhaité étudier les performances de ces technologies sur des problèmes d'apprentissage statistique bien connus.

Ce problème est complexe car il dépend de nombreux paramètres. On peut définir un volume de données comme *massif* lorsque ces données ne peuvent plus être stocé" et/ou traité à partir d'un ordinateur. Mais cette définition dépend alors des nombreuses caractéristiques de l'ordinateur (taille du disque dur, de la mémoire vive, nombre de processeurs ou de cartes graphiques). Lorsque le stockage et les calculs se font sur un environnement distribué, le nombre de paramètres explose. Cette complexité empêche une comparaison exhaustive de toutes les technologies en combinant tous les différents paramètres. L'objectif a donc été réduit à la comparaison de différentes implémentations d'algorithmes connus dans trois environnements différents : *R*, logiciel bien connu des statisticiens et proposant de nombreuses méthodes d'apprentissage statistique ; la librairie *Scikit-Learn* dédiée à l'apprentissage machine du langage *Python* et les librairies *SparkML*, *MLlib* du *framework Spark* dédiées au calcul sur données distribuées.

Applications

Afin de réaliser cette comparaison, nous avons tiré profit d'expériences réalisées dans le cadre d'applications contractuelles au sein de l'entreprise Datasio. À travers ces applications, nous avons utilisé différentes méthodes d'apprentissage statistique sur des jeux de données aux caractéristiques différentes. Il est également nécessaire pour une entreprise

de travailler avec les meilleurs outils technologiques. Ainsi différentes solutions ont été testées afin de déterminer la meilleure option. Pour des raisons de confidentialité, ces applications ne peuvent être publiées dans cette thèse. Cependant, nous présentons dans cette partie les conclusions tirées sur trois cas d'apprentissage statistique appliqués sur des jeux de données publiques dont la taille et les caractéristiques correspondent à celles rencontrées sur ces applications industrielles :

Reconnaissance de caractères (MNIST)

La reconnaissance de caractères manuscrits est un problème commun à de nombreux domaines et applications de reconnaissance de documents informatisés (passeport, documents bancaires ou postales). Le jeu de données étudié (MNIST [Lecun et al., 1998]) est composé de 60000 images de chiffres manuscrits. Les niveaux de gris de chacun des 784 pixels de chaque image sont utilisés comme variable prédictive. Un jeu de données de 10.000 images est utilisée comme jeu test. Les performances des différentes implémentations de l'algorithme des forêts aléatoires dans les trois environnements étudiés sont comparés.

Résultats Expérimentaux : Les implémentations de *R* et *Scikit-Learn* de l'algorithme des forêts aléatoires sont similaires et conduisent à des résultats analogues. En revanche, les temps d'exécution sous *R* dépendent fortement de la librairie utilisée. La librairie *ranger*[Wright and Ziegler, 2016] concurrence l'implémentation en *Python* et *Cython* de *Scikit-Learn*. Avec *Spark*, nous n'avons pas réussi à obtenir des résultats similaires. Le paramètre de profondeur des arbres influence fortement la qualité de prévision, or ce paramètre dépend directement de la mémoire allouée, et il aurait fallu disposer de clusters plus conséquents pour obtenir des résultats comparables en précision. Nous avons obtenus de meilleurs résultats sur une architecture intégrée de 8go de mémoire avec *Python Scikit-Learn* ou *R ranger* plutôt qu'avec une architecture distribuée de 6 fois 7Go.

Recommandation de films

Ce cas d'application concerne la recommandation de films par filtrage collaboratif. La méthode consiste à prédire la note qu'un utilisateur attribuerait à un produit, en l'occurrence un film, en fonction des goûts de cet utilisateur et des notes attribuées par d'autres utilisateurs ayant des préférences similaires pour ce film. Le jeu de données étudié pour ce cas d'application est composé de 4 matrices creuses provenant du site *movielens.org*. Ces matrices représentent les notes attribuées par des utilisateurs (lignes) à différents films (colonnes). La plus petite matrice contient 100.000 notes de 1.000 utilisateurs sur 1.700 films, tandis que la plus grande contient 22.000.000 notes de 138.000 utilisateurs sur 27.000 films. Deux implémentations d'algorithmes de complétion de matrices ont été comparées. Celui de la librairie *softImpute* de *R* et celui par factorisation non négative (NMF) de la librairie *MLlib* de *Spark*. L'implémentation de NMF proposée par *Scikit-Learn* n'est pas

adaptée à la complétion de matrice.

Résultats Expérimentaux : Pour ce cas d'application, la méthode provenant de la librairie *MLlib* de *Spark* donne de bien meilleurs résultats que celle provenant de la librairie de *softImpute*. Les résultats obtenus avec *MLlib* concurrencent les meilleurs résultats trouvés dans les blogs de discussion, à la fois en temps et en précision. L'algorithme implémenté dans cette librairie s'avère ainsi efficace et bien adapté à l'architecture distribuée.

Catégorisation de produits :

La catégorisation de produit est un problème bien connu des sites d'e-commerce. Pour un gérant de site e-commerce, il est en effet très utile de pouvoir classer automatiquement un produit à vendre dans la bonne catégorie parmi toutes celles composant l'arborescence du site. Cette catégorisation peut être effectuée à partir des descriptions des produits déjà classés sur le site. Le jeu de données étudié est composé de plus de 15.000.000 de produits. Il a été publié dans le cadre du concours proposé par *Cdiscount* et hébergé par le site *datascience.net*. Nous avons comparé deux implémentations différentes de l'algorithme de régression logistique sous *Python* et *Spark* pour la procédure de classification supervisée. Plusieurs étapes de pré-traitements (nettoyage et *vectorisation*) sont nécessaires afin de pouvoir exploiter correctement ces données textuelles. Ces étapes ont également fait l'objet d'une comparaison entre ces deux environnements.

Résultats Expérimentaux : Les résultats de l'étape de nettoyage des données sont ceux attendus en fonction du nombre de cœurs ou du nombre d'exécuteurs et nettement en faveur de l'architecture distribuée de *Spark*. Ainsi, une architecture distribuée avec *Spark* est plus efficace pour l'étape de nettoyage qu'une architecture intégrée. L'étape de vectorisation, en revanche, pose rapidement des problèmes de mémoire avec *Spark*. En effet en sélectionnant plus de 75.000 *features*, une erreur mémoire apparaît, ce qui n'est pas le cas pour *Python* qui accepte des valeurs nettement plus grandes. De plus, le temps d'exécution de cette étape est toujours plus rapide en *Python*, quelque soit le nombre de variable sélectionnées. Enfin, pour la prévision, l'architecture intégrée sous *Python* se montre plus efficace en temps et en précision que l'architecture distribuée sous *Spark* qui nécessite, globalement, des ressources mémoires importantes pour éviter des erreurs à l'exécution.

Conclusion

Les comparaisons réalisées sur ces trois cas d'usage permettent de tirer quelques conclusions sur les environnements comparés. D'une part, pour la gestion (*munging*) de volumes et/ou de flux de données importants, les fonctionnalités déjà opérationnelles de *SparkSQL*, et *Spark Streaming*, utilisant une architecture distribuée, se montrent plus efficaces que l'utilisation d'architecture intégrée. En revanche, lorsque l'on considère les méthodes d'apprentissage statistique, le recours à *MLlib* ou *SparkML*, n'est pas une priorité. Les librairies de *Python Scikit-Learn* et de *R* utilisé sur une machine avec suffisamment de mémoire et

de processeurs offre de meilleures performances. La distribution des algorithmes sur plusieurs machines soulève d'avantages de problèmes, notamment de gestion de la mémoire, qu'elle n'en résout pour des algorithmes complexes. L'utilisation de ces technologies n'est à tenter que mûrement réfléchie et justifiée. Cependant la validité de ces résultats est limitée dans le temps avec le niveau de maturité de ces technologies. Ainsi *Spark*, dont une diffusion stable n'est disponible que depuis 2014, est en plein développement ; la version 2 est annoncée pour septembre 2016 de même que la version stable 0.18 de Scikit-Learn.

III Conclusion

Tout au long de cette thèse, nous avons étudié l'application de méthodes d'apprentissage statistique sur des jeux de données plus ou moins massifs afin d'extraire de l'information de ces données.

Nous avons établi dans un premier temps une méthodologie originale permettant l'application de méthodes de prévision sur des données de géolocalisation. Cette méthodologie consiste à structurer la donnée de manière à pouvoir en extraire les informations sur le trafic et le comportement des utilisateurs. Cette nouvelle structure permet ainsi de modéliser les données de manière à les rendre facilement utilisables et exploitables. A partir de ce modèle, nous avons défini trois méthodes de prévisions, soit le temps d'arrivée, la destination finale et la prochaine localisation. Les résultats de l'application de ces méthodes sur deux jeux de données différents prouvent que notre modèle est adapté à l'extraction rapide de caractéristiques nécessaires à ces prévisions. De plus, le modèle défini ne nécessite pas de paramètre additionnel en entrée, et peux s'adapter à tout type de jeux de données géolocalisées. Enfin les méthodes définies peuvent être appliquées en temps réel.

Dans un second temps nous avons chercher le meilleur environnement technologique à utiliser pour manipuler des gros volumes de données et appliquer notre méthode. Nous avons pour cela comparé des implémentations d'algorithmes du *framework Spark* de calculs sur données distribués, à celles présentes dans des librairies *Scikit-Learn* de *Python* ou *R* sur trois cas d'usage. Ces expériences nous ont permis de conclure que l'utilisation du *framework Spark* dépend fortement du contexte et doit être prise avec précaution. Si certaines fonctionnalités sont pleinement opérationnelles, particulièrement pour la gestion des données et la parallélisation d'opérations simples, les librairies d'apprentissage statistique ne sont pas encore fiables vis-à-vis des problèmes de gestion de la mémoire et suffisamment optimisées.

Ainsi, dans le cadre de notre méthode, certaines opérations de pré-traitement comme la conversion des données en trajectoires) ou encore le calcul de la distance (*SSPD*) entre trajectoires peuvent tirer bénéfice de ces nouvelles technologies. En effet ces étapes sont réalisées indépendamment des trajectoires ou par couple de trajectoires, et sont ainsi facilement parallélisables.

Cependant pour les étapes suivantes, l'utilisation de ces technologies est peu indi-

quée. Pour l'étape de classification non-supervisée des trajectoires, nous avons retenu la *classification ascendante hiérarchique, (CAH)* avec la distance ou saut de *Ward* comme mesure de dissimilarité entre les classes. Nous avons vu que cette méthode était la plus adaptée car elle permet de prendre en compte des distances qui ne sont pas soumises à l'inégalité triangulaire. Cet algorithme ne peut cependant pas être facilement implémenté de façon distribuée. En effet, il nécessite le calcul de la matrice symétrique d'une distance entre n individus, qui est potentiellement très grande ($n(n-1)$). Or l'algorithme nécessite également un accès à cette matrice de distance à chaque itération, ce qui rend sa parallélisation difficile. Plusieurs algorithmes implémentés de façon distribuée ont été proposé [Wang and Dutta, 2011], [Jin et al., 2015], [Tang et al., 2010] afin de limiter les calculs et le coût de stockage ou les répartir. Ces adaptations se présentent souvent comme des problèmes d'optimisation sur graphe, [Jin et al., 2015], ou de parallélisation sur des sous-ensemble de données [Wang and Dutta, 2011], [Tang et al., 2010] avant de fusionner les résultats et ne permettent pas d'obtenir la qualité de classification recherchée. Ces algorithmes sont en outre souvent exclusivement réservés à une utilisation de la méthode *CAH* avec la distance ou saut de *minimum* comme mesure de dissimilarité entre les classes. Enfin aucune implémentation n'est aujourd'hui disponible au sein des différentes librairies *Spark*. On privilégiera donc la stratégie d'une *CAH*, implémentée sur une architecture classique et appliquée sur un ensemble de données réduit, en fonction des points de départ/destination. Une fois le partitionnement réalisé, l'application des modèles de mélanges Gaussiens se fait sur des ensembles de données encore une fois réduit et ne nécessite pas d'être distribuée.

Enfin, comme nous l'avons vu précédemment, les implémentations distribuées des différentes méthodes d'apprentissage statistique ne sont pas encore mûres. La nouvelle structure que nous avons définie permet d'extraire facilement les bonnes caractéristiques nécessaires à nos méthodes de prédiction. Or, nous avons également pu tirer la conclusion, dans la deuxième partie, qu'ajouter toujours plus de données au jeu d'apprentissage, ne permet pas d'améliorer indéfiniment la qualité de la précision. En effet, dans les trois cas d'usage étudiés, nous avons pu constater qu'il existe un seuil pour lequel ajouter des données n'améliore que légèrement la qualité de la prédiction, mais fait également exploser le temps de calcul. Ces conclusions sont à nuancer, du fait du degré de maturité des solutions étudiées. Cependant, il apparaît que, si la prolifération des données a motivé le développement de nouvelles méthodes et nouveaux algorithmes, une analyse préalable fouillée est indispensable avant toute étape de prévision, afin d'extraire les bonnes caractéristiques (*features*). Dans le cas des données géolocalisées, nous avons montré que, même si les jeux de données initiaux sont de volumes très conséquents (11 et 83 millions de localisations GPS), leur préparation initiale et la recherche de structures et caractéristiques spécifiques conduit à des volumes raisonnables qui ne nécessitent pas une architecture distribuée pour obtenir des résultats de prévision très encourageants. La stratégie optimale, dans l'état actuel de développement des librairies de calcul et d'apprentissage, semble donc

bien de distribuer la préparation des données avant d'utiliser une librairie plus fiable (e.g Scikit-Learn) sur une architecture intégrée.

Chapter 1

Trajectory - Distance & Clustering

Knowing the main routes followed by people or vehicles during the day can represent precious information for mobility analysis. For example, a group of trajectories may highlight the presence of important routes not adequately covered by the public transportation service, representing information on how it can be improved. Indeed, knowing which routes have not only high density but also high continuity helps optimize bus line and terminal arrangement. It could also be beneficial for local stores in order to send advertisements, special offers or discounts, or for mobile devices representing people passing by their stores. The discovery of groups of "similar" trajectories can be solved by "clustering". In cluster analysis, the goal is to partition a data set into groups representing closely related objects, while the objects belonging to different groups are not. There are several ways to define how close objects are from one another, hence many different clustering algorithms have been established. Depending the group we want to obtain and the type of the data, the choice of a good clustering method is an important question.

In this chapter, we tackle the problem of finding the best clustering method for vehicle trajectories. We have seen that using criteria such as moving in the same way at the same time is sometimes too restrictive for discovering useful information, and thus the temporal constraint is removed in our data. Hence we are looking for groups of objects that follow the same route (i.e. the temporally oriented spatial projection of a trajectory) but at any given moment in time.

For that, we will discuss first the main known methods for clustering objects and their particularities. We then study the different existing proposals to apply these methods to trajectory data as well as the original method developed for trajectory data. In the first case, it requires that the semantics of trajectory data is completely encapsulated in a distance function, while in the second case, it is exploited in the whole clustering algorithm. We will show that the former gives the best results and we propose then a review of the existing distances for trajectory. Based on the limitations of these distances, we also establish a new distance: *Symmetrized Segment Path Distance*.

1.1 Clustering of trajectories

1.1.1 Clustering methods

Clustering is the process of dividing a set of objects into groups called *clusters*, in such a way that objects which are in a same group are more similar to each other than objects which are in different groups. There are plenty of ways of defining similarity between singular objects and even more between groups of objects. Hence, there are many different notions of what constitutes a cluster: small distances between cluster members, small distance to the representative object of the cluster, object in dense areas or objects following the same statistical distribution *etc...* Hence, many algorithms have been established according to the desired type of clusters and how to obtain it efficiently. How exactly each cluster is composed, then is decided by the choice of the (generic) clustering algorithm adopted. For instance, centre-based algorithms like *K-Means* will yield a flat set of spherical clusters; hierarchical methods will organize clusters in a multi-level structure of clusters and sub-clusters and density-based methods will form maximal, crowded groups of objects, thus not limiting the group size and, in some cases, also putting together pairs of very dissimilar objects. It is essential to understand how these algorithms work, which parameter settings they require in order to use them correctly. In this section, we present the most important algorithms used in cluster analysis. We study their particularities in order to choose the best algorithm regarding type of clustered data and the type of cluster desired. To accomplish this, we use the notation defined in Table 1.1

Table 1.1 – Notation for clustering algorithm

$\mathcal{X}$	Set of individuals to be clustered
N	Size of $\mathcal{X}$
x_i	i^{th} individual
C	Set of clusters
K	Size of C
C^k	k^{th} cluster
μ_k	mean of all object in cluster C^k
X^k	Representative object of cluster C^k
$c_i(=k)$	label of i^th individual (belonging to cluster k)

1.1.1.1 Representative clustering

The *representative clustering* methods create clusters around a representative object, which can be a member of the cluster to which it belongs (*K-medoids*, *CLARA*, *CLARANS*,

AP) or a created object such as the mean of the objects belonging to the same cluster (*K-Means*).

- **Partitioning methods**. *K-means* organizes clusters around the means object. It assumes that the objects have the same dimensions. The number of clusters desired, k, is fixed. *K-means* algorithm seeks to minimize an objective function called the Within-Cluster Sum of Squares (WCSS):

$$WCSS(\mathcal{X}) = \sum_{k=0}^{K} \sum_{\{i|c_i=k\}} (x_i - u_k).$$

K-Means then uses the Llyod algorithm, which is an approximation method which aims to minimize the $WCSS$ function. The means of each cluster are first chosen randomly. In a stepwise fashion, all objects are assigned to their closest mean and new means are then computed from all individuals belonging to the same group. This step is repeated until no change appears between clusters and the computed means are the same. *K-mean* is efficient for large data sets, but very sensitive to outliers, which can influence the mean value of a cluster.

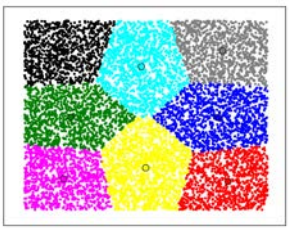


Figure 1.1 – Example of K-means clustering result

The *K-Medoid* method is very similar to *K-Mean*. It is applied when a mean object can not be defined. Clusters are built around a representative object selected among all objects in the set called a medoid. The objective function to be minimised is then:

$$cost(\mathcal{X}) = \sum_{k=0}^{K} \sum_{\{i|c_i=k\}} (x_i - X^k).$$

The most commonly used algorithm to minimise this objective function is the *Partition around medoid* algorithm, *PAM*[Kaufman and Rousseeuw, 1987]. This algorithm starts by randomly selecting k medoids in the set of individual objects. Each object is then assigned to its closest medoid. At each step, a medoid and a non-medoid individual are swapped and the new cost is computed. If the cost increases,

the swap is canceled, otherwise, a new label is computed. This algorithm can handle small datasets, but it becomes too costly for large values of K and N.

The *CLARA*[Rousseeuw, 2009] and *CLARANS*[Ng and Han, 2002] have been developed in order to solve this problematic. Instead of finding representative objects for the entire data set, *CLARA* draws a sample of the data set, applies PAM on the sample, and finds the medoids of the sample while *CLARANS*[Ng and Han, 2002] doesn't take into account all neighbors of a medoid. But it does not restrict its search of medoid to a particular subset. While *CLARA* draws a sample of nodes at the beginning of a search, *CLARANS* draws a sample of neighbors in each step of a search. With *CLARA*, the best medoids may not be selected during the sampling process. *CLARANS* avoids this problem by considering all nodes, and by sampling neighbors it handles the outliers problem.

These algorithms require the number of clusters, K, as a parameter. They also create clusters of approximately similar size and form, as in Figure 1.1, which is not suitable for discovering clusters with non-convex shapes or clusters of very different sizes.

- **Message Passing method**. Frey et al. [Frey and Dueck, 2007] propose a method based on message-passing called *affinity propagation*. Its goal is to maximize the net similarity, S_{net}, which is the negative energy plus a constraint function that enforces valid configurations:

$$S_{net}(\mathcal{X}) = -E(\mathcal{X}) + \sum_{k=1}^{K} \delta_k(X_k) \tag{1.1}$$

$$= \sum_{i=1}^{N} s(x_i, X^{c_i}) + \sum_{k=1}^{K} \delta_k(X_k) \tag{1.2}$$

where $\delta_k(X_k) = \begin{cases} -\infty & \text{if } c_k \neq k \text{ but } \exists i \ s.t. \ c_i = k \\ 0 & \text{otherwise} \end{cases}$

Here, $\delta_k(c_k)$ is a penalty term which assures that, if some data point x_i has chosen x_k as its exemplar, x_k has to be correctly labeled as its own exemplar. s is an arbitrary similarity measure, which does not have to respect the triangular inequality. *Affinity propagation* does not require the number of clusters to be prespecified. It takes as input a real number $s(k,k)$ for each data point k so that data points with larger values of $s(k,k)$ are more likely to be chosen as exemplars. These values are referred to as *preferences*. The number of clusters is influenced by the values of the input *preferences*. The greater the value of the preference is, the bigger the number of clusters is.

To maximize the net similarity, Frey et Al. developed an algorithm based on message passing.

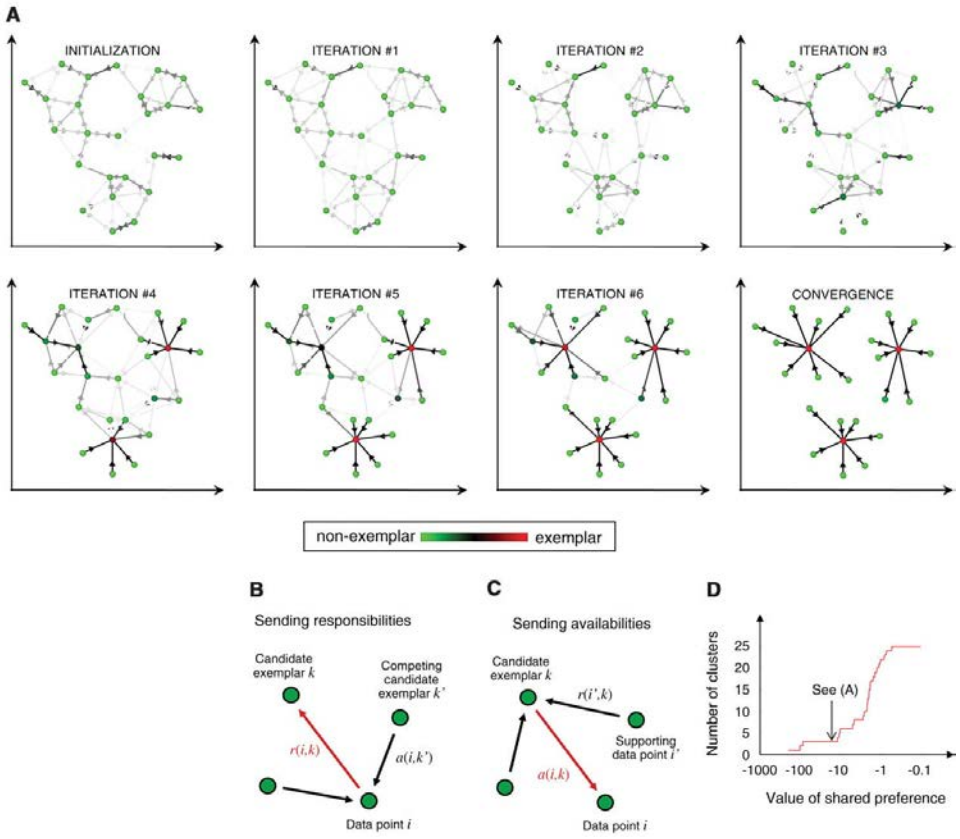


Figure 1.2 – Illustration of a message passing algorithm for affinity propagation. Taken from [Frey and Dueck, 2007]

Two kind of messages are exchanged between data points, the responsibility and the availability. The availabilities are first initialized to zero, $a(i,k) = 0$. Then the responsibilities, $r(x_i, x_k)$ are updated:

$$r(x_i, x_k) = s(x_i, x_k) - \max_{k' \text{ s.t. } k' \neq k} \{a(x_i, x'_k) + s(x_i, x'_k)\}$$

This responsibility $r(x_i, x_k)$, sent from x_i to x_k, reflects how much point x_k could be an exemplar for point x_i compared to other potential exemplars. The availability is updated as:

$$a(x_i, x_k) = \begin{cases} min\Big\{0, r(x_k, x_k) + \sum_{i' \text{ s.t. } i' \notin \{i,k\}} \max\{0, r(x_{i'}, x_k)\}\Big\} & \text{if } i \neq k \\ \sum_{i' \text{ s.t. } i' \neq k} \max\{0, r(x_{i'}, x_k)\} & \text{otherwise} \end{cases}$$

The availability $a(x_i, x_k)$, sent from point x_k to point x_i, reflects how appropriate it would be for point x_i to choose point x_k as its exemplar, taking into account the support from other points for which point x_k could be an exemplar. The message-passing procedure may be terminated after a fixed number of iterations, after changes in the messages fall below a threshold, or after the local decisions stay constant for some number of iterations.

Unlike clustering algorithms such as *K-Means* or *k-medoids*, AP does not require the number of clusters to be predetermined. But it requires us to know beforehand the preference parameters which will influence this number. Moreover, it is slow to run on a large dataset.

1.1.1.2 Hierarchical clustering

Hierarchical clustering creates a hierarchical decomposition of a dataset $\mathcal{X}$. This decomposition is based on a distance d between individuals of this dataset. A partition of the clustering for every level, from 1 to K clusters is obtained from matrices between all objects to be classified. These partitions can be obtained following two different strategies:

- Agglomerative or *Bottom-Up*: First, each object is represented as a cluster of size 1. At each step, two clusters are merged until there is just one cluster left.
- Divisive or *Top-Down*: First, the entire set of objects form one cluster of size $|\mathcal{X}|$. At each step, a cluster is selected and split into two clusters until each object represents a cluster.

The choice of merging or splitting a cluster is made according to a given linkage criterion. A linkage criterion, d_{cl} defines a distance between two sets of clusters. Agglomerative methods are easier to run. At each step, clusters with the shortest criterion linkage are combined into a single cluster. With the divisive method, we have to determine which cluster will be split and how it will be split, which is more costly. Hierarchical cluster results in meaningful results which are easily interpretable with dendogram representation. Indeed these methods will not produce a unique partitioning of the data set, but a hierarchy of partitioning. They are not very robust to outliers, which will either appear as additional clusters or even cause other clusters to merge.

1.1.1.3 Model-based clustering

Model-based methods assume that individuals of dataset $\mathcal{X}$ have been generated from the same statistical distribution. This implies that it compares individuals within the same dimension. The Gaussian mixture model is the most often used Model-Based Clustering method. In this model, each individual x_i is assumed to be generated from a mixture of Gaussian, i.e a sum of a given number K of normal distributions ϕ_k, multivariate or not :

$$\Phi(x_i, \Theta) = \sum_{k=0}^{K} w_k \phi_k(p|\mu_k, \Sigma_k),$$

where μ_k, Σ_k are the mean and the covariance of normal distribution ϕ_k. For a fixed number of clusters, K, the parameters ($\Theta = (\mu_1, \ldots, \mu_K, \Sigma_1, \ldots, \Sigma_K)$ are estimated using the expectation-maximization algorithm(EM). The expectation step creates a function for the expectation of the log-likelihood evaluated using the current estimate for the parameters, and the maximization step, which computes parameters maximizing the expected log-likelihood found on the E step. A label is assigned for each individual in the distribution k having the greatest likelihood:

$$c_i = \max_{k \in [1,\ldots,K]} \phi_k(x_i|\mu_k, \Sigma_k)$$

The main problem with these methods is that the likelihood will increase by adding parameters. But this will lead to over-fitting. Information criteria like *AIC* or *BIC* solve this problem by penalizing the number of parameters in the model, helping to establish the right number of cluster.

This method is fast, and works well as long as it is applied on data defined from a mathematical model, which is not always the case in real value data.

1.1.1.4 Density-based clustering

Density methods have been mainly designed to cluster data of arbitrary forms. It is based on the concept that the density of points inside a cluster is higher than outside a cluster. DBSCAN [Ester et al., 1996] is the first algorithm designed to find density-based clusters. The main idea is that if an individual, x_i, has at least a given number of neighbors $MinPts$ inside a given radius ϵ, this point is defined as a *core object* and all the points in its $\epsilon - neighborhood$ belong to the same cluster. For a point x_j of the $\epsilon - neighborhood$ of x_i, if x_j is a *core object*, all the points are also added to the cluster and the same study is applied to these points. Otherwise, only x_j is added to the cluster. The algorithm stops once all individuals have been visited.

Figure 1.3 – Exemple of a DBSCAN clustering result. Taken from [Ester et al., 1996]

DBSCAN efficiently discovers clusters of uniform density with arbitrary shape (Figure 1.3). If the clusters have different density, *DBSCAN* will fail. *OPTICS* [Ankerst et al.,

1999] algorithm deals with this problem by fixing only the $MinPts$ parameter. The ϵ is not fixed. The distance at which each individual x_i will be considered as a *core object* is computed. This algorithm, as in hierarchical clustering, does not result in fixed clustering. It produces hierarchical clusters, which are illustrated with a reachability-plot, Figure 1.4. A cluster can be obtained by selecting a threshold on the y-axis as seen in the lower part of the reachability plot. The result will then be similar to a DBSCAN clustering result with the same ϵ and $MinPts$ parameters.

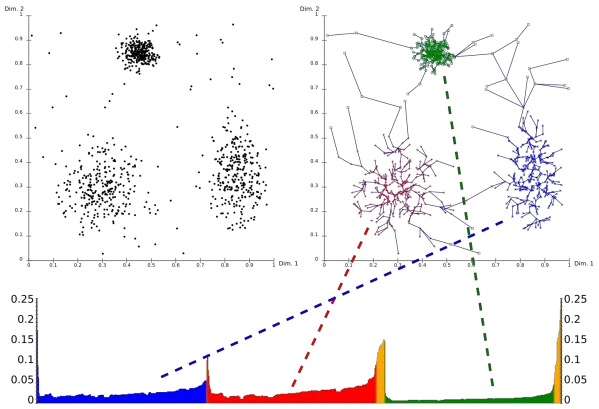


Figure 1.4 – Exemple a reachability plot

OPTICS is more costly and slower compared to *DBSCAN*. But the ϵ parameter no longer needs to be set. However, *OPTICS* will not produce strict partitioning. This is a visual method requiring study of the reachability plot results. *OPTICS* makes limits between density cluster more flexible. But, for both *DBSCAN* and *OPTICS*, the limit has to clearly exist and can be applied only on clearly separable clusters. Density based methods will result in unique cluster if they are applied on the dataset in Figure 1.1.

1.1.2 Application to trajectory

As seen section I.1, trajectory objects are particular objects. They can be of different sizes, and can have several definitions. The visual information give by vector data, can sometimes lead us to a good intuition of the best method to be used in determining trajectories. But *K-means* or *Gaussian Mixture* can not be directly used because of the different sizes of the trajectory objects. A density based-method needs clear limits between cluster. Trajectories do not respect these conditions. We will explain here, different methods that have been developed in order to adapt these different clustering techniques to trajectory objects.

1.1.2.1 Model-based methods

The work of Gaffney et al.[Gaffney and Smyth, 1999] was the first attempt to cluster a set of trajectory data T^j where the number of measurements n^j of each trajectory data is different and also measured at different time indexations t^j. The goal of this work is to look for individuals of a population moving together where the deviation between the trajectories in a cluster is expressed as noise in a probabilistic formulation: every object is assumed to be drawn according to a mixture of k components. Assuming we know the parameter of a model θ and the time indexation t^j, the probability that trajectory T^j has been drawn from this model is

$$P(T^j|\theta_k, t^j) = \sum_k w_k f_k(T^j|\theta_k, t^j) = \sum_k w_k \prod_{i=1}^{n^j} f_k(T_i^j|\theta_k, t_i^j)$$

where w_k is the prior assumption for an object T^j to belonging to cluster k. The conditional density function $f_k(T_i^j|\theta_k, t_i^j)$ assumes a relation between T_i^j and t_i^j, e.g,

$$T_i^j = g_k(t_i^j) + eps_k$$

where g_k is a deterministic function with parameter θ_k, and $eps_{i,k}$ a Gaussian noise. In [Gaffney and Smyth, 1999], this function is a simple linear regression, but later [Gaffney, 2004] this model will be applied to more complex models such as polynomial regression, splines, kernel, etc... In a successive work [Chudova et al., 2003], spatial and (discrete) temporal shifting of trajectories within clusters is also considered and integrated as parameters of the mixture model. All the parameters, $\Theta = (\theta_{1,...k}, eps_{1,..k}, w_{1,..k})$ are estimated with an EM algorithm. This solution permits clustering trajectories of different sizes. In [Gaffney and Smyth, 1999], this algorithm is successfully applied to video stream clustering. These data can be easily fit with a simple model such as polynomial regression, splines, etc... because they are really smooth. Trajectories are network constrained objects. We can not fit a trajectory with a simple regression of a polynomial curves. Another model-based approach is presented in [Jonathan et al., 2003], where the representation of a cluster is not a trajectory but a Markov model that tries to explain the transitions between a position and the next, positions being discretized a priori. More exactly, Hidden Markov models (HMMs) are used to model clusters, and a mixture model approach and the Baum-Welch algorithm, in particular, is adopted for the parameter estimation task. Here again, this solution is limited by the temporal properties of vehicle trajectories which make time progression very irregular. Moreover, the performance of this method is strongly related to the precision of discretization of the space.

1.1.2.2 Density-based methods

Density-based methods appear like a natural choice when clustering network constrained trajectories because of the possibility of their intuitive geographical interpretation. How-

ever, algorithms such as *DBSCAN* or *OPTICS* can't directly be used on points that form a trajectory, there is no obvious density area which can be interpreted as being well delimited cluster. These algorithms have to be adapted to trajectories topology, i.e, find another variable which enables to delimit clusters. We present here three different frameworks which have been designed to cluster trajectory data based on density-based methods.

In [Lee et al., 2007], Lee et al. built a method called *TRACLUS*. They adapted the DBSCAN aglorithm to be applied to segments instead of points. Segments are objects more easily separable than points. The first step of the *TRACLUST* method is to make the segments that build a trajectory even more delimited. Each trajectory is modelled by its trajectory partition.

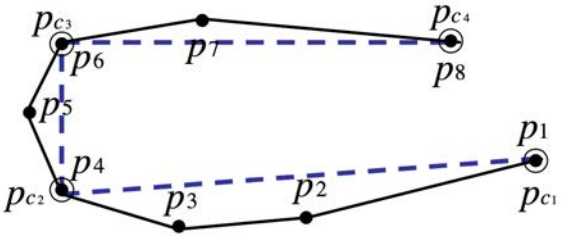


Figure 1.5 – Exemple of a trajectory (full-line) and its trajectory partition (hashed-line). Taken from [Lee et al., 2007].

The distance between segment, D_s is defined as a combination of three distances,

$$Ds(s_i, s_j) = w_\perp D_\perp(s_i, s_j) + w_{||} D_{||}(s_i, s_j) + w_\theta D_\theta(s_i, s_j),$$

where $D_\perp$, $D_{||}$ and D_θ are respectively the perpendicular, the parallel and the angular distance between two segments as defined in Figure 1.6.

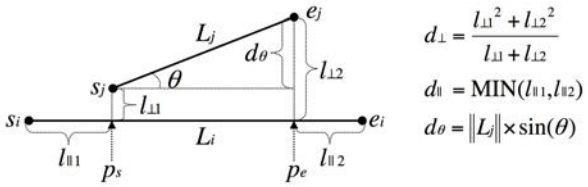


Figure 1.6 – Segment distance. Taken from [Lee et al., 2007].

An adapted DBSCAN algorithm is then applied on segments from the trajectory partition according to the segment distance D_s. The *TRACLUS* algorithm produces clusters of segments. A representative trajectory of a cluster is then produced for all the clusters. But it represents the overall movement of trajectory partitions an not the all of the trajectory. Its main advantage is the discovery of common sub-trajectories. Moreover, *TRACLUS*

supposes that the trajectory partition is well delimited, and that characteristic point is easily detectable, which is dependent on the trajectory's frequency.

Kalnis *et al.*[Kalnis et al., 2005] considered the problem of discovering density-based spatial clusters that persist along several contiguous time slices. Persistence of a cluster means that the locations contained in a cluster at a given time interval are the same to those that appear in a cluster in the next time interval.

In [Nanni and Pedreschi, 2006], Nanni *et al.* looked for groups of objects that move together within some time interval. Here, trajectories are clustered by the *OPTICS* algorithm where the distance is the mean of the Euclidean distance between the trajectories within a given time interval. Then, for each time interval T, a clustering is produced, on the subtrajectory laying within T. This algorithm aimed at supporting efficient range queries which are the core operation of the OPTICS algorithm. Rinzivillo and al. [Rinzivillo et al., 2008] describe a visually–driven analysis also based on the characteristics of the *OPTICS* algorithm. They first define several basic distance functions that compare the trajectories according to various spatial and spatio-temporal data. For example, the distance between the origin, the destination, or the average of the origin and the destination of the locations of two trajectories. Spatio-temporal distances compare distance at different check-points, or look for the closest pair of positions in two trajectories at given time. Then, a progressive clustering is applied. No strict framework is given. A user can, for example, start by clustering trajectories based on their origins, and after by their destinations and finally by their time index. Each of these clustering steps needs a visual analysis before going further. Both these methods, [Nanni and Pedreschi, 2006] and [Rinzivillo et al., 2008] seem a natural choice when dealing with density-based methods. Indeed, this method enables us to delimit data according to a give parameter one by one, which enables us to find clearly delimited clusters. But the choice of $MinPts$ and ϵ are strongly dependent on the dataset we explore. And their choice requires us to know the dataset we are dealing with. But it is impossible to apply this method to different datasets in a generic way.

Kim et al. [Kim and Mahmassani, 2015] propose a four steps framework to cluster trajectory determination and classification of new trajectories. The two first steps enable one to cluster trajectories. First, similarities are measured between all trajectory. An adapted Longest Common Subsequence (*LCSS*) is used. *LCSS* is a warping distance, which counts the number of overlap in a given area limit (see section 1.2.3.1). The distance defined by Kim et al. returns a percentage of common coverage between two trajectories. If the trajectories are going to opposite directions, their coverage is zero. DBSCAN is then applied to this distance. DBSCAN clusters points based on their density area. DBSCAN, based on this adapted *LCSS* distance, clusters trajectory based on the portion on route they share together. The results of this approach is a few spatially distinct traffic clusters which represent the main traffic stream in a given area. They have clearly separable origins from one another. Once again, the results of this cluster are strongly dependant of the parameters of the limit distance for both *LCSS* distance and DBSCAN cluster algorithm.

The density-based methods presented above, used to cluster trajectories share common characteristics. They define a new way, different from the points density method, to clearly delimit clusters, based on segment distance [Lee et al., 2007], percentage coverage[Kim and Mahmassani, 2015] , common origin [Rinzivillo et al., 2008] *etc...* These methods are effective because they respect the geometry characteristic of the trajectory. However, they require a prior knowledge of the studied dataset for the choice of the ϵ parameter. They are hardly adjustable to different datasets with different characteristics, without preliminary study.

1.1.2.3 Distance-based method

We have seen in the previous section methods that attempt to adapt existing clustering methods to fit the characteristics of trajectory data. Another solution is to directly apply the clustering methods detailed in section 1.1.1 without any modifications. This requires that the distance characteristics of trajectory data has to be completely encapsulated in the distance function. Therefore, the choice of the distance function is primordial. Defining a distance between trajectories implicitly determine which trajectory should be part of the same cluster and hence, which kind of clusters we are going to discover. For example [Vasquez and Fraichard, 2004] and Hu et al. [Hu et al., 2006] both apply classical clustering techniques, hierarchical clustering [Vasquez and Fraichard, 2004] or fuzzy k-means algorithm [Hu et al., 2006]. For that they use simple distance. Hu et al. use euclidean distance. Vasquez et al. use *partial distance*:

$$d_{partial}(T_c^j, T_c^k) = \left(\frac{1}{\Delta_{i,j}} \int_{t=0}^{\Delta_{i,j}} \|T_c^j(t) T_c^k(t)\|_2 dt \right)^{\frac{1}{2}},$$

where $\Delta_{i,j} = \max(\Delta_i, \Delta_j)$ This distance suppose that trajectories are continuous, or at least that both have the same indexation. In both [Vasquez and Fraichard, 2004] and [Hu et al., 2006], these clustering methods are applied on video stream clustering, with high frequency indexing and smooth trajectories, for the same reason as that for model base clusters approach, can hardly be adapted to network constrained trajectories. Taking into account both the time and the space in the same definition of a distance is impossible due to to the strong inconsistency of the trajectory data from vehicle trip.

For theses reasons we focus on distances which can compare the shape of the trajectory in order to be able to cluster together the trajectories with similar behaviour on the same road. This problematic is tackle in the following Section, "Review and Perspective for Distance Based Clustering of Vehicle Trajectories". This work has been published in the IEEE Transactions on Intelligent Transportation Systems journal [Besse et al., 2016]

1.2 Review and perspective for distance based clustering of vehicle trajectories

1.2.1 Introduction

A trajectory is a set of positional information for a moving object, ordered by time. This kind of multidimensional data is prevalent in many fields and applications, for example, for understanding migration patterns through studying trajectories of animals, predicting meteorology with hurricane data, improving athletes performance, etc. Our study concentrates on vehicle trajectories within a road network. The growing use of GPS receivers and WIFI-embedded mobile devices equipped with hardware for storing data enables the collection of an enormous amount of data that can be used to extract relevant information in order, for instance, to find the optimal path to go from point A to point B, detect abnormal behavior, optimize the traffic flow in a city, predict the next location or final destination of a moving object *etc.* This aims actually to build from the data the different features that characterize the different daily movement of the vehicles on the road network. For this, we consider clustering methods for trajectories. Clustering techniques aim to regroup similar trajectories together into groups that are different from one another. The complexity of trajectory makes this a challenging task as objects can move along many different paths in a given area, moreover the road network of a highway does not have the same complexity as that of a city, and finally the road network in a city differs between the suburbs and downtown. In addition, the speed of an object varies between regions, and between paths taken within a single region. Even within the same path the speed depends on exogenous variables such as the time of day, or whether it is a weekday or the weekend.

Several methods can be used to cluster trajectories. We will focus on distance based trajectory clustering but other specific methodologies have been investigated. Dealing with the functional properties of trajectories considered as continuous function of time Gaffney (2009, [Gaffney and Smyth, 1999]), Vasquez *et al.* (2004[Vasquez and Fraichard, 2004]), Hu *et al.* (2006[Hu et al., 2006]) successfully apply trajectory clustering methods on video-stream trajectories. Gariel et al. [Gariel et al., 2011] also use the continuous definition of trajectory to re-sample the trajectories and obtain time series of equal length. A principal components analysis is then applied on these new trajectories to obtain principal components and finally cluster them. This method is applied to airplane routes. All these methods take into account both the spatial and the temporal aspect of the trajectory, they are not adapted to the vehicle trajectories constrained to a road network whose time progression is very irregular. Rinzivillo *et al.* (2008,[Rinzivillo et al., 2008]) and Kim *et al.*[Kim and Mahmassani, 2015]) propose density-based methods. Both require the definition of a density parameter and a minimum cluster size which implies an extensive knowledge of the studied area or a precise question to obtain good results. Hence, these

methods are hard to automatically adapt from one dataset to another. Finally Lee *et al.* (2007,[Lee et al., 2007]) and Wu *et al.* (2014[Wu et al., 2013]) propose to use clustering methods on trajectory line segments to enable the detection of important areas of flow, though this does not consider the trajectory as a whole path. Our main objective here is to detect the main path and traffic flow which can later be used to study the different behaviors along these paths.

In this context, the goal of this work is to construct, in a data driven way, a collection of trajectories that model the behaviors of car drivers. These models are learned from a data set of car locations. In this work we focus on clustering trajectories having similar paths. This clustering is based on the comparison between trajectory objects, and as such a new definition of distance between the studied trajectory objects is required.

A large amount of work has been done to give new definitions of trajectory distance. Tiakas *et al.*(2009[Tiakas et al., 2009]) , Rossi *et al.* (2012[El Mahrsi and Rossi, 2012]), Han *et al.*(2015[Han et al., 2015]) or Hwang *et al.*(2005[Hwang et al., 2005a]) propose road network based distances. They assume that the trajectories studied are perfectly mapped on the road network. However, this task is strongly dependent on the precision of the GPS device. When the time interval between two GPS locations is significant, several paths on the graph are possible between locations, especially when the network is dense. Moreover it requires the knowledge of the road network. Here, we focus on entirely data driven methods without any *a priori* information. Several methods have been used to cluster data set of trajectories. Clustering methods using Euclidean distance lead to inaccurate results mainly because trajectories have different lengths. Hence, several methods based on warping distance have been defined , Berndt (1994[Berndt and Clifford, 1994]), Vlachos *et al.* (2002[Vlachos et al., 2002]), Chen *et al.* (2004[Chen and Ng, 2004]), and Chen *et al.* (2005 [Chen et al., 2005]). These methods reorganize the time index of trajectories to obtain a perfect match between them. Another approach is to focus on the geometry of the trajectories, in particular their shape. Shape distances like Hausdorff and Fréchet distances can be adapted to trajectories but fail to compare them as a single entity. Lin *et al.* (2005[Lin and Su, 2005]) proposed a method based exclusively on the shape of the trajectory but at high computational cost.

In section 1.2.2 the paper's definitions, notations and problem statement are introduced. In section 1.2.3 several distances on trajectory are studied and compared. A new distance will be presented in section 1.2.4: the Symetrized Segment-Path Distance (SSPD). SSPD is a shape-based distance that does not take into account the time index of the trajectory. It compares trajectories as a whole, and is less affected by incidental variation between trajectories. It also takes into account the total length, the variation and the physical distance between two trajectories. For all these different distances, we obtain different clusterings. So we can compare the distance on these results. The choice of clustering used is detailed section 1.2.5. The SSPD and the other studied distances were implemented in a python package, *trajectory_distance* available on github. The presenta-

tion of this package and the experimental evaluation of these distances with the chosen clustering techniques on some trajectory sets are analyzed in section 1.2.6.

1.2.2 Model for trajectory clustering

1.2.2.1 Trajectory

A continuous trajectory is a function which gives the location of a moving object as a continuous function of time. In our case we will only consider discrete trajectories defined here after.

Definition 8. *A trajectory T is defined as*
$T : ((p_1, t_1), \ldots, (p_n, t_n))$,
where $p_k \in \mathbb{R}^2, t_k \in \mathbb{R}\ \forall k \in [1 \ldots n],\ \forall n \in \mathbb{N}$ *and n is the length of the trajectory T.*

The exact locations between time t_i and t_{i+1} are unknown. When these locations are required, a piece wise linear representation is used between each successive location p_i and p_{i+1} resulting in a line segment s_i between these two points. This new representation is called a piece wise linear trajectory. In this representation, no assumption is made about time indexing of segment s_i.

Definition 9. *A piece wise linear trajectory is defined as* $T_{pl} : ((s_1), \ldots, (s_{n-1}))$ *, where* $s_k \in \mathbb{R}^4$ *and* n_{pl} *is the length of the trajectory.*

The length of the trajectory n_{pl} is the sum of the lengths of all segments that compose it : $n_{pl} = \sum_{i \in [1 \ldots n-1]} \|p_i p_{i+1}\|_2$.

The notation used in this paper are summarized in Table 1.2.

1.2.2.2 Distance

There are many ways to define how close two objects are from one another. Beyond the notion of mathematical distance, many functions can be used to qualify this dissimilarity. The terminology used in literature to define them is not completely standardized. Therefore we will use the definition established in Deza *et al.* (2009[Deza and Deza, 2009]) as a reference.

Definition 10. *Let $\mathcal{T}$ be a set of trajectories. A function $d : \mathcal{T} \times \mathcal{T} \mapsto \mathcal{R}$ is called a dissimilarity on $\mathcal{T}$ if for all $T^1, T^2 \in \mathcal{T}$:*

- $d(T^1, T^2) \geq 0$
- $d(T^1, T^2) = d(T^2, T^1)$
- $d(T^1, T^1) = 0$

If all of these conditions are satisfied and $d(T^1, T^2) = 0 \implies T^1 = T^2$ *d is considered to be a symmetric. If the triangle inequality is also satisfied, d is called a metric. These notations are summarized in Table 1.3.*

X indicates the required properties for each distances, while $*$ indicates properties that are automatically satisfied (by the presence of the other required properties for the metric).

1.2.2.3 Desired properties of clustering and distances

Our aim is to regroup trajectories sharing similar behaviour. We want that trajectories in the same cluster, take similar paths. Hence our goal is to define a clustering method that will regroup trajectories

- with similar shape and length
- which are physically close to each other
- which are similar as a whole with more than just similar sub-parts
- all of these properties should be considered without regard to their time indexing

Moreover we want to design a very general procedure which is able to treat all trajectories data, without prior knowledge of the particular geographical location where they are collected. To obtain such clustering, the goal of this work is to find a distance that respects such properties and to succeed in extracting these features. Actually, the desired distance should have the following properties,

Table 1.2 – Notation for the study of trajectory distance.

$\mathcal{T}$	The set of trajectories
T^i	The i^{th} trajectory of set $\mathcal{T}$
T^i_{pl}	The piece wise linear representation of T^i
n^i	Length of trajectory T^i
n^i_{pl}	Length of the T^i_{pl}
p^i_k	The k^{th} location of T^i
p^i_{pl}	The set of continuous points that compose T^i_{pl}
s^i_k	The line segment between p^i_j and p^i_{k+1}
t^i_k	The time index of location p^i_k
$\|p_k p_l\|_2$	The Euclidean distance between p_k and p_l

Table 1.3 – Metric Definition

Property \ Metric Name		*dissimilarity*	*symmetric*	*metric*
Non-Negativity	$D(T^1, T^2) \geq 0$	X	X	$*$
Symmetry	$D(T^1, T^2) = D(T^2, T^1)$	X	X	X
Reflexivity	$D(T^1, T^1) = 0$	X	$*$	$*$
Triangle Inequality	$D(T^1, T^3) \leq D(T^1, T^2) + D(T^2, T^3)$			X
Identity of indiscernible	$D(T^1, T^2) = 0 \implies T^1 = T^2$		X	X

- it compares trajectories as a whole
- the compared trajectories can be of different lengths,
- the time indexing can be very different from one trajectory to another
- the trajectories can have similar shapes but can be physically far from each other and vice versa
- extra parameters should not be required.

1.2.3 Distance on trajectories: a review

Three main kind of distances have been introduced in the literature. The first uses the underlying road network, **Network-Constrained Distance**. These distances will not be detailed in this paper. They assume that the road network is known and that trajectory data are perfectly mapped on it. Distances that do not use the underlying road network can also be classified into two categories: those who only compare the shape of the trajectory, **Shape-Based Distance** and those who take into account the temporal dimension, **Warping based Distance**.

Performance of clustering algorithms using these distances will be compared in section 1.2.6, as well as their computation cost and their metric properties.

1.2.3.1 Warping based Distance

Euclidean distance, Manhattan distance or other L^p-norm distances are the most obvious and the most often used distances. They compare discrete objects of the same length. They can be used to look for common sub-trajectories of a given length but they cannot be used to compare entire trajectories. Moreover, these distances will compare locations with common indexes one by one. At a given index i, location p_i^1 of trajectory T^1 will be compared only to location p_i^2 of trajectory T^2. However, these locations can be strongly

different according to the speeds of the trajectories. Hence, it makes no sense to compare them without taking this into account. This problem is also common in time series analysis and not restricted to trajectory analysis.

Warping distance aims to solve this problem. To accomplish this, they enable matching locations from different trajectories with different indexes. Then, they find an optimal alignment between two trajectories, according to a given cost δ between matched location. Several warping based distances have been defined. *DTW* (Berndt *et al.*, (1994 [Berndt and Clifford, 1994])) and later *LCSS* (Vlachos *et al.*, 2002[Vlachos et al., 2002]), *EDR* (Chen *et al.*, 2005[Chen et al., 2005]) and *ERP* (Chen *et al.*, 2005[Chen and Ng, 2004]). These distances are defined the same way, but they use different cost functions.

In order to define a warping distance, two compared time series trajectories, T^i, T^j, are arranged to form a $n^i \times n^j$ grid G. The grid cell, $g_{k,l}$, corresponds to the pair (p^i_k, p^j_l).

Definition 11. *A warping path, $W = w_1, \ldots, w_{|W|}$, crosses the grid G such that*

- $w_1 = g_{1,1}$,
- $w_{|W|} = g_{n^i,n^j}$,
- *if $w_k = g_{k_i,k_j}$, then w_{k+1} is equal to g_{k_i+1,k_j}, g_{k_i,k_j+1} or g_{k_i+1,k_j+1}.*

The order of the locations in a trajectory are maintained but they can be repeated, deleted or replaced by an arbitrary value, a *gap*, along the warping path. The distance is then computed by minimizing or maximizing the sum of a given cost δ between all pair of locations that make a warping path W, for all existing warping paths.

Definition 12. *A warping distance is defined as*

$$\begin{aligned} D(T^i, T^j) = &\quad \min_W \left[\sum_{k=1}^{|W|} \delta(w_k) \right], \\ or &= \max_W \left[\sum_{k=1}^{|W|} \delta(w_k) \right], \end{aligned} \tag{1.3}$$

where $\delta(w_k) = \delta(g_{k_i,k_j}) = \delta(p^i_{k_i}, p^j_{k_j})$, is the cost function and W is a warping path.

They are generally computed by dynamic programming. Table 1.4 displays the cost functions as well as the dynamic formulation of these distances.

Contrary to the three other distances, *LCSS* is a similarity. The exact similarity used in Vlachos *et al.*, 2002[Vlachos et al., 2002] is $S(T^i, T^j) = \frac{LCSS(T^i,T^j)}{\min\{n^i,n^j\}}$, which is between 0 and 1. We will then use the distance

$$DLCSS(T^i, T^j) = 1 - S(T^i, T^j),$$

to compare distances to each other.

The metric types of these distance functions, and computational cost for the four methods are summarized in table 1.5.

Table 1.4 – Re-Indexing based distance definition

NAME	Cost function $\delta_{NAME}(p_1, p_2) =$	Distance $NAME(T^i, T^j) =$
DTW	$\|p_1p_2\|_2$	$= \begin{cases} 0 & \text{if } n^i = n^j = 0 \\ \infty & \text{if } n^i = 0 \text{ or } n^j = 0 \\ \delta_{DTW}(p_1^i, p_1^j) + \min \left\{ \begin{array}{l} DTW(rest(T^i), rest(T^j)), \\ DTW(rest(T^i), T^j), \\ DTW(T^i, rest(T^j)) \end{array} \right\} & \text{otherwise} \end{cases}$
LCSS	$\begin{cases} 1 & \text{if } \|p_1p_2\|_2 < \varepsilon_d \\ 0 & \text{if } p_1 \text{ or } p_2 \text{ is a gap} \\ 0 & \text{otherwise} \end{cases}$	$= \begin{cases} 0 & \text{if } n^i = 0 \text{ or } n^j = 0 \\ LCSS(rest(T^i), rest(T^j)) + \delta_{LCSS}(p_1^i, p_1^j) & \text{if } \delta_{LCSS}(p_1^i, p_1^j) = 1 \\ \max \left\{ \begin{array}{ll} LCSS(rest(T^i), T^j) & + \ \delta_{LCSS}(p_1^i, gap), \\ LCSS(T^i, rest(T^j)) & + \ \delta_{LCSS}(gap, p_1^j) \end{array} \right\} & \text{otherwise} \end{cases}$
EDR	$\begin{cases} 0 & \text{if } \|p_1p_2\|_2 < \varepsilon_d \\ 1 & \text{if } p_1 \text{ or } p_2 \text{ is a gap} \\ 1 & \text{otherwise} \end{cases}$	$= \begin{cases} n^i & \text{if } n^j = 0 \\ n^j & \text{if } n^i = 0 \\ EDR(rest(T^i), rest(T^j)) & \text{if } \delta_{EDR}(p_1^i, p_1^j) = 0 \\ \min \left\{ \begin{array}{ll} EDR(rest(T^i), rest(T^j)) & + \ \delta_{EDR}(p_1^i, p_1^j), \\ EDR(rest(T^i), T^j) & + \ \delta_{EDR}(p_1^i, gap), \\ EDR(T^i, rest(T^j)) & + \ \delta_{EDR}(gap, p_1^j) \end{array} \right\} & \text{otherwise} \end{cases}$
ERP	$\begin{cases} \|p_1p_2\|_2 & \text{if } p_1, p_2 \text{ are not gaps} \\ \|p_1g\|_2 & \text{if } p_2 \text{ is a gap} \\ \|gp_2\|_2 & \text{if } p_1 \text{ is a gap} \end{cases}$	$= \begin{cases} \sum_{k=1}^{n^i} \|p_k^i g\|_2 & \text{if } n^j = 0 \\ \sum_{l=1}^{n^j} \|p_l^j g\|_2 & \text{if } n^i = 0 \\ \min \left\{ \begin{array}{ll} ERP(rest(T^i), rest(T^j)) & + \ \delta_{ERP}(p_1^i, p_1^j), \\ ERP(rest(T^i), T^j) & + \ \delta_{ERP}(p_1^i, gap), \\ ERP(T^i, rest(T^j)) & + \ \delta_{ERP}(gap, p_1^j) \end{array} \right\} & \text{otherwise} \end{cases}$

Comparisons

- All of these distances handle local time shifting.
- The cost function δ uses the Euclidean distance. Some of these distances have been defined using a L1-norm, but Euclidean distance is more adapted for real values.
- $LCSS$ and EDR's cost function count the number of occurrences where the Euclidean distance between matched location does not match a spatial threshold, ε_d. The former counts similar locations, the latter the difference. This threshold makes the distance robust to noise. However, it has a strong influence on the final results. If the threshold is large, all the distances will be considered similar and if low, only those having very close locations will be considered similar.
- In comparison, ERP and DTW add a weighting to these differences by computing the real distance between the locations. In this sense they can be viewed as more accurate.
- ERP is the only distance which is a *metric* regardless of the L_p norm used, yet it works better for normalized sequences, especially for defining the gap value g. It does not apply for vehicle trajectories.

Table 1.5 – Re-Indexing based distance properties

Name	Metric Types	Computation Cost
DTW	*symmetric*	$O(n^2)$
LCSS	*distance*	$O(n^2)$
EDR	*symmetric*	$O(n^2)$
ERP	*metric*	$O(n^2)$

- In addition, these distances may include a time threshold, ε_t. Thus, two locations will not be compared if the difference between their time indexing is too large. However, it is very hard to estimate the value of this threshold when comparing trajectories due to the presence of noise.

Pros and Cons The main advantage of these distances is that they enable comparison of sequences of different lengths.

The two main limitations of warping based distance are the following

- Warping methods are based on one-to-one comparison between sequences. Hence, it often requires the choice of a particular series that will be used as a reference, onto which all other sequences will be mapped. The index of two sequences being compared should be well balanced in order to best capture the variability, for instance, in order to detect if there were accelerations and decelerations during the measurement of the time series. Hence the choice of the reference sequence is very important.

- The performance of the usual methods based on warping techniques is hampered by the large amount of noise inherent to road traffic data, which is not the case when examining time series.

Instead of correcting the time index, the solution is to use distances that have the effect of time removed.

1.2.3.2 Shape-based distance

These distances try to catch geometric features of the trajectories, in particular, their shape. Among **Shape-Based Distances**, the Hausdorff distance (Hausdorff, 1914 [Hausdorff, 1914]), and the Fréchet distance (Fréchet, 1906[Fréchet, 1906]) are likely the most well known.

Hausdorff The *Hausdorff* distance is a *metric*. It measures the distance between two sets of metric spaces. Informally, for every point of set 1, the infimum distance from this point to any other point in set 2 is computed. The supremum of all these distances defines the *Hausdorff* distance.

Definition 13. *The Hausdorff distance between two sets of metric spaces is defined as*

$$Haus(X,Y) = \max\{\sup_{x\in X}\inf_{y\in Y}\|xy\|_2, \sup_{y\in Y}\inf_{x\in X}\|xy\|_2\}.$$

This distance is complicated and resource intensive to compute when applied to most existing sets. But in the case of polygonal curves like trajectories, some simplification can be made due to the monotonic properties of a segment. Distance from a point p to a segment s is defined as follows.

Definition 14. *Point − to − Segment distance.*

$$D_{ps}(p^1_{i_1}, s^2_{i_2}) = \begin{cases} \|p^1_{i_1}p^{1proj}_{i_1}\|_2 & \text{if } p^{1proj}_{i_1} \in s^2_{i_2}, \\ \min(\|p^1_{i_1}p^2_{i_2}\|_2, \|p^1_{i_1}p^2_{i_2+1}\|_2) & \text{otherwise.} \end{cases}$$

Where $p^{1proj}_{i_1}$ is the orthogonal projection of $p^1_{i_1}$ on the segment $s^2_{i_2}$.

Hence, the *Hausdorff* distance between two line segments is

$$\begin{aligned} D_{Haussdorf}(s^1_{i_1}, s^2_{i_2}) = \max\{ & \sup_{p\in s^1_{i_1}} D_{ps}(p, s^2_{i_2}), \\ & \sup_{p\in s^2_{i_2}} D_{ps}(p, s^1_{i_1})\} \\ = \max\{ & D_{ps}(p^1_{i_1}, s^2_{i_2}), D_{ps}(p^1_{i_1+1}, s^2_{i_2}), \\ & D_{ps}(p^2_{i_2}, s^1_{i_1}), D_{ps}(p^2_{i_2+1}, s^1_{i_1})\}. \end{aligned}$$

Indeed, a segment is monotonic. As seen in Fig. 1.7, the supremum of the *Point − to − Segments* distance from any point of a segment $s^1_{i_1}$ to a segment $s^2_{i_2}$ occurs at one of the end points of the segment $s^1_{i_1}$. The *Hausdorff* distance between two trajectories can then be computed with the following formula.

Definition 15. *Hausdorff distance between two discrete trajectories.*

$$\begin{aligned} D_{Haussdorf}(T^1, T^2) = \max\{ & \max_{\substack{i_1\in[1\ldots n^1] \\ j_2\in[1\ldots n^2-1]}} \{D_{ps}(p^1_{i1}, s^2_{j2})\}, \\ & \max_{\substack{j_1\in[1\ldots n^1-1] \\ i_2\in[1\ldots n^2]}} \{D_{ps}(p^2_{i2}, s^1_{j1})\}\}. \end{aligned}$$

The *Hausdorff* distance can then be computed in a $O(n^2)$ computational time.

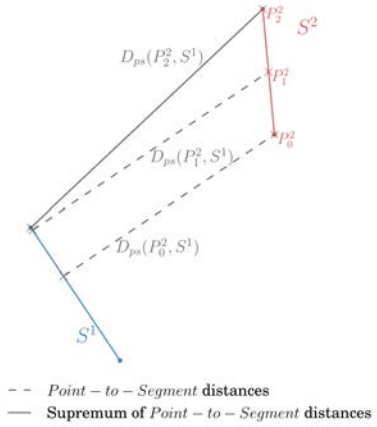


Figure 1.7 – Supremum of $Point - to - Segment$ distance from point of segment s_1^1 to segment s_1^2.

Frechet and discrete Fréchet The *Frechet* distance measures similarity between curves. It is often known as the "walking-dog distance". Imagine a dog and its owner walking on two separate paths without backtracking from one endpoint to one other. The Fréchet distance is the minimum length of leash required to connect a dog and its owner. While the Hausdorff distance takes distance between arbitrary points, the Fréchet metric takes the form of the two curves into account.

Definition 16. *The Fréchet distance between two curves is defined as*

$$D_{Frechet}(A, B) = \inf_{\alpha,\beta \in X} \max_{t \in [0,1]} \left\{ \|A\Big(\alpha(t)\Big), B\Big(\beta(t)\Big)\|_2 \right\}.$$

Similar to the *Hausdorff* distance, the *Frechet* distance is a *metric*. It is also resource intensive. Alt *et al.* (1995[Alt and Godau, 1995]) developed an algorithm measuring the exact Fréchet distance for polygonal curves based on the free space definition.

Definition 17. *The free space $F_\epsilon(T^1, T^2)$ between two trajectories is the set of all pairs of points whose distance is at most ϵ.*

$$F_\epsilon(T^1, T^2) := \{(p^1, p^2) \in (T^1, T^2)\} | \|p^1, p^2\|_2 \leq \epsilon\}.$$

The *Fréchet distance* between two trajectories T^1 and T^2 is the minimum value of ϵ for which a curve exists within the corresponding F_ϵ from (p_0^1, p_0^2) to $(p_{n^1}^1, p_{n^2}^2)$ with the property of being monotone existing in both trajectories. Computing the *Fréchet distance* means finding the minimum value of ϵ. By exploiting the monotonic property of the segments and the definition of free space, this task can be accomplished more efficiently.

Indeed, the *Frechet* distance between segments is equal to the *Hausdorff* distance between segments, *i.e.*

$$\begin{aligned} D_{Frechet}(s^1_{i_1}, s^2_{i_2}) &= \max\{ \quad D_{ps}(p^1_{i_1}, s^2_{i_2}), \\ &\qquad\quad D_{ps}(p^1_{i_1+1}, s^2_{i_2}), \\ &\qquad\quad D_{ps}(p^2_{i_2}, s^1_{i_1}), \\ &\qquad\quad D_{ps}(p^2_{i_2+1}, s^1_{i_1})\} \\ &= \epsilon_{i_1,i_2}. \end{aligned}$$

To compute the *Frechet* distance between trajectories T^1 and T^2 , we need only look among the set E of *Frechet* distances between all pairs of segments of T^1 and T^2. $E = \{\epsilon_{i_1,i_2}$ for $(i_1, i_2) \in ([1 \dots n^1-1] \times [1 \dots n^2-1])\}$. This simplification enables us to compute the *Frechet* distance between trajectories T^1 and T^2 in $O(n^2 log(n^2))$. We highlight that this computational cost is higher than all the other calculation methods of the studied distances.

Eiter *et al.* (1994[Eiter and Mannila, 1994]) describes an approximation of this distance for polygonal curves called the *discrete Fréchet* distance. This distance is close to the definition of the warping based distance.

Definition 18. *The discrete Fréchet distance is defined as*

$$D_{Frechet_{Discr}}(T^1, T^2) = \min_W \{ \max_{k \in [1 \dots |W|]} \|w_k\|_2 \}.$$

with W being the warping path defined in definition 12. The *discrete Fréchet* distance can be computed in $O(n^2)$ time.

This distance is bounded as follows.

Theorem 1. *For any trajectories T^i and T^j [Eiter and Mannila, 1994]*

$$D_{Frechet}(T^i, T^j) \leq D_{Frechet_{Discr}}(T^i, T^j) \leq D_{Frechet}(T^i, T^j) + \epsilon$$

Where, $\epsilon = \max\{ \max_{k \in [1 \dots n^i - 1]} \{\|p^i_k p^i_{k+1}\|_2\}, \max_{l \in [1 \dots n^j - 1]} \{\|p^j_l p^j_{l+1}\|_2\}\}$.

One Way Distance Lin *et al.* 2005[Lin and Su, 2005] defines the *One-Way-Distance*, *OWD*, from a trajectory T^i to another trajectory T^j as the integral of the distance from points of T^i_{pl} to trajectory T^j_{pl} divided by the length of T^i_{pl}

Definition 19. *The OWD distance is defined as*

$$D_{OWD}(T^i, T^j) = \frac{1}{n^i_{pl}} \int_{p^i \in T^i_{pl}} D_{point}(p^i T^j) dp^i,$$

Table 1.6 – Shape based distance properties

Name	Metric Types	Computation Cost
Hausdorff	*metric*	$O(n^2)$
Frechet	*metric*	$O(n^2 log(n^2))$
discrete Fréchet	*symmetric*	$O(n^2)$
OWD	*symmetric*	$O(n^2 log(n))$
OWD_{grid}	*symmetric*	$O(mn)$

where $D_{point}(p, T)$ is the distance from the point p to the trajectory T so that

$$D_{point}(p, T) = \min_{q \in T_{pl}} \|pq\|_2.$$

The *OWD* distance is not symmetric, but $D_{SOWD}(T^i, T^j) = (D_{OWD}(T^i, T^j) + D_{OWD}(T^j, T^i))/2$ is. This distance is a *symmetric* because it does not satisfy the triangle inequality.

Lin *et al.*[Lin and Su, 2005], have defined two algorithms to compute the *OWD* in case of piecewise linear trajectories.

- The first consists of finding the parametrized OWD function $D_{OWD}(s_k^i, T^j)$ from a segment s_k^i of T_{pl}^i to all segments s^j of T_{pl}^j and for all segments of T_{pl}^i

$$D_{OWD}(T^i, T^j) = \frac{1}{n_{pl}^i} \sum_{k=1}^{n^i - 1} D_{OWD}(s_k^i, T^j).\|p_k^i p_{k+1}^i\|,$$

  with a complexity of $O(n^2 log(n))$.

- The second uses a grid representation of the trajectory. As we see in Fig. 1.8, the space is discrete Trajectories are defined as the succession of grids they have crossed.

  Definition 20. *A grid representation trajectory is defined as*

$$T_{grid} := (g_0, \ldots, g_{n_{grid}}),$$

  where g_n are cells of the discrete space.

  This representation simplifies the computation and reduces the complexity to $O(nm)$ where m is the number of local min points. Local min points of a grid cell g are the grids with distances to g shorter than those of their neighbors' grid cell.

Table 1.6 displays the metric types and the computational cost of these distances.

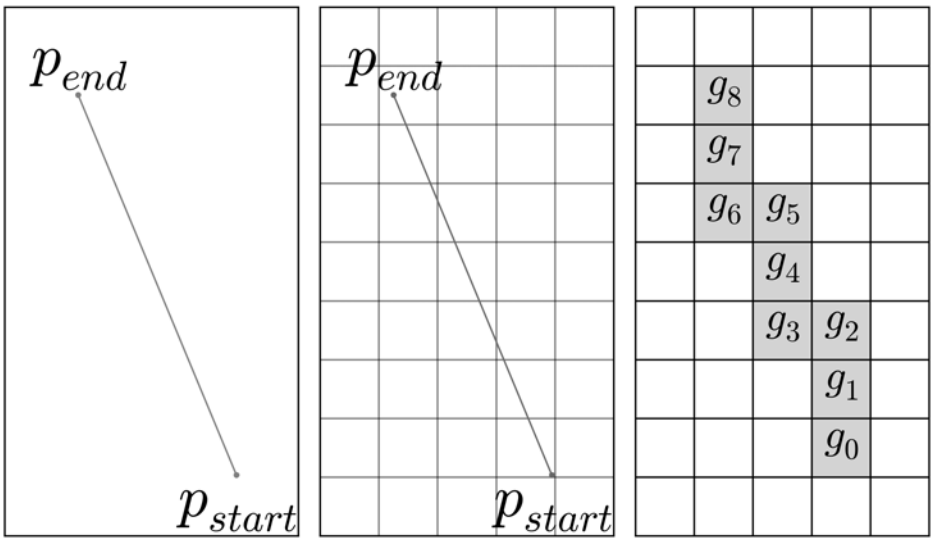


Figure 1.8 – Grid representation of a segment

Pros and Cons

- *Frechet* and *Hausdorff* distances are both *metrics*, meaning they satisfy triangular inequality. With clustering algorithms like *dbscan* or *K-medoid* this is a necessary property of the distance used if we want the clustering algorithm to be efficient. They have been widely used in many domains where shape comparison is needed. But they can fail to compare trajectories as a whole. Indeed both *Fréchet* and *Hausdorff* distance return a maximum distance between two objects at given points within the two objects. As we can see in Fig. 1.9, despite the fact that the trajectories T^1 and T^2 are well separated at the maximum value of x, they are clearly more similar to each other than to T^3. But with a *Hausdorff* calculated distance, there are no strong differences between $D_{Haussdorf}(T^1, T^2) = 3.26$, $D_{Haussdorf}(T^1, T^3) = 3.02$ and $D_{Haussdorf}(T^2, T^3) = 3.5$. With Frechet, $D_{Frechet}(T^1, T^2) = 6$ is even bigger than both $D_{Frechet}(T^1, T^3) = 4.19$ and $D_{Frechet}(T^2, T^3) = 4.17$.

- The *Discrete Fréchet* distance requires considerably less computing time compared to the *Frechet* distance. But *Discrete Frechet* is not a *metric*. Moreover, due to its similarity with the warping distance it shares the same inconveniences.

- The distance present in Lin *et al.* (2005[Lin and Su, 2005]) is by far the one that

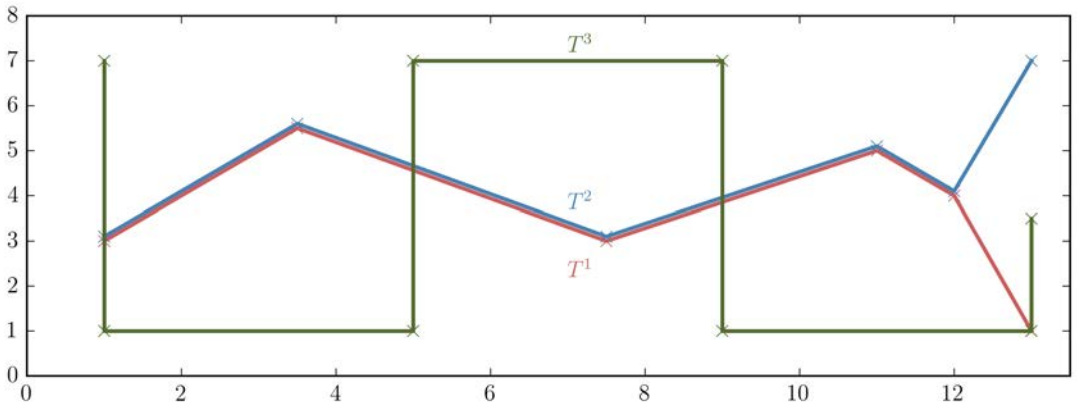


Figure 1.9 – Frechet And Hausdorff Computation between three trajectories

best meets our requirements. It compares trajectories as a whole, taking into account their shapes and their physical distances, the required features for our distance. However, its complexity makes it computationally slow. The algorithm for grid representation is faster. Its computational time is $O(mn)$. Yet it does not take into account the computation time required for matching the trajectory to the grid. Moreover, the size of the grid chosen strongly influences the final result and makes it imprecise. Furthermore, the distance gives the same "weight" to all points defining the trajectory: points directly issued from the GPS location, and points which compose the piece wise linear representation. The greater the length of the segment s is, the stronger its influence on the trajectory is. The more separated the endpoints of a segment s are, the less confident the interpolation between them is.

In the following section, a new distance will be established inspired from both the OWD and the Hausdorff distances.

1.2.4 A new distance : Symmetrized Segment-Path Distance (SSPD)

In this section, we define a new shape based distance, the *Symmetrized Segment-Path Distance*, and we compare it to other shape based distances. We propose *SSPD* in order to fulfill the desired properties defined in section 1.2.2.3.

Like the Hausdorff distance (Definition 15), the definition of *SSPD* is based on the $Point - to - Segment$ distance (Definition 14), D_{pt}, from a point p to a trajectory T. It is the minimum of distances between this point and all segments s that compose T. The *Segment-Path distance* from trajectory T^1 to trajectory T^2 is the mean of all distances from points composing T^1 to the trajectory T^2

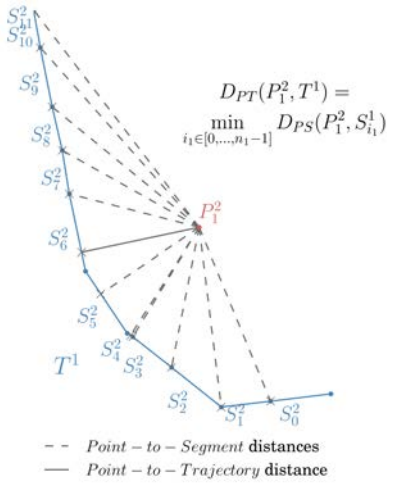


Figure 1.10 – Distance from point p_1^2 to trajectory T^1

Definition 21. *SPD distance is defined as*

$$D_{SPD}(T^1, T^2) = \frac{1}{n_1} \sum_{i_1=1}^{n_1} D_{pt}(p_{i_1}^1, T^2).$$

where, $D_{pt}(p_{i_1}^1, T^2) = \min_{i_2 \in [0,\ldots,n_2-1]} D_{ps}(p_{i_1}^1, s_{i_2}^2)$.

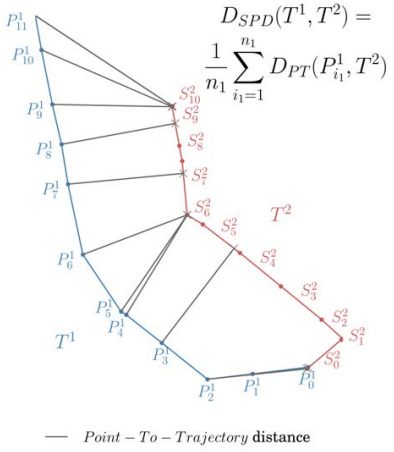


Figure 1.11 – *SPD* Distance from trajectory T^1 to trajectory T^2

Proposition 1. *If the points that compose the trajectory* T^1 *lie in the set of points* p_{pl}^2 *that compose the piece wise linear representation,* T_{pl}^2*, of trajectory* T^2*, then* $D_{SPD}(T^1, T^2) = 0$.

Proof. If the points that compose the trajectory T^1 lie in the set p^2_{pl} that compose the piece wise linear representation, T^2_{pl}, of trajectory T^2, all points of T^1 lie within one of the segments, that compose T^2_{pl}. By definition $D_{ps}(p^1_{i_1}, s^2_{i_2}) = 0 \ \forall p^1_{i_1} \in T^1, s^2_{i_1} \in T^2_{pl}$. It follows that $D_{pt}(p^1_{i_1}, T^2) = 0 \ \forall p^1_{i_1} \in T^1$ and finally $D_{SPD}(T^1, T^2) = 0$ □

This distance is not symmetric. If T^1 is a very small sub-trajectory of T^2, $D_{SPD}(T^1, T^2) = 0$, $D_{SPD}(T^2, T^1)$ can be very large. By taking the mean of these distances, the **"Symmetrized Segment-Path Distance"**, SSPD, is defined and is symmetric.

Definition 22. *Symmetrized Segment-Path Distance*

$$D_{SSPD}(T^1, T^2) = \frac{D_{SPD}(T^1, T^2) + D_{SPD}(T^2, T^1)}{2}.$$

In definitions 21 and 22, distances *SPD* and *SSPD* are computed by taking the mean of the *Point-to-Trajectory* distance and the *SPD* distance. If the maximum is used instead of the mean, one recovers the Hausdorff function between two trajectories. Computing only one distance between two locations makes it very sensitive to noise. Yet our method computes the mean of such quantities which makes it less so. For example, for the trajectories in Fig. 1.9, the *SSPD* distance between T^1 and T^2 is smaller than the distance between T^1 and T^3 or T^2 and T^3 ($D(T^1, T^2) = 0.58, D(T^1, T^3) = 1.5, D(T^2, T^3) = 2.03$).

Proposition 2. *SSDP is a symmetric.*

Proof. *SSDP* is a sum of Euclidean distances. By definition *SSDP* is greater or equal to 0. By definition 22, SSDP is symmetric. Finally theorem 1 states that, if $D_{SDP}(T^1, T^2) = 0$, T^1 is a sub trajectory of T^2. Therefore if $D_{SSDP}(T^1, T^2) = 0$, both $D_{SDP}(T^1, T^2) = 0$ and $D_{SDP}(T^1, T^2) = 0$, and $T^1 = T^2$. *SSDP* is then a *symmetric.* □

SSDP is quite similar to *OWD* but its definition resolves most of the problems of *OWD* regarding the desired properties defined in section 1.2.2.3

- The points coming from the interpolation of two observed locations of a trajectory are less trustworthy that the real observations. Hence, it is natural to give more weight to the observed points.
- *SSPD* distance does not require any additional parameters such as a threshold or a grid to be computed.
- Its computation cost is $O(n^2)$. It only depends on the number of locations.

1.2.5 Clustering

To evaluate these different distances, we will study different clustering methods obtained with the same algorithm but with distances computed using all previous distances. The different selected clustering methods and the quality of cluster criterion are examined in this section.

1.2.5.1 Methods

The choice of the clustering method is restricted by the characteristics of the trajectory object. Indeed, trajectories have different lengths which complicates an easy definition of a mean trajectory object. The *k-means* method cannot be used on our trajectory set, nor *spectral clustering* method. *k-medoid* can be used but an efficient algorithm, like *partitioning around medoids*, or *dbscan* method, require a valid *metrics*. Indeed, these algorithms are based on nearest neighbor and require the distance used to be known in order to satisfy the triangular inequality. Most of the studied distances, *SSPD*, *LCSS*, *DTW*, are not metrics. In this way, *dbscan* or *partitioning around medoids* algorithms will not be used. Moreover, *dbscan* depends on two extra parameters that are hard to estimate in this case.

To perform the clustering of the trajectories, we will focus on two methodologies : *hierarchical cluster analysis* (HCA) and *affinity propagation* (AP). As a matter of fact, *HCA* and *AP* can use distance/similarity which does not satisfy the triangle inequality. We point out that the choice of the clustering method is restricted to the trajectory object we deal with. Actually, trajectories have different lengths. *HCA* and *AP* are both methods which only require the distance/similarity matrix, and thus can cluster objects of different lengths. Both of these methods will be used to evaluate our distance.

1.2.5.2 Quality criterion of cluster result

A clustering algorithm aims to gather objects into homogeneous groups that are far one from another. Hence, the quality of a clustering is usually evaluated by looking at the between and within variance of the obtained clusters. On the one hand, the within variance shows the spread of elements belonging to the same groups. Because we want the elements of the same groups to be as similar as possible, we want the within variance to be as small as possible. On the other hand, we want objects that belong to different groups to be as far as possible from one another. Hence, the between variance, which shows the spread between clusters, should be as big as possible. The definition of within group variance and between group variance require the definition of a mean object. In our case, they cannot be computed here because of the impossibility of computing the mean of the trajectory object. Therefore, we approximate this mean by considering an exemplar, T^{ex}, of a set of a trajectory $\mathcal{T}$ of length $n^{\mathcal{T}}$, defined as:

$$T^{ex}_{\mathcal{T}} = \min_{\substack{T^i \\ i\in[0\dots n^{\mathcal{T}}]}} \Big\{ \sum_{\substack{j=1 \\ j\neq i}}^{n^{\mathcal{T}}} D(T^i, T^j) \Big\}.$$

By using the exemplar definition, we can define new criteria to approximate the between and within variance : the *Between-Like* and the *Within-Like* Criteria. Let $\mathcal{C}_1, \dots, \mathcal{C}_K$ be a set of clusters of $\mathcal{T}$, the *Between-Like* and the *Within-Like* criteria are defined as:

Definition 23. *Between-Like and Within-Like*

$$BC = \sum_{k=1}^{K} D(T_{\mathcal{T}}^{ex}, T_{\mathcal{C}_k}^{ex}),$$

$$WC = \sum_{k=1}^{K} \frac{1}{|\mathcal{C}_k|} \sum_{T^i \in C_k} D(T_{\mathcal{C}_k}^{ex}, T^i).$$

The Within-Like criterion shows the spread of elements belonging to the same cluster while the Between-Like criterion shows the spread between clusters. As for the variance, for a given number of clusters, we want the Within-Like criterion to be as small as possible, and the Between-Like criterion to be as big as possible.

These quality criteria can also be used to select the number of clusters. Indeed, the Within-Like criterion decreases with the number of clusters, while the Between-Like criterion increases with the number of clusters. Hence, we are looking for a trade off between these criteria and the number of clusters. We choose the number of clusters so that adding one more cluster does not decrease significantly the Within-Like criteria.

1.2.6 Experimental evaluation

In this section, we evaluate and compare 7 distances *LCSS*, *DTW*, *Hausdorff*, *Fréchet*, *Discrete Fréchet*, *OWD grid* and the *SSDP*. We present the python package *trajectory_distance* where the distances have been implemented. We compare their computational cost and the results of the application of different clustering techniques for each of them. We also use python for the implementation of the chosen clustering algorithms, the *sklearn* library for *affinity propagation* and *scipy* library for *hierarchical clustering analysis*. For the latter, *weighted*, *average*, *ward*, *complete* and *single* linkage criteria are compared.

1.2.6.1 A python package : trajectory_distance

All distances have been implemented in python and are available in the *trajectory_distance* package available on github from this url : **https://github.com/bguillouet/traj-dist**. Distances have also been implemented in Cython for 2-D trajectories. Cython is a language which enables the declaration of static variables and to use the C math library. All distances but *OWD grid* are based on Euclidean distance for point to point computation. To compute *OWD grid*, we implemented the grid algorithm defined in [Lin and Su, 2005]. The trajectories need to be mapped in a grid space. We use the Geohash system to accomplish this task. It is based on a hash function which subdivides the geographical space into a grid. Different precision parameters of this grid are available from 1 ($5009.4km \times 4992.6km$ cell) to 12 ($3.7cm \times 1.9cm$). The hash function enables quick mapping of latitude/longitude coordinates to the appropriate grid. Once the trajectory

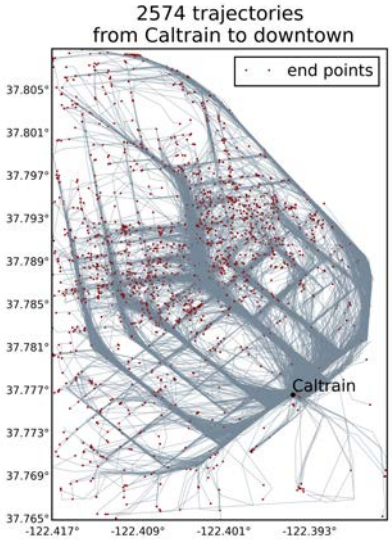


Figure 1.12 – Trajectories subset

locations have been mapped to a grid, we implemented a algorithm to find all grids they crossed between two locations and to obtain the grid representation of the trajectory as defined in Definition 20.

1.2.6.2 The data

The data we used are GPS data from 536 San-Francisco taxis over a 24-day period. These data are public and can be found on [Piorkowski et al., 2009]. We extracted a subset of this data as shown Fig. 1.12.

This subset is a blend of 2574 trajectories. All have the same pickup location, the Caltrain station, and all have a drop-off location in downtown San-Francisco.

1.2.6.3 Computation cost

In Table 1.7 we can observe the computation time needed to compute the matrix distance for a subset of 100 taxis trajectories of the studied subset described in section 1.2.6.2. Trajectories are composed of 3 to 39 locations, most having around 10.

Fréchet distance is the distance that requires the most computation time. It is the only method that runs in $O(n^2 log(n^2))$. *DTW*, *LCSS* and *Discrete Fréchet* distances are the fastest computed methods, all having the same order of computation time. This three methods require computing the Euclidean distance between each pair of points that compose the two trajectories. They only differ by their cost function. As explained in section 1.2.4, *Hausdorff* and *SSPD* distance also are computed in the same way. *Hausdorff* uses the maximum of the *Point-to-Trajectory* distance and *SSPD* the mean, which explains

Table 1.7 – Computation Time in seconds

Distance	Computation time
Fréchet	268.32
Discrete Fréchet	0.58
Hausdorff	2.47
DTW	0.66
LCSS	0.60
SSPD	2.46
OWD Grid 5	1.88
OWD Grid 6	7.44
OWD Grid 7	52.96

why they have almost the same computation time. If they both have the same complexity as that of *DTW*, *LCSS* and *Discrete Fréchet*, $log(n^2)$, they require more operations, which explains their relative slowness. The time computation of the *OWD grid* is strongly dependent on the precision we choose. Indeed, a precision of 5 resolves the studied space into a 3×3 grid space, while precisions of 6 and 7 resolve a grid space of 10×5 and 34×25 respectively. This implies that the number of cells required to represent the trajectory is very different from one precision to another. With precision 5, 1 to 3 cells are needed to represent the trajectory. With precision 7, the shortest trajectory is represented with only 1 cell but the longest needs 47 cells to fully describe it. Therefore, with precision 5, *OWD grid* is faster than *SSPD*, but with precision 6 and 7, *OWD grid* is 3 times and more than 20 times faster than SSPD distance. A good trade-off needed to be found to have a precision which enabled to adequately describe the trajectories in the new grid space, within a reasonable time computation.

1.2.6.4 Analysis of the clustering method

In Fig. 1.13 we observe the evolution of the Within- and the Between- like criteria described in section 1.2.5 for the distance *SSPD* and for the selected methods *AP* and *HCA*. Both the Between-Like and the Within-Like criteria are shown because the sum of these two criteria is not constant as opposed to the sum of the between and within variance.

The *HCA single* method gives much worse results than the other methods, regardless of the number of clusters. All other *HCA* methods display the same evolution of the studied criteria with respect to the cluster size. A plateau can be observed starting at cluster sizes between 10 and 20. Adding more clusters does not decrease significantly the Within-Like Criterion. We can see Figure 1.13-c, that the *HCA ward* give better results than the other *HCA* method for all the different number of clusters. The same conclusions can be made regardless of the distance used.

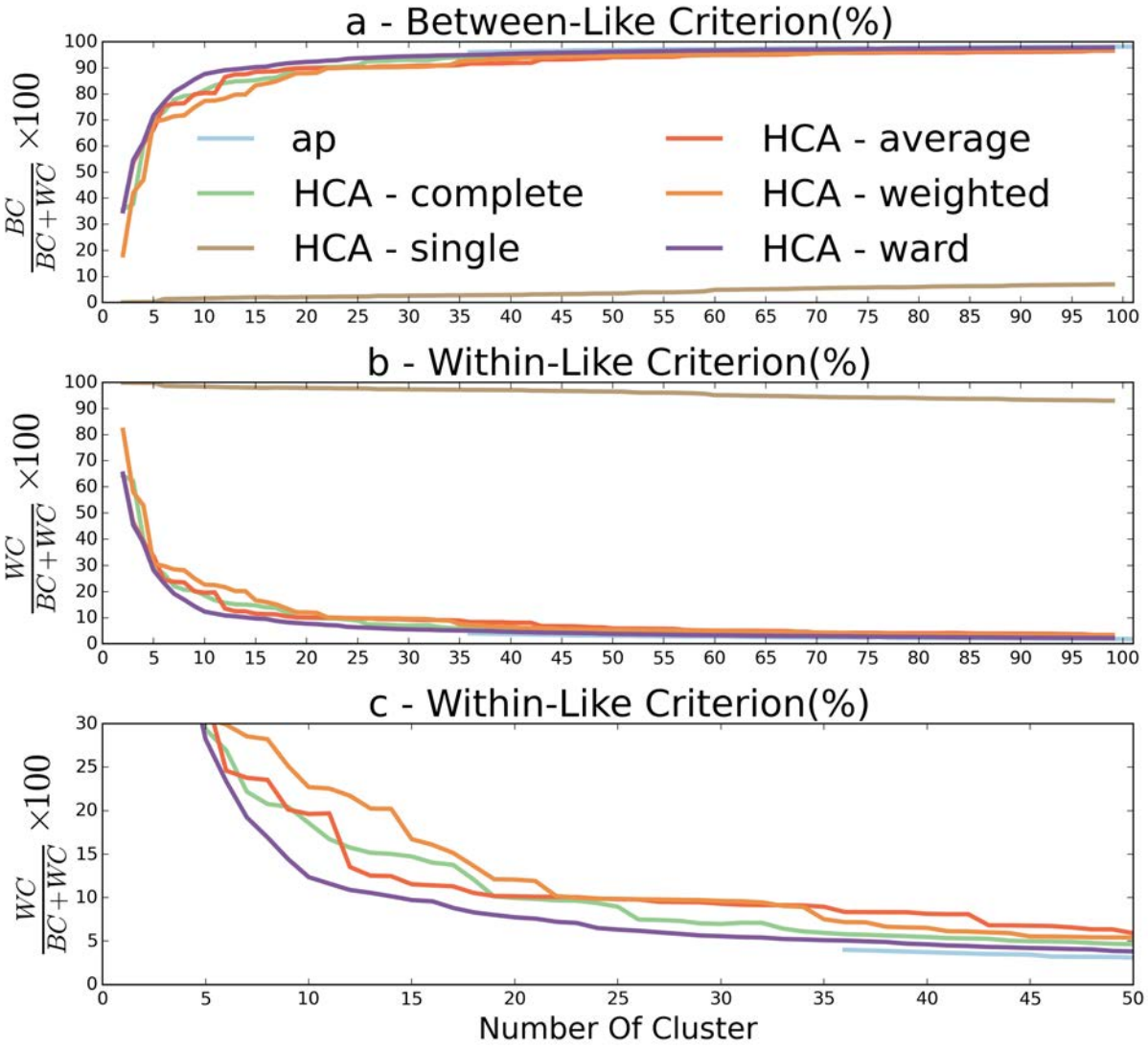


Figure 1.13 – Evolution of the Between-Like (a) and Within-Like (b) criteria depending on cluster size for clusterings obtained using *SSPD*. On (c), Within-Like criterion is displayed with a reduced HCA30.

AP give the best results. However the latter does not achieve clustering using any less than 36 clusters, a potentially large number, given that *HCA* ward achieves clustering around 15, and with good Within-Like criterion results. Moreover, the minimum cluster size found by the *AP* method differs significantly according to the distance used. No less than 21 clusters are found with the *DTW* distance and 54 with *Hausdorff*. This is the main inconvenient of this method.

The *HCA Ward* method and the *AP* method with the *preference* parameter fixed

to the minimum of the computed matrix distance will be used to compare the studied distances in more details.

1.2.6.5 Analysis of the distances

We can observe the evolution of the Within-Like and the Between-Like criteria for the two selected clustering methods as well as for all studied distances. The *HCA WARD* results are display in Fig. 1.14, and the *AP* results in Fig. 1.15.

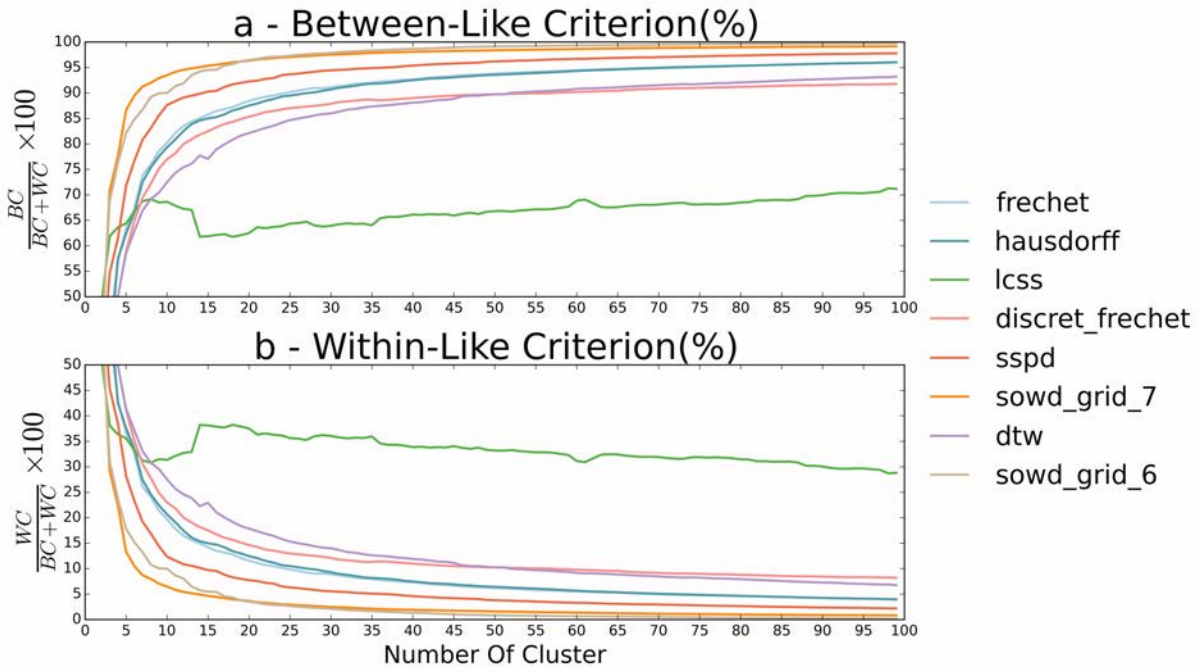


Figure 1.14 – Evolution of the Between-Like (a) and Within-Like (b) criteria depending on cluster size for all distances using the HCA-WARD method

In Figure 1.14, the evolutions of the two criteria when the number of cluster increase is similar for all the distances but *LCSS*. For instance, the curves which represent the Within-Like criterion decrease quickly when adding more cluster for a low number of cluster. Then they all reach a point when adding more clusters does not decrease significantly the Within-Like criterion. Theses point vary from one distance to another. It is reached around 10 clusters for the *OWD grid* and around 25 clusters for the *Discret Frechet* distance. The minimum cluster size found by the *AP* method differs significantly according to the distance used. No less than 21 clusters are found with the *DTW* distance, 36 with *SSPD* and 54 with *Hausdorff*. However, the same conclusion can be made in terms of Within-Like and Between-Like criteria regardless of the number of cluster.

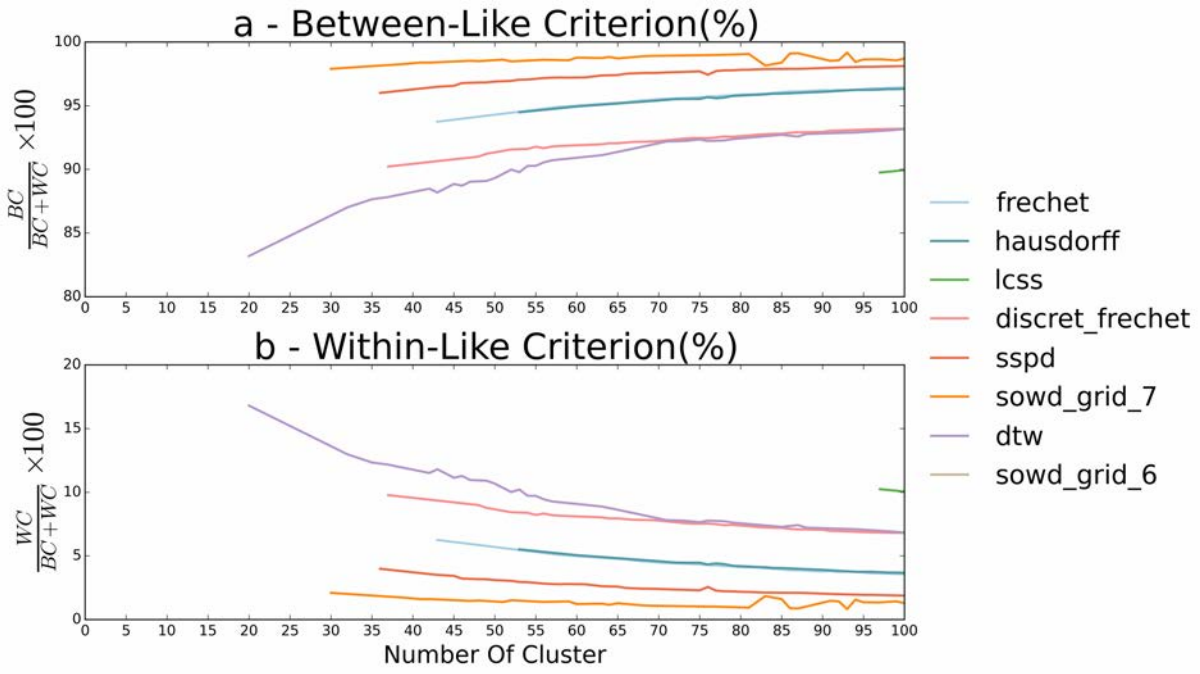


Figure 1.15 – Evolution of the Between-Like (a) and Within-Like (b) criteria depending on the cluster size for all distances using the AP method

The Warping-based distances, *LCSS* and *DTW*, give the poorest results with *LCSS* being significantly worse than *DTW*. The two shape-based distances *Frechet* and *Hausdorff* give better results. The evolution of their criteria is very similar to one another. The *Discrete Fréchet* distance is between these two types of distances. These results confirm that shape-based distances are better adapted than warping-based distances for our objectives.

OWD grid gives the best results. It has the lowest value of Within-Like Criterion for all cluster sizes using both *HCA WARD* and *AP* clustering methods. The results for precision 5 are not displayed here. The discretization of the space is too inaccurate and did not provide good clustering. Precision 7 gives slightly better results than precision 6. This shows that increasing the precision parameter yields better results. However, when there are more than 15 clusters found, the within and Between-Like criteria are almost the same. In section 1.2.6.3 we have seen that the computation time to compute the distance with precision 7 is seven times higher than the computation time with precision 6. Hence, we need to look for the optimal criteria to find a good trade off between good clustering results, and reasonable computational time. The choice of this criteria is a strong disadvantage of the *OWD method*, because it implies to look for the best precision parameter for each data set.

Finally, the new distance *SSDP* is the distance which best approaches the results found

with *OWD grid*, regardless of the number of cluster. But unlike with *OWD grid*, we do not need to look for the optimal precision parameter, in order to compute it, nor to map the trajectory to a new space. This enables our distance to be more easily adapted to different subsets of trajectories.

We observe the visual results for this distance and both *AP* and *HCA ward* clustering methods, in Fig. 1.16, and the isolated clusters, in Fig. 1.17. For the *HCA ward* method, we display the clustering result obtained with 15 clusters because we have seen Section 1.2.6.4 that a plateau can be observed on the evolution of the Within-Like criteria with respect to the number of cluster starting at cluster sizes between 10 and 20 for the *SSPD*. For the *AP*, clustering results with 36 clusters is displayed since no less cluster can be obtained with this method.

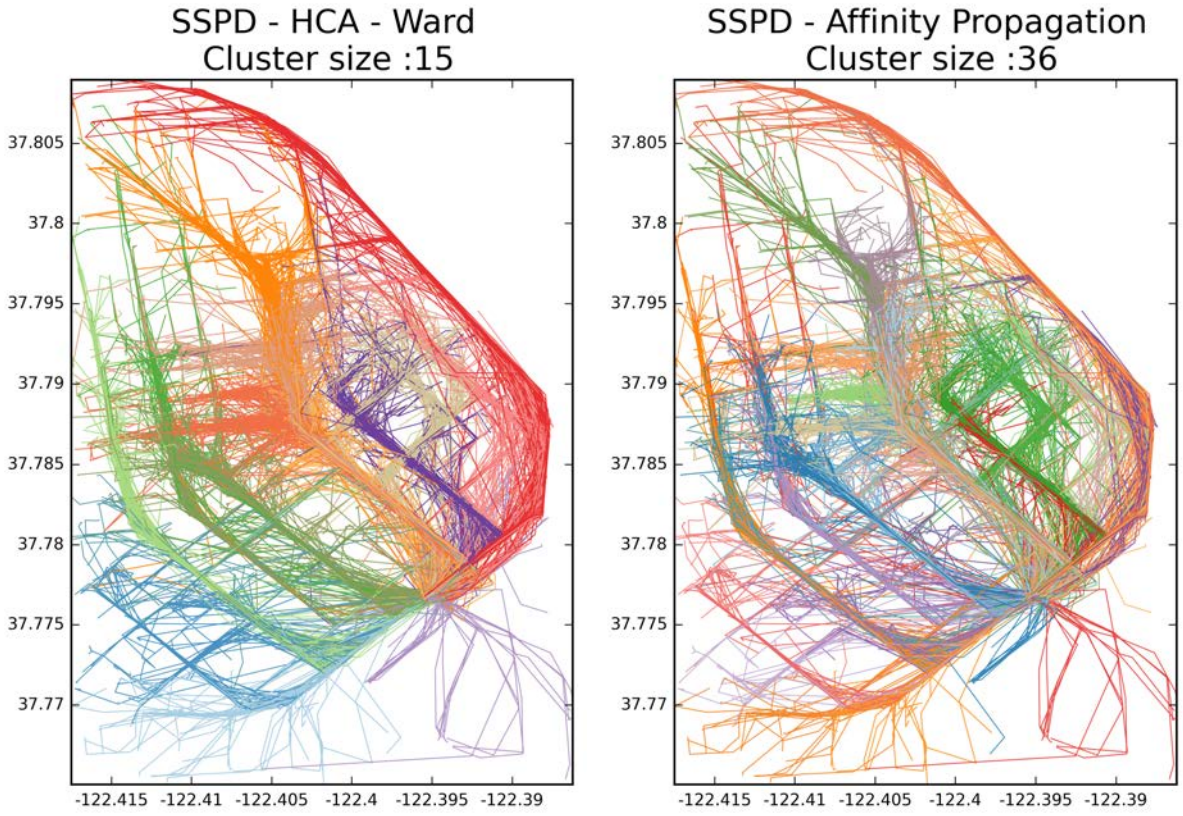


Figure 1.16 – Clustering results with *SSPD* distance

We observe that trajectories are well classified according to their path. In Fig. 1.17, clusters found using *HCA WARD* seem to be consistent. The cluster size with *AP* method is 36. This is a large number according to the Within-Like criterion computed with *HCA*. In fact, the Within-Like criterion does not decrease much between 15 and 36. However, we can see that the number of clusters found with *AP* are still consistent.

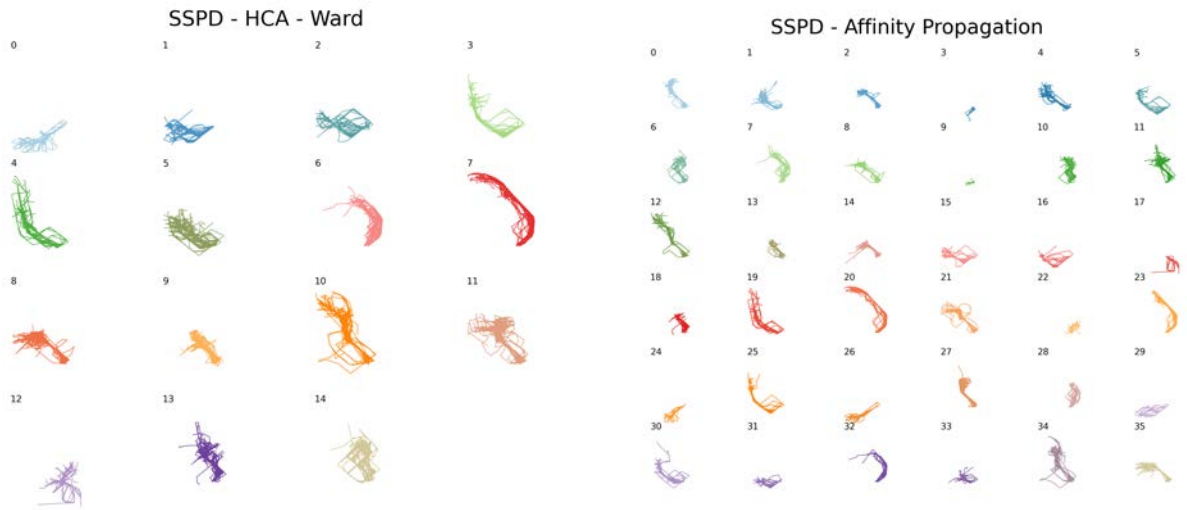


Figure 1.17 – The isolated clusters

A cluster computed with the *HCA WARD* method based on a matrix distance computed with *SSPD* gives the best result. The Between-Like and Within-Like criteria show that this method is best used to regroup cluster around exemplar. We obtain a partition of the trajectories subset, such as each cluster represents a path taken by the drivers. We obtain a partition of traffic based on the taxi drivers' behavior leaving the Caltrain station in San Francisco. The number of trajectories in each cluster gives us a representation of the importance of each network traffic stream.

1.3 Conclusion

Clustering of non Euclidean objects deeply relies on the choice of a proper distance. For trajectories analysis, we presented different distances focusing on different features of such objects. To cope with their different weaknesses we propose a new distance, the *Symmetrized Segment-Path Distance*. This distance is time insensitive, and compares the shape and the physical distance between two trajectory objects. It does not require any additional parameters nor mapping trajectories in a different space. Hence, It can be applied on any set of trajectories, regardless of the area they come from. It enables us to obtain good clustering using either *hierarchical clustering* and *affinity propagation* methods. In this way, the clusters obtained are homogeneous with regard to shape and seem to properly capture the behaviours of the drivers. We have thus obtained a partition of the network based on the drivers' usage that can still be interpreted as vehicle trajectories. This partition can be used to solved different problem. Many applications which recommend places to visit, or which target advertising based on our destination need to forecast the final destination of drivers or predict the travel time of driver trips. Cities

which wish to organize trip distribution of a city, also need to know the behaviours of the cars drivers. Some of these problems will be tackled in a following work, based on the partition obtained with our method to cluster trajectories.

1.A Appendix

1.A.1 More results

1.A.1.1 Analysis of the number of clusters selection

Figures 1.18 and 1.19 show the evolution of the Whithin-Like Criterion and Between-Like Criterion for the selected methods *AP* and *HCA* and for the distances *DTW*, *Frechet* and *OWD grid* with a precision parameter equal to 7. As stated in section 1.2.6.5, results for these distances are the same as that for the *SSPD* distance: the *HCA* method with *single* linkage criterion method gives the worst results, while the textitHCA method with *ward* and *AP* appear to give the best results. For all methods a plateau can be observed starting from a clusters size between 10 and 20? Adding more clusters does not decrease significantly the Within-Like Criterion. This confirms that twenty is a good cluster size. We can also see that with AP methods, we can not find less than 20 clusters with *DTW*, 43 with *Frechet* and 30 with *OWD grid*. This confirms the main drawback of the *AP* method and prevents a complete comparison of the performance of the studied distances using this clustering method.

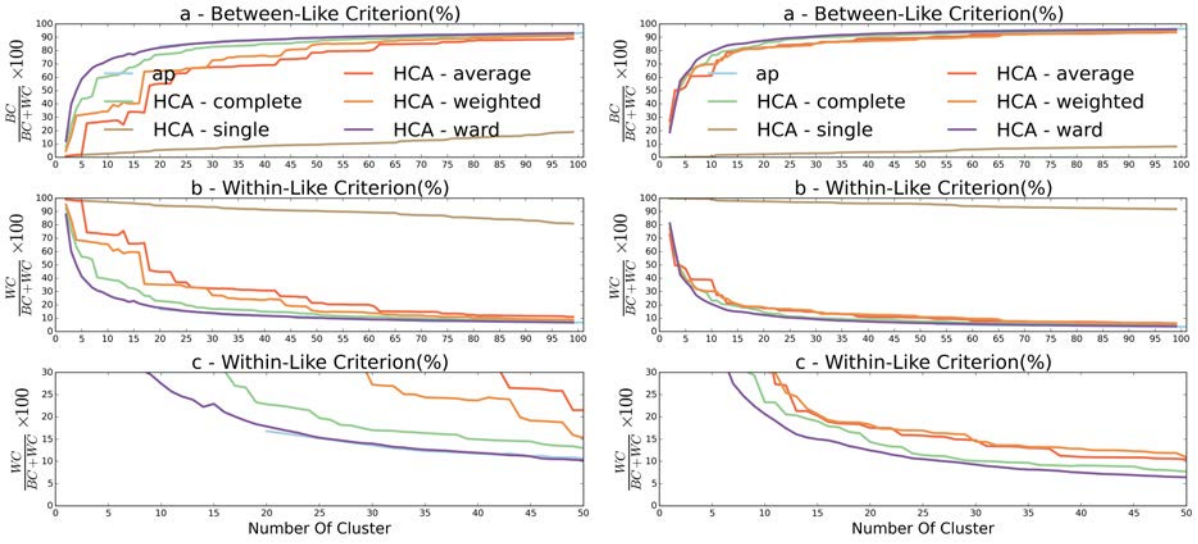


Figure 1.18 – Evolution of the Between-Like (a) and Within-Like (b) criteria depending on cluster size for clusterings obtained using DTW and Hausdorff. In (c), Within-Like criterion is displayed with a reduced scale of 0 to 30.

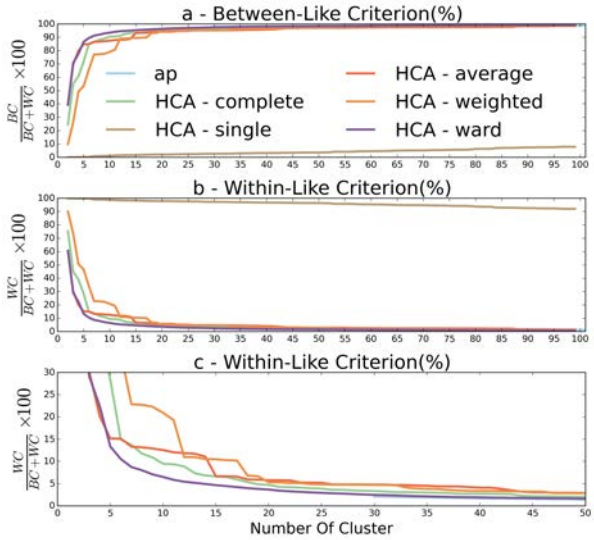


Figure 1.19 – Evolution of the Between-Like (a) and Within-Like (b) criteria depending on cluster size for clusterings obtained using *OWD grid* with precision equal to 7. In (c), Within-Like criterion is displayed with a reduced scale of 0 to 30.

1.A.1.2 Analysis of the distance

In figure 1.20 we can observe the evolution of the Within-Like and the Between-Like criteria for the two best clustering methods, the *HCA WARD* and the *AP* for the Caltrain city center dataset. Once again, the same conclusion can be made on this dataset compared the Caltrain downtown dataset, whose results are discussed on Section 1.2.6.5. The Warping-based distances, *LCSS* and *DTW*, give the poorest results with *LCSS* being significantly worse than *DTW*. The two shape-based distances *Frechet* and *Hausdorff* give better results. The evolution of their criteria is very similar to one another. The *Discrete Frechet* distance obtained is between these two types of distances. These results confirm that shape-based distances are better adapted than warping-based distances for our objectives. The *OWD grid* still gives the best results.

1.A.1.3 Visual results

On Figures 1.22 and 1.23 we can observe more visual results of the clustering of trajectories with CAH WARD clustering applied on a distance matrix computed with *SSPD* methods for both the Caltrain city center and Porto Sao Bento datasets. The number of clusters are 25 and 45 respectfully.

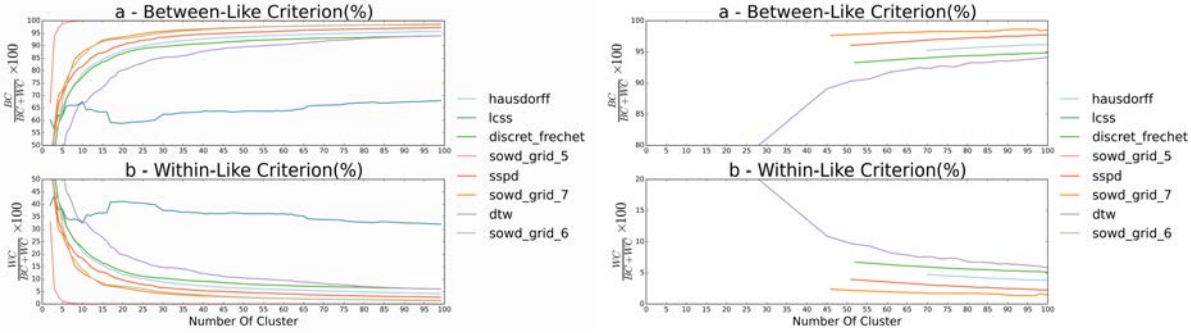


Figure 1.20 – Evolution of the Between-Like (a) and Within-Like (b) criteria depending on cluster size for all distances using the *HCA-WARD* and the *AP* method on the Caltrain city center dataset

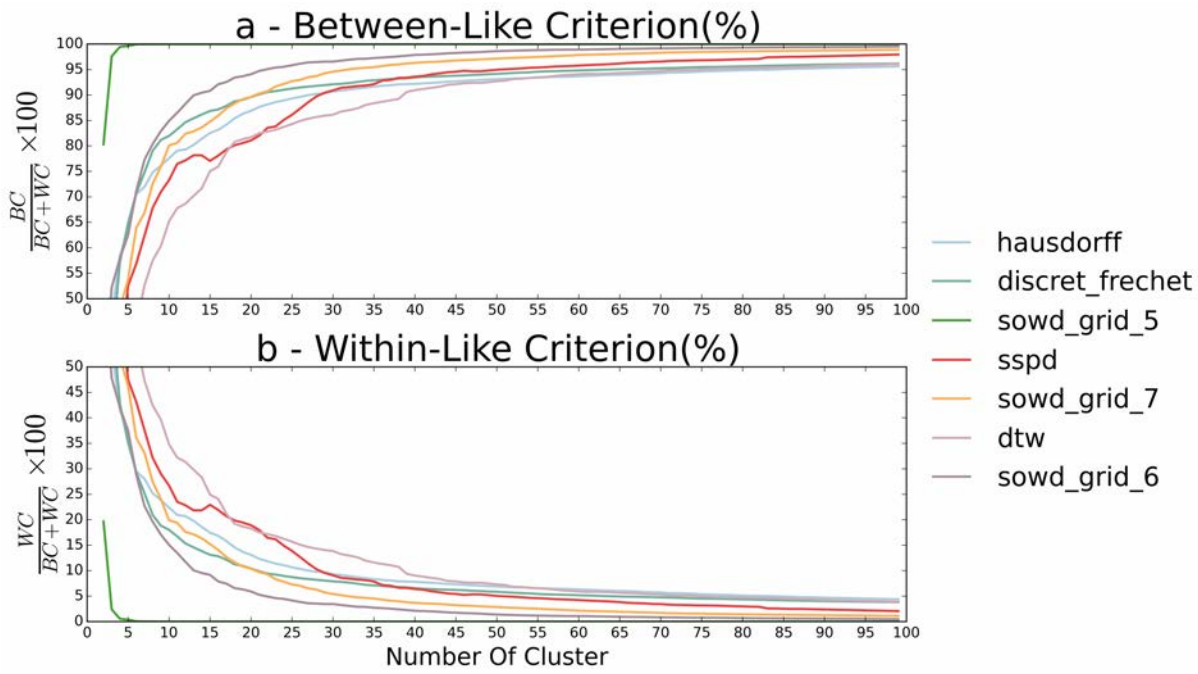


Figure 1.21 – Evolution of the Between-Like (a) and Within-Like (b) criteria depending on cluster size for all distances using the *HCA-WARD* and the *AP* method on the Sao Bento city center dataset

1.A.2 A python package : trajectory distance

Nine distances have been implemented in python and are available within a Python module :*trajectory_distance*. These distances have all been implemented in both Python and Cython. The Python implementation handles all trajectories, whatever their dimensions

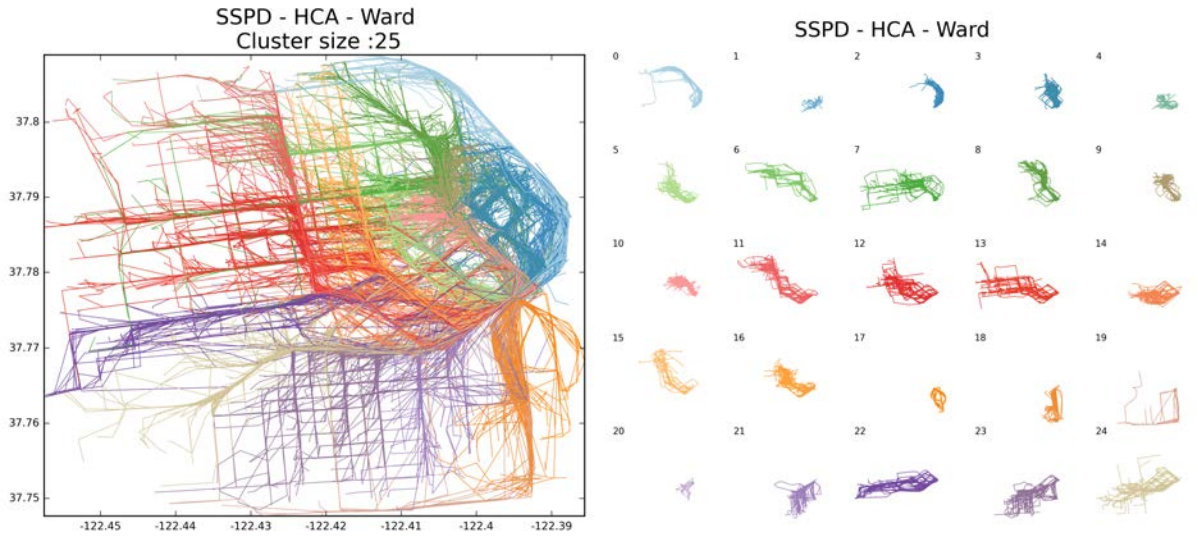


Figure 1.22 – Clustering results with SSPD distance and the isolated clusters for Caltrain city center dataset

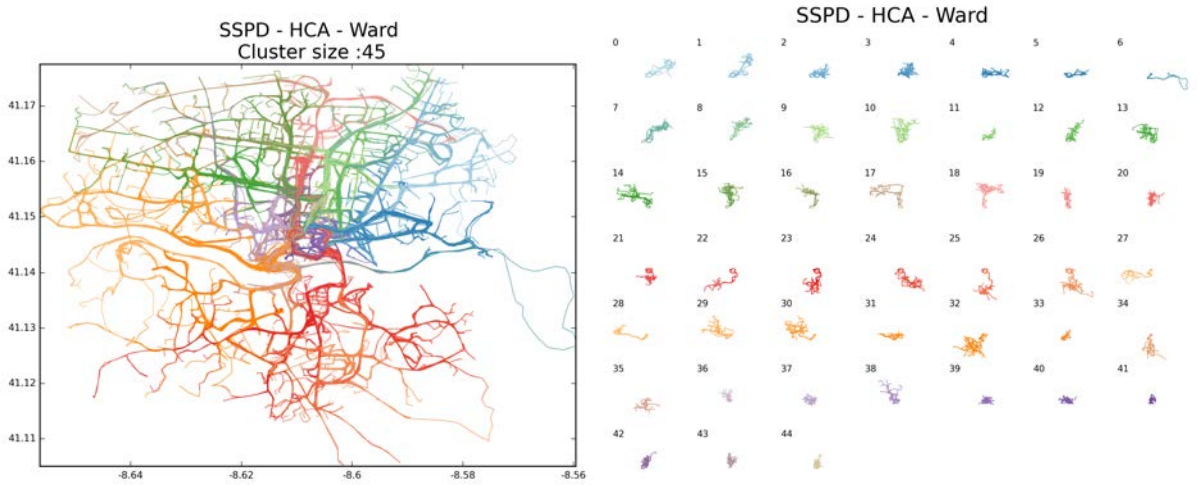


Figure 1.23 – Clustering results with SSPD distance and the isolated clusters for Caltrain city center dataset

are. Distances have also been implemented in Cython. Cython is a language which enables the declaration of static variables and the use of the C math library. Hence the implementation handles only 2-D trajectories. For 2-D trajectories the default implementation use is the *cython* one, unless the *implementation* parameter is set to *python*.

The implementations of the warping distances, *DTW*, *LCSS*, *EDR* and *ERP*, follow the same template. These distances are computed recursively, starting from the first points

Table 1.8 – Python Package

Distance	Euclidean	Haversine	Cell	Parameters
DTW	X	X		None
LCSS	X	X		ϵ
EDR	X	X		ϵ
ERP	X	X		g
Discret_Frechet	X			None
Frechet	X			None
Hausdorff	X	X		None
SOWD grid			X	*Converted*, *Precision*
SSPD	X	X		None

of the trajectories, and for all combinations of sub-trajectories. They are then computed according to their recurrence formula and the cost function detailed in Table 1.4. DTW does not require any parameters. *LCSS* and *EDR* require ϵ parameter, which is the distance threshold for which locations are considered equal or not, while for *ERP* the g is a zero vector $\in \mathbb{R}^n$, where n is the trajectory dimension. All these distances have been implemented using the *Euclidean Distance*. But they can be also be computed using the *Haversine distance* (see Definition 38) which compare distances in meters. When trajectories are built with longitude and latitude coordinates, it is easiest to select and ϵ values in meters.

The *Discret_Frechet* distance and the *Frechet* distances have been implemented according to the algorithms described in [Alt and Godau, 1995] and [Eiter and Mannila, 1994] respectively. These algorithms are defined on an Euclidean space. Hence, their implementation is not available with the *Haversine Distance*

The *SSPD* distances implementation follows the definition presented in section 1.2.4. Due to their similarity, the *Hausdorff* distance follows almost the same implementation. They are both implemented using *Euclidean* and *Haversine* distances.

Finally, the *OWD grid* is the only distance not implemented using the neither with *Euclidean* nor *Haversine* distances. The trajectories need to be mapped in a grid space, then a distance between cell is computed knowing that the distance between two adjacent cells is equal to 1. The trajectories then need to be mapped in a grid space. We use the Geohash system to accomplish this task. It is based on a hash function which subdivides the geographical space into a grid. Different precision parameters of this grid are available from 1 (5009.4km x 4992.6km cell) to 12 (3.7cm x 1.9cm cell). The hash function enables quick mapping of latitude/longitude coordinates to the appropriate grid. This implies that the *OWD grid* implementation is limited to longitude/latitude coordinates so far, except if the trajectory coordinates are already mapped to a grid space. Once the trajectory locations have been mapped to a grid, we implemented a algorithm to find all grids they

crossed between two locations in order to obtain the grid representation of the trajectory as defined in Definition 20. Once the trajectory has been mapped to a grid, we compute the *OWD grid*, with the grid algorithm defined in [Lin and Su, 2005].

The based distances and the parameters required for all the distances are detailed in Table 1.8.

1.A.3 Newtork constrained clustering

Vehicles on roads, cannot move freely. Their trajectories are constrained by the underlying spatial network. The main advantages of this type of method are obvious. If we can match the GPS data of the trajectory to the road network, we can improve its precision. Also, It can be possible to retrieve the exact path between two locations, taking into account the available roads. However, matching GPS data to a road network requires a considerable amount of work. We did not consider these distances in our review, because it implies knowing the road network of each studied area. Hence, it prevents a quick application from one dataset to another. However, methods based on the underlying road network also have been widely studied, and we present here some of these methods.

Instead of defining trajectories on Euclidean spaces as a set of successive locations, trajectories are defined on graph $G(V, E)$ and can be defined as a set of successive vertexes or edges:

Definition 24 (Vertex Trajectory)**.** T_v*:* $((v_1, t_{v_1}), \ldots, (v_{n_T}, t_{v_{n_v T}}))$ *, where* $v_k \in V$*, and* n_v *is the length of the vertex trajectory.*

Definition 25 (Edge Trajectory)**.** T_e*:* $((e_1, t_{e_1}), \ldots, (e_{n_T}, t_{e_{n_e}}))$ *, where* $e_k \in E$*, and* n_e *is the length of the edge trajectory.*

New distances can then be established based on graph theory.

1.A.3.1 NETSCAN (Han *et al.*, [Kharrat et al., 2008]) 2008

Han *et al.* proposed the NETSCAN algorithm based on the density based algorithms DBSCAN. It is a two-step clustering algorithm, which takes advantage of the road network to estimate the object density. The first step in NETSCAN is to find the network paths that are the densest in terms of moving objects transiting on them, using a transition matrix relative to the road network. This step requires the definition of the same parameters used in DBSCAN: density threshold MinPts and similarity threshold ϵ_{seg}. In the next step, NETSCAN groups the trajectories according to their similarity to each dense path generated in the first step. The similarity measure used in this step compares two trajectories where one is the reference. This measure reflects the resemblance to an object and it is not symmetric. The similarity is computed as the ratio between the common

length among trajectories and the length of the reference trajectory

$$Sim\_traj = \frac{Lenghth(commonpart)}{Lengt(reftraj)}$$

For each dense path, hNETSCAN computes the similarity for each trajectory. If the similarity is above a user defined threshold value, ϵ_{traj}, then the trajectory is kept in the cluster.

1.A.3.2 Network distance measure (Tiakas et al.,2009[Tiakas et al., 2009])

Tiakas et al. established a *Network distance measure* between two *vertex trajectories* T_v^i and T_v^j as the sum of a given cost d between vertexes that compose them.

Definition 26 (Network Distance Measure)**.**

$$D_{net}(T_v^i, T_v^j) = \frac{1}{n_v m} \sum_{k=1}^{v_v m} d(v_k^i, v_k^j).$$

The cost distance d between vertexes could be defined as $d(v_k, v_l) = \frac{c(v_k,v_l)+c(v_k,v_l)}{2D_G}$ or $d(v_k, v_l) = \frac{\|v_k,v_l\|_2}{2D_G}$ where $c(v_k, v_l)$ is the shortest path distance from source node v_k to a destination node v_l and D_G is the maximum value of the shortest path distance over all pair of vertexes (*i.e.* $D_G = \max\{c(v_k, v_l)\}, \forall v_k, v_l \in V(G)$).

On the other hand, time information is independently computed. A distance with respect to time, D_{time} is given by incorporating time information.

$$D_{time}(T^i, T^j) = \frac{1}{n_v m - 1} \sum_{k=1}^{n_v m-1} \frac{|(t^i_{v_k+1}k + 1 - t^i_{v_k}k) - (t^j_{v_k+1}k + 1 - t^j_{v_k}k)|}{\max\{(t^i_{v_k+1}k + 1 - t^i_{v_k}k), (t^j_{v_k+1}k + 1 - t^j_{v_k}k)\}}. \quad (1.4)$$

Finally, the total distance can be expressed as follows:

$$D_{total}(T^i, T^j) = W_{net} \cdot D_{net}(T^i, T^j) + W_{time} \cdot D_{time}(T^i, T^j), \quad (1.5)$$

where D_{total} is a true distance.

This distance assumes that the compared trajectories have the same length n_{v_T}. If the trajectories do not have the same lengths, the sub-trajectory of the longest trajectory which matches the most the shortest one is used. But the trajectories are never compare as a whole. It makes this distance use full for query processing, but not for global comparison. Moreover no information is provided regarding the choice of W_{net} and W_{time}.

1.A.3.3 NNCluster [Roh and Hwang, 2010] 2010

Roh et al. adapt the Hausdorff distance to network-constrained category.

Definition 27. *Road segment distance The road segment between e_k and e_l is defined as follows:*

$$D_{RoadSeg}(v_i, v_j) = max \left\{ \begin{array}{l} \min\{d_{sp}(v_{k,s}, v_{l,s}), d_{sp}(v_{k,s}, v_{l,e})\} \\ \min\{d_{sp}(v_{k,e}, v_{l,s}), d_{sp}(v_{k,s}, v_{l,e})\} \\ \min\{d_{sp}(v_{l,s}, v_{k,s}), d_{sp}(v_{l,s}, v_{k,e})\} \\ \min\{d_{sp}(v_{l,e}, v_{k,s}), d_{sp}(v_{l,s}, v_{k,e})\} \end{array} \right\}$$

Definition 28. *Trajectory distance The Trajectory distance between T^i and T^j is defined as follows:*

$$D_{RoadTraj}(T^i, T^j) = max \left\{ \begin{array}{l} \max_{e_k^i \in T^i} \min_{e_l^j \in T^j} \{D_{roadSeg}(e_k^i, e_l^j)\} \\ \max_{e_l^j \in T^j} \min_{e_k^i \in T^i} \{D_{roadSeg}(e_l^j, e_k^i)\} \end{array} \right\}$$

This distance inherits the advantage of the *SSPD* distance listed in section 1.2.4. Moreover it respects the real topology of the trajectory and is a *metric*. The *NNCluster* is based on this given distance. The idea of the *NNCluster* is to cluster trajectories together if they have a common nearest neighbour. Clusters are first initialized based on nearest neighbour. Then each cluster is extended based on a distance between clusters which is the *trajectory distance* between exemplars of each cluster.

1.A.3.4 NEAT (Han et al., [Han et al., 2012]) 2012

The NEAT algorithm is a density-Based algorithm with respect to the network. The trajectory Tf^j is defined as a sequence of t-segments, tf_i^j, which are consecutive points of the same trajectory linked to the same edge.The Neat Algorithm is a three step algorithm:

- Base cluster: T-fragments from all trajectories are grouped into base clusters, S, according to their edge identifier. $S_e = \{tf_i^j |\ tf_i^j \in \mathcal{T},\ tfi.edge = e\}$.
- Flow cluster formation: Starting from the base Cluster S_e with the highest density do build a flow cluster. Base clusters are then iteratively merged to the flow cluster according to three parameters which are the flow factor q, the density factor k and the speed limit factor v which are merged into a unique criterion called the *merging selectivity* $SF_j = w_q.q + w_k.k + w_v.v$. The base cluster having the highest merging selectivity value is merged with the flow cluster. The iterations stop when the last base cluster merged to the flow cluster has no neighbor.
- Phase 3 - Flow cluster refinement: Cluster Flow cluster with DBSCAN according to an adapted Hausdorff distance on trajectory mapped on Graph.

Given two flow clusters F_i and F_j with endpoints $\{v_{i1}, v_{i2}\}$ and $\{v_{j1}, v_{j2}\}$ respectively, the Hausorff distance on a graph is defined as:

$$D_{HausGraph}(F_i, F_j) = max_{a\in\{v_{i1},v_{i2}\}} min_{b\in\{v_{j1},v_{j2}\}}(D_{sp}(a, b))$$

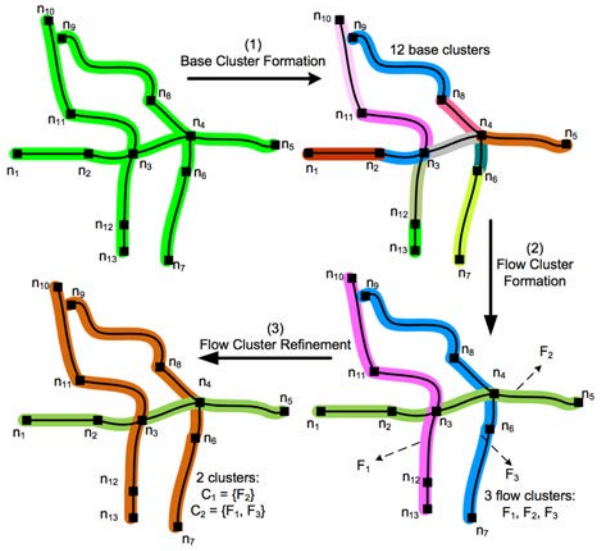


Figure 1.24 – The NEAT algorithm

According to the value of the weight, w_q, w_k, w_v we choose, different clustering can be found using the NEAT algorithm. Increasing w_k results in clusters where traffic is highly concentrated while increasing w_v will produce flow clusters that describe the routes where objects can travel the fastest. This method is effective in finding roads with high density. But it clusters t-segments in one and only one cluster which is really restrictive if we want to cluster all trajectories in an area.

1.A.3.5 Bag-of-Segments similarity (Rossi and El Mahrsi, [El Mahrsi and Rossi, 2012])Rossi 2012

Rossi and El Mahrsi compare trajectories on a segment-basis. They do not take into account the order of the segments composing a trajectory. Rossi and El Mahrsi [El Mahrsi and Rossi, 2012] consider trajectories as a set of edges. They assign first a weight to each segment (edges) e for every trajectory T^i. This weight, $\omega_{e,T}$ is assigned in a TF-IDF (*Term Frequency - Inverse Document Frequency*) way..

$$\omega_{e,T} = \frac{length(e)}{\sum_{e' \in T^i} length(e')} \cdot \frac{|\mathcal{T}'|}{|\{T' : e \in T'\}|}, \tag{1.6}$$

where $length(e)$ is the spatial length of the segment e, $|\mathcal{T}|$ is the cardinal of the data set $\mathcal{T}$ and $|\{T' : e \in T'\}|$ the cardinal of the subset of trajectories containing the segment e.

First Term of the product defines the importance of segment e in the trajectory t as the ratio of the length of e over the total length of the trajectory. The second term is the inverse of the importance of the segment over the all set of trajectory. The less the segment is present over all trajectory, the more significant he is for the trajectory T. This weight is inspired by the TF-IDF weight, widely used in text mining.

Then the cosine similarity measure as follows:

$$similarity(T^i, T^j) = \frac{\sum_{e \in E} \omega_{e,T^i} \cdot \omega_{e,T^j}}{\sqrt{\sum_{e \in E} \omega_{e,T^i}^2} \cdot \sqrt{\sum_{e \in E} \omega_{e,T^j}^2}}. \tag{1.7}$$

1.A.3.6 Conclusion

Due to the precision of the road network, numerous methods can be established regarding of the final objective. Instead of using nodes of the graph, Hwang et al. (2005[Hwang et al., 2005a]) only use the point of interest (POI) of the graph (important intersections of roads or places), while Han et al. (2012[Han et al., 2012]) cluster trajectories based on traffic flows. All these methods are efficient, precisely because they are based on a constrained network. But they assume that GPS data and their situation are perfectly mapped on the road network, which is not always an easy thing, and strongly dependent on the precision of the GPS device. Tiakas et al., (2009[Tiakas et al., 2009]) and Hwang et al.(2005,[Hwang et al., 2005a]) even assume that each nodes where the trajectories go through are known. Often this information is not sure. When the time interval between two GPS locations is significant, several path on the graph are possible to link the locations. The path can go trough different nodes. Moreover, the road network can be really dense, and the GPS data is not accurate enough to place them on the graph.

Chapter 2

A new model for road traffic modelling and predictive application

In the previous Chapter, we presented a clustering method producing clusters of trajectory which represent the main pattern of the behaviours of the drivers. The partition obtained contains then information about how the drivers used the network than can be used in order to solved different problems. However Two questions remains in order to be able to solve these problems:

- The clustering of trajectory regroups trajectories according to their spatial properties. Hence the partition obtained describes the main paths used by the drivers, but does not describe how the driver are moving within each of these paths. However, the way the drivers are moving along a path can be really different from one another. It is necessary to be able to compare them according to the way they are moving, but also according to more local features, such as their transit time at a given point, their arrival location, *etc...* Hence, the first question is: How to model the space in order to catch the movement of the drivers within each of the clusters of trajectory ?

- Assuming the first question is solved, and we are able to describe the way vehicles are moving within each cluster and to compare them. To take advantage of this modelling for a trajectory, it implies that this trajectory belong to one of the cluster. The trajectories that belong to the dataset used for the clustering are assigned to their cluster. But a new trajectory, for which prediction method are needed, is not linked to any of theses clusters. Hence, the second question is: How to link a new trajectory, which is not part of the dataset used for the clustering, to the cluster he belongs the most likely ?

We present in this chapter, a method enabling to solved both of these problems. We model for that each of the clusters of trajectories with a *Gaussian Mixture Model*.

This probabilistic model provides two significant advantages. First it divides the space into states. The trajectories can therefore be interpreted as succession of state enabling to extract features and quick comparison between them. Secondly, the space is then modeled by several Gaussian distributions, it enables then quick evaluation of the probability for a location to belong to one of these Gaussian distributions. Hence this will enable us to both assign a new trajectory to the cluster it belong the most likely and therefore, develop and apply prediction methods. The Section 2.1, is a paper, submitted in *IEEE Transactions on Intelligent Transportation Systems* magazine, in which we describe in detail how we establish the model. We also describe in this paper how we assign new trajectories to cluster of trajectory. Finally we develop a method on this model enables to predict the final destination of a vehicle trips. Experimental results are provided for both these methods. In Section 2.2 two other methods are develop, based on the model, in order to predict the arrival time of a vehicle trip, and its next location during the accomplishment of this trip. We then prove in this Section that the way we model the space enables to easily define new methods to compare trajectories to each other and then application to solved different traffic problems.

2.1 Destination prediction by trajectory distribution based model

2.1.1 Introduction

Monitoring and predicting road traffic is of great importance for traffic managers. With the increase of mobile sensors, such as GPS devices and smartphones, much information is at hand to understand urban traffic. In the last few years, a large amount of research has been conducted in order to use this data to model and analyze road traffic conditions. The aim of this paper is to tackle the issue of predicting the destination of vehicles given a prefix of their trajectory. It is useful to predict the final destination for several reasons. Some applications focus on recommending sightseeing places or targeting advertising based on a destination. These applications and advertising appear on our smartphone. They are not necessarily aware of the final destination of the user if he uses a cab or if he uses a know route previously taken and does not have indicated his final destination on his personal device. The prediction of the final destination is essential for these applications. Some research are also ongoing to automatically set destination in navigation systems based on our daily ride habits. Finally for cabs, predicting the final destination of a taxi has become an essential task. Cab companies, recently adopt electronic dispatch systems instead of VHF-radio systems. In most cases, taxi drivers operating with an electronic dispatch system do not indicate the final destination of their current ride. Hence, it is extremely difficult for dispatchers to know which taxi to contact. in order to improve the efficiency of electronic taxi dispatching systems it is important to be able to predict

the final destination of a taxi while it is in service. Particularly during periods of high demand, there is often a taxi whose current ride will end near or exactly at a requested pick up location from a new rider. If a dispatcher knows approximately where their taxi drivers will be ending their current rides, they would be able to best identify which taxi to assign to each pickup request. This problem has been the subject of a Kaggle challenge entitled "ECML/PKDD 15: Taxi Trajectory Prediction (I)"[kaggle, 2015].

The observations are time-stamped locations that correspond to the different positions of vehicles moving within a city monitored at different observation times. When dealing with a dataset composed of trajectories, the difficulty lies in the fact that the data convey both spatial information (locations of the vehicles on the map of the city) and temporal information (for each vehicle, the locations are indexed by time, which creates a sequence of locations that compose a full trajectory). Hence the data have a spatio-temporal structure that must be taken into account in order to model their evolution while the trajectories of the destination points to be predicted are unknown. Vehicle trajectories are also constrained to a road network which makes their time progression very irregular. Locations of vehicles can be seen as two-dimensional data in $\mathbb{R}^2$, that have to be compared to one another, taking into account characteristics of the trajectories they belong to, such as origin and destination.

In this paper, we propose a method that relies on a distribution based model for the trajectories. We first focus on the temporal structure of the data and gather the locations into clusters of points that belong to similar trajectories. For this, we rely on a distance that takes into account the geometric properties of trajectories which was developed in a previous work [Besse et al., 2016]. Then, we model the observations within each obtained cluster by the realisation of a random variable that must be estimated with a distribution on $\mathbb{R}^2$. This estimation step is achieved by considering a mixture of 2-dimensional Gaussian distributions fitted to the data by a maximum likelihood procedure. Thus, each cluster of trajectories corresponds to a parametric distribution model obtained by a mixture of Gaussian distributions. Using this procedure, we obtain a distribution model for points based on the assumption that they belong to a cluster of trajectories. Forecasting the destination of vehicles is a two step procedure :

- we first attribute the observed path of these vehicles to a cluster of trajectories and
- then extract from the trajectories within the cluster a feature that stands for the final destination point.

Hence using the learning set, we obtain a density classification method based on preliminary trajectories clustering. In addition to the forecast properties of this model that will be analysed on the taxis data set, this methodology provides a probabilistic model for spatio-temporal analysis of vehicle flows. It enables the extraction of distribution mobility patterns in an vehicle transportation system in order to understand urban mobility and flow.

The paper is organised into the following sections. Section 2.1.2 is devoted to the presentation of the data and the related work. In Section 2.1.3 we present how we obtained trajectories clustering by using a proper distance that takes into account the spatial properties of vehicle trajectories. Section 2.1.4 describes the issue of clustering points into clusters of trajectories with a mixture of Gaussian distributions and how to use these models to predict the final destination of taxi trip. Finally, in Section 2.1.5, we present the experimental results and the performance of our models.

2.1.2 Presentation of the forecast problem and related work

Consider vehicles' location data that consist of locations $p(t) \in \mathbb{R}^2$ observed at several observation times t that may differ for each vehicle's path. As an example of such data, through the paper we will test our procedure using two different datasets. The first contains over 11 million taxi-GPS samples of approximately 500 taxis collected over 30 days in the San Francisco, United-States[Piorkowski et al., 2009]. The second contains more than 83 millions taxis-GPS data points describing July 2013 to June 2014 for all 442 taxis in circulation in Porto, Portugal. This dataset has been provided for a Kaggle Competition [kaggle, 2015]. This dataset is also composed of metadata associated to the taxi trips, such that client, taxi stand or taxi identification but the dataset in San Francisco only got the taxi identification. Hence, we deliberately do not use these attributes because we want our method to be easily adapted from one dataset to another. We only use locations in $\mathbb{R}^2$ and the associated timestamp.

Our goal is to be able to determine the final destination point by observing the beginning of a trajectory. This issue is common to various area and data, animal migrations [Gaffney and Smyth, 1999], VideoFrames applied on Robotic [Kruse et al., 1997], Human [Vasquez and Fraichard, 2004] Vehicle on crossroads [Hu et al., 2006], or GPS data to study behaviour in Urban Commercial Complexes [Kim et al., 2011]. But vehicle trajectories are very different objects, they are constrained to a road network and have very irregular time progressions. The study of destination prediction requires comparing the information of previous trajectories with the current location of trajectories in order to identify the destination. Many authors have already discussed this issue. In the winning solution of the Kaggle-ECML/PKDD discovery challenge on taxi destination prediction, De Brebrisson *et al.* [de Brébisson et al., 2015] used a multi-layer perceptrons neural network on features vector composed of coordinates of beginnings of trajectories, and diverse context information, such as the departure time, the driver id and client information. Their training set has been built to match the trajectories in the test set's competition. If this solution can easily be adapted to other datasets, it implies a new training for some tested trajectories using more location information. Moreover, neural network scores are hard to interpret and can not be used to better understand the characteristics of the dataset. Krumm *et al.* [Krumm and Horvitz, 2006] and Ziebart *et al.* [Ziebart et al., 2008] also used external information, in addition to historical trajectories, such as travel time,

trajectory length, accident reports, road condition, and driving habits. They incorporate this information into Bayesian inference to compute the probabilities of predicted destinations. Parteson GRAPH *et al.* [Patterson et al., 2003] also used a Bayesian method to predict destination but for specific individuals based on their historical transport modes. The main idea of these studies is to use the external information to enhance the quality of the prediction. It then becomes dependent on the presence of this information and is inapplicable without them.

Monreale *et al* [Monreale et al., 2009] built a decision tree, named T-pattern Tree based on extracted movement patterns and predicted the next location of a new trajectory finding the best matching path in the tree. Tiesyte and Jensen [Tiesyte and Jensen, 2008] proposed a nearest-neighbour trajectory method that utilised distance measures to identify the historical trajectory most similar to the current partial trajectory.

Finally, most of the work dealing with destination prediction issue uses probabilistic methods based on the location to identify the most probable location after creating the probability model. Among them, the Markov model has been widely applied in predicting destinations. Ashbrook *et al.* [Ashbrook and Starner, 2003] find potential destinations by clustering GPS data, then predicting destinations from these candidates based on Markov models trained to find the next most likely destination based on those recently visited. Gambs *et al.* [Gambs et al., 2012] determined the destination by using the mobility Markov chains from the sequence of the POI (Point of Interest) to create the model. Simmon *et al.* [Simmons et al., 2006] also built the probabilistic model through observation of the drivers' habits. All these studies build prediction based on habits of one or a group of specific individuals based on their historical trips. But they require knowing the identity of the driver. Xue *et al.* [Xue et al., 2013] proposed a method which decomposes all available patterns into subtrajectories of neighboring locations. The subtrajectories are assembled into synthesized trajectories. Then, they build the Markov model, which quantifies the correlation between adjacent locations. The main drawback of both bayesian inference and the Markov model is in establishing how well they discretize the space. Either they use the true road network, [Patterson et al., 2003, Ashbrook and Starner, 2003, Simmons et al., 2006], which requires significant amount of extra work to map the GPS data to the graph network, or they use a grid of square cells [Krumm and Horvitz, 2006, Ziebart et al., 2008, Xue et al., 2013] which is a rough representation of the space and produces results dependent on the choice of the discretization of the grid.

Choi and Hebert [Choi and Hebert, 2006] present a Markov model based on segments of the trajectories, where latent segments are obtained by clustering segments of past trajectories. New trajectories are then modelled as a concatenation of segments, which are assumed as noisy realisations of the latent segment. One of the major drawbacks of this method is that it is used for short term prediction (at most 10 seconds ahead). Wiest *el al.* [Wiest et al., 2012] proposed a probabilistic trajectory prediction based on two types of mixture models. They also predict the vehicles trajectories only several seconds into

the future.

2.1.3 Model to Cluster Trajectories

In this section we describe how we cluster trajectories. We first recall the definition of trajectory used in this paper.

Definition 29. *A trajectory T^i is defined as*
$T^i : [(p^i_1, t^i_1), \ldots, (p^i_{n^i}, t^i_{n^i})]$,
where $p^i_k \in \mathbb{R}^2$, is composed of the longitude and the latitude of the observation, $t_k \in \mathbb{R}$ is the time at which the observation has been made $\forall\ k \in [1, \ldots, n^i]$, and where $n^i \in \mathbb{N}$ is the length of the trajectory T^i.

To cope with the sampling issues of trajectories, we first complete, when required, the locations between p^i_j (at time t^i_j) and p^i_{j+1} (at time t^i_{j+1}) by the piece wise linear representation between each successive location p^i_j and p^i_{j+1} resulting in a line segment s^i_j between these two points. This new representation is called the *piece wise linear* trajectory. In this representation, no assumption is made about time indexing of segment s^i_j.

Definition 30. *A piece wise linear trajectory is defined as $T^i_{pl} : (s^i_1, \ldots, s^i_{n^i-1})$, where $s^i_j = [p^i_j, p^i_{j+1}] \in \mathbb{R}^4$ and n^i_{pl} is the length of the piece wise linear trajectory.*

The length of the PL-trajectory n^i_{pl} is the sum of the lengths of all segments that compose it : $n^i_{pl} = \sum_{j \in [1 \ldots n^i - 1]} \|p^i_j p^i_{j+1}\|_2$.

In a previous paper[Besse et al., 2016], we proposed a method to cluster trajectories based on the behaviours of the users. This clustering is obtained by *hierarchical clustering* with the *ward* linkage criterion based on a distance between trajectories, the *Symmetrized Segment-path-Distance*.

Definition 31. *Symmetrized Segment-Path Distance*

$$D_{SSPD}(T^1, T^2) = \frac{D_{SPD}(T^1, T^2) + D_{SPD}(T^2, T^1)}{2}.$$

where

$$D_{SPD}(T^1, T^2) = \frac{1}{n_1} \sum_{i_1=1}^{n_1} D_{pt}(p^1_{i_1}, T^2).$$

and where $D_{pt}(p, T)$ is the minimum distance from the point p to the piecewise representation of the trajectory T.

This distance compares trajectories as a whole, regardless of their time indexing or the number of locations that compose them. It enables us to produce clusters of trajectories describing the traffic flow of the trajectory set $\mathcal{T}$. The partition, $\mathcal{C}(\mathcal{T})$ of $\mathcal{T}$ shows the

Table 2.1 – Notation

$\mathcal{T}$	The set of trajectories
T^i	The i^{th} trajectory of set $\mathcal{T}$
n^i	Number of locations in trajectory T^i
p^i_j	The j^{th} location of T^i
t^i_j	The time index of location p^i_j
l^i	Label of the i^{th} Trajectory
$\mathcal{T}^m = \{T^i \vert l^i = m\}$	Set of trajectories in cluster m
$\mathcal{P}^m = \{p^i_j \vert l^i = m\}$	Set of points in cluster m
$\mathcal{C}(\mathcal{T}) = \{\mathcal{T}^1, \ldots, \mathcal{T}^K\}$	Set of cluster of trajectories
K	Number of clusters of trajectories in $\mathcal{C}(\mathcal{T})$
l^i_j	Label of the location p^i_j
$\mathcal{P}^m_n = \{p^i_j \vert l^i_j = m, l^i = n\}$	Set of points in cluster (m, n)
$\mathcal{C}(\mathcal{P}^m) = \{\mathcal{P}^m_1, \ldots, \mathcal{P}^m_{k^m}\}$	Set of points cluster
k^m	Number of clusters of points in $\mathcal{C}(\mathcal{P}^m)$
$\#S$	Number of elements in the set S

K main paths taken by the users. The number of clusters K, is not fixed. It depends on the dataset, or the precision we want to use in order to described it. In Section 2.1.5, we discuss the choice of K for both datasets according to the values of the different quality criteria described in the same Section.

2.1.4 A Probabilistic Model for Trajectory Classification

After obtaining clusters of trajectories that discriminate the main patterns of the traffic flow in the city, we aim to predict the final destination of a vehicle for which we only observe the beginning of its path. Hence, We observe a succession of locations in $\mathbb{R}^2$. To assign these points to a cluster of trajectories, we model the clusters by a mixture of 2d Gaussian distributions. Thus, for each cluster, we obtain a Gaussian likelihood estimated using only the data belonging to this cluster. The observed locations will then be assigned to the most likely cluster according to these different likelihoods.

2.1.4.1 Points Partitioning within Clusters of Trajectories

In the previous Section we have obtain K clusters of trajectories, $\mathcal{T}^m$, for $m \in [1 \ldots K]$. For a new trajectory, we want to be able to predict its final destination. We only observe

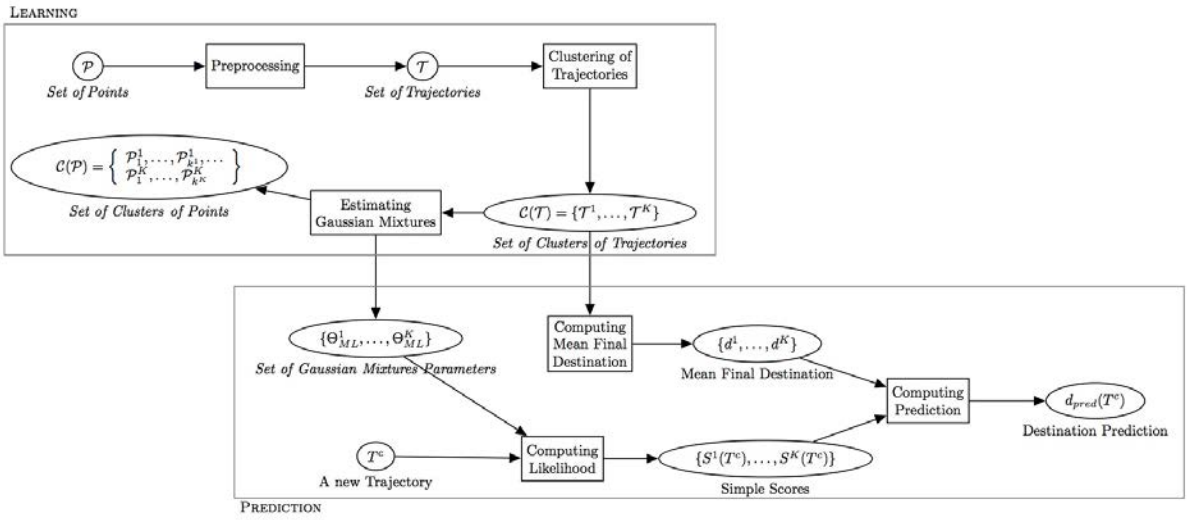


Figure 2.1 – Structure of the Learning and Prediction steps of the system

the beginning of its path, which is represented as a succession of locations in $\mathbb{R}^2$, and we want to evaluate its probability to belong to the each cluster of trajectory. For that we will then consider the set of points $\mathcal{P}^m$ which is composed of all the points that composed the trajectories within the m^{th} cluster of trajectories, $\mathcal{T}^m$. We build a Gaussian mixture model for each of this set of points, $\mathcal{P}^m$. The Gaussian mixture model will then produce a partitioning of each set of point $\mathcal{P}^m$, to k^m subsets: $\mathcal{P}_n^m$ for $n \in [1, \ldots, k^m]$. Hence we denote by m the mth cluster of trajectory and by (m, n) the n^{th} cluster of point within the set of point, $\mathcal{P}^m$. These notations are resumed in Table 2.1, and we can observed Figure 2.1, the diagram of the different steps of methodology.

A Gaussian mixture model assumes that all points from $\mathcal{P}^m$ are generated from the sum of k^m Gaussian distributions ϕ , which are, in our case, 2-variate Gaussian distributions.

Definition 32. *A Gaussian Mixture Model is a weighted sum of k^m component Gaussian densities as given by the equation,*

$$\Phi^m(p) = \Phi(p|\Theta^m) = \sum_{k=1}^{k^m} \omega_k^m \cdot \phi_k^m(p),$$

where ω_k^m is the mixture weight, i.e. the prior probability for any point p belonging to the k^{th} cluster, such that $\sum_{k=1}^{k^m} \omega_k^m = 1$, and $\phi_k^m(p)$, $i = 1, \ldots, k^m$ are the component Gaussian densities.

Each component density is a Gaussian density on $\mathbb{R}^2$.

Definition 33. *The density function of a normal distribution, ϕ_k^m, is defined as,*

$$\begin{aligned}\phi_k^m(p) &= \phi(p|\mu_k^m, \Sigma_k^m) \\ &= \frac{1}{\sqrt{(2\pi)^2|\Sigma_k^m|}} e^{\left[-\frac{1}{2}(p-\mu_k^m)^{tr}\cdot(\Sigma_k^m)^{-1}\cdot(p-\mu_k^m)\right]},\end{aligned}$$

where $\mu_k^m \in \mathbb{R}^2$ and $\Sigma_k^m \in \mathbb{R}^{2x2}$ are respectively the location and the covariance matrix of ϕ_k^m, and $|\Sigma_k^m|$ is the determinant of the matrix Σ_k^m.

$\Theta^m = \{\omega_1^m, \mu_1^m, \Sigma_1^m, \ldots, \omega_{k^m}^m, \mu_{k^m}^m, \Sigma_{k^m}^m\}$ is the list of parameters of the GMM distribution Φ^m.

To evaluate the parameters Θ^m, we use the maximum likelihood estimation. Its aim is to find the parameters which maximise the likelihood function of Φ^m, given the training set $\mathcal{P}^m$. The *GMM* likelihood, can be defined as,

$$\mathcal{L}(\Theta^m|\mathcal{P}^m) = \prod_{p\in\mathcal{P}^m} \Phi^m(p). \tag{2.1}$$

The maximum likelihood estimators, Θ_{ML}^m, are the parameters which maximise the *GMM* likelihood function.

$$\Theta_{ML}^m = \arg\max_{\Theta} \mathcal{L}(\Theta^m|\mathcal{P}^m). \tag{2.2}$$

For each *GMM*, this maximum likelihood estimators, Θ_{ML}^m, is estimated for different values of k, from 1 to 20. The number of components k^m is then set to the value which maximise the criterion information $BIC = -2\ln L(\Theta^m|\mathcal{P}^m) + k\ln(|\mathcal{P}^m|)$.

$$k^m = \arg\max_{k\in[1,\ldots,20]} BIC(k). \tag{2.3}$$

We used the BIC criterion instead of the AIC criterion because it enabled to producing models using fewer parameters while producing similar results. The complete set of trajectories is then modelled by the set of K *GMM*'s, one for each set of points, $\mathcal{P}^m$. Each of these sets has been partitioned into k^m groups : $\mathcal{C}(\mathcal{P}^m) = \{\mathcal{P}_1^m, \ldots, \mathcal{P}_{k^m}^m\}$. Using this modelling procedures, we obtain several cluster of locations, each one corresponding to a mode of the estimated Gaussian mixture distribution. We got a density based clustering of the cloud points, which produces a data driven grid of similar points within a cluster of trajectories.

Now that we have described the space, we want to use the model to predict the final destination of a new trajectory in progress: T^c. We consider this trajectory at a given time t_n^c at which the trajectory is composed of n locations, *i.e.*, $T^c : [(p_1^c, t_1^c), \ldots, (p_n^c, t_n^c)]$.

For that, we want to be able to assign the new trajectory to the cluster of trajectories it most likely belongs. For this purpose we compute the *simple score*, $s^m(T^c)$ for all the *GMM*s Φ^m. The score is the value of the likelihood function of Φ^m given the points that compose the trajectory T^c. It represents how likely the trajectory T^c belongs to the cluster m.

Definition 34. *The simple score, $s^m(T^c)$, for a trajectory, T^c, to be assigned to the cluster m is defined as:*

$$\begin{aligned} s^m(T^c) =& \mathcal{L}(\Theta^m_{ML}|T^c) \\ =& P(T^c|\Theta^m_{ML}) \\ =& \prod_{j=1}^{n} \Phi^m(p^c_j|\Theta^m_{ML}) \end{aligned} \tag{2.4}$$

In this way, we can assigned the trajectory to the cluster with the highest simple score.

$$l^c_{guess} = \max_{m\in[1...K]} s^m(T^c). \tag{2.5}$$

We highlight the fact that this method enables us to compute a score for the trajectory and for each cluster. The trajectory is not attributed to one cluster and one cluster only. This is relevant because when only a few points of the trajectory are known, we cannot always be totally certain of the final destination. Several destinations are possible. Hence the score computed is a probability that the trajectory belongs to the cluster of trajectories. We also emphasise that this *simple score* can easily be updated if we have more information about the trajectory in progress T^c. Hence if we consider the simple score, $s^m(T^c)$, computed at time t^n at which the trajectory is composed of n locations, and a new location p^c_{n+1} of the trajectory T^c observed at time t^c_{n+1}, we can estimate the updated value $s^m_*(T^c)$ of the *simple score* according to this equation:

$$\begin{aligned} s^m_*(T^c) =& \prod_{j=1}^{n+1} \Phi^m(p^c_j|\Theta^m_{ML}) \\ =& s^m(T^c) \cdot \Phi^m(^c_{j+1}|\Theta^m_{ML}) \end{aligned} \tag{2.6}$$

It is then quite easy to update the *simple score* between a trajectory in progress and the different cluster of trajectories. This characteristic will allow us to then apply prediction in real-time.

2.1.4.2 Complete Model

We want to test the influence of auxiliary variables on the quality of our classification method, such hour of the day, or day of the week, during which the trip takes place.

The likelihood score as defined in Definition 34 does not take into account contextual information. However, we can assume that prior knowledge may help to discriminate the trajectories. Indeed, a path may more likely be taken than an other at a given hour of the day or day of the week. We look forward to verifying this hypothesis by including *auxiliary weights*. For this we define a new *complete score* taking into account the following weights.

Definition 35. *The complete score, $s_c^m(T^c)$, for a trajectory, T^c, to be assigned to the cluster m is defined as:*

$$\begin{aligned} s_c^m(T^c) &= \mathcal{L}_{ap}(\Theta_{ML}^m | T^c, E^c) \\ &= P(E^c|\Theta_{ML}^m) P(T^c|\Theta_{ML}^m) \\ &= \alpha(m, h^c, d^c) \prod_{p_j^c \in T^c} \Phi^m(p_j^c|\Theta_{ML}^m) \end{aligned}$$

The definition of the *complete score*(35) is generic. $h^c \in [0, \ldots, 23]$ and $w^c \in [1, \ldots, 7]$ are respectively the hour of the day and the day of the week at which the trajectory T^c begins. This information can be interpreted in different manners which will result in a different value for the *auxiliary weight* $\alpha(m, h^c, w^c)$ according to the information we are taking into account. We define three different weights:

- The *Empirical weight* describes the distribution information of the trajectory cluster.

$$a_{emp}(m) = \frac{\#\mathcal{T}^m}{\#\mathcal{T}}.$$

- The *Weekday weight* describes the distribution information of the trajectory cluster at a given day of the week,

$$a_{wd}(d, m) = \frac{\#\{T^c \mid T^c \in \mathcal{T}^m, d^c = d\}}{\#\{T^c \mid T^c \in \mathcal{T}, d^c = d\}}.$$

- The *Hours weight* describes the distribution information of the trajectory cluster at a given hour of the day,

$$a_h(h, m) = \frac{\#\{T^c \mid T^c \in \mathcal{T}^m, h^c = h\}}{\#\{T^c \mid T^c \in \mathcal{T}, h^c = h\}}.$$

The *auxiliary weight* $\alpha(m, h^c, w^c)$ is the product of any combination of these weights.

2.1.4.3 Model for Final Destination Prediction

We present here how our model can be used to predict the final destination of the user trips. We have defined, in the previous sections, a *simple score* and a *complete score* for each trajectory to belong to a cluster of trajectory. Hence we can assign the new trajetory to the clusters it most likely belongs. We can then use the information from the trajectories that compose these clusters to predict the final destination of the new trajectory. From this, we define two different methods for predicting final destination.

On the one hand, we consider only the trajectories from the cluster m with the highest score.

$$\begin{aligned} d_{pred_1}(T^c) &= \frac{1}{\#\mathcal{T}^m} \sum_{\substack{i \ s.t. \\ l^i = m}} p^i_{n^i} \\ &= d^m, \ s.t. \ m = l^c_{guess}, \end{aligned} \tag{2.7}$$

where d^m is the mean of the locations of all final destinations of the trajectories in cluster $\mathcal{T}^m$, and l^c_{guess} is determined according to Equation 2.5.

On the other hand, in order to take advantage of the fact that the trajectory T^c is not strictly assigned to one and only one cluster, we define, d_{pred_2}, as a weighted sum of the mean final destination of every cluster.

$$\begin{aligned} d_{pred}(T^c) &= \sum_{m=1}^{K} \frac{s^m(T^c)}{\sum_{k=1}^{K} s^k(T^c)} \cdot d^m \\ &= \sum_{m=1}^{K} s^m_w(T^c) \cdot d^m, \end{aligned} \tag{2.8}$$

where $s^m_w(T^c)$ is the *weighted affinity score* of $s^m(T^c)$ The *complete score* can be used instead of the *simple score* to take into account the effect of the auxiliary variable in the final destination prediction. Both final destination formulas are computed using the mean of the locations of all final destinations d^m which is a constant, and the *simple score*, $s^m(T^c)$, which, as we have seen Equation 2.6.

2.1.5 Experimental Results

In this section we present experimental results to evaluate both classification and final destination's prediction methods. To evaluate prediction error, we use the *Haversine Distance* (see Definition 38), which is the evaluation metric used in the Kaggle competition [kaggle, 2015]. The Haversine Distance measures distances between two points on a sphere based on their latitude and longitude. We use a 10-cross validation method to calculate this error by learning on 90% the data: the training set $\mathcal{T}_{train}$, and forecasting the remaining 10%: the test set $\mathcal{T}_{test}$. The error forecast is the average for all the training sets. We

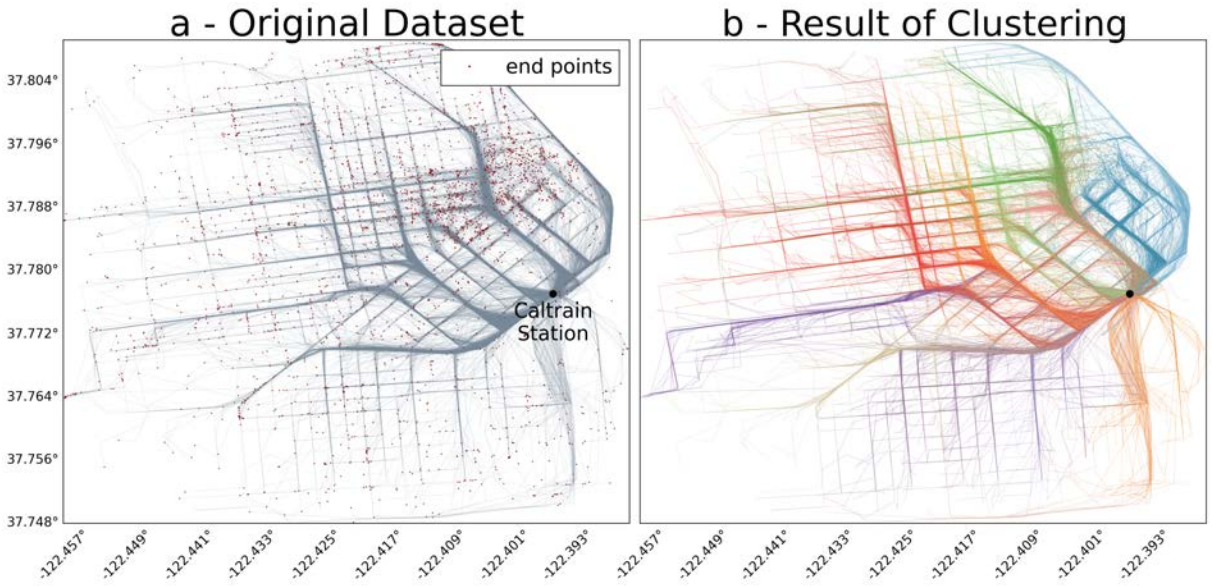


Figure 2.2 – Caltrain Station, San Francisco Dataset and its Partitioning in 25 clusters

repeat this operation ten times, such that every set has been considered as the test set, to ensure a more accurate estimation of model prediction performance.

To evaluate our method during trajectory completion, we introduce the definition of a partial trajectory, below. A p-trajectory, $T^i(p)$ of a trajectory T^i is a subset of this trajectory such that the length of the piecewise representation of $T^i(p)$ is at most p times the size of the length of the piecewise representation of T^i.

Definition 36. *The p-trajectory* $T^i(p), \forall p \in [0, \dots 1]$ *is defined as the trajectory:*
$T^i(p) = ((p^i_1, t^i_1), \dots, (p^i_{n^i(p)}, t^i_{n^i(p)}))$ *s.t.* $\frac{n^i_{pl}(p)}{n^i_{pl}} \leq p$
where $n^i(p)$ *is the number of locations that compose the p-trajectory* $T^i(p)$.

2.1.5.1 Data And Clustering Results

To analyse our result and test its scalability, we test our model on two different subsets. The first one is a subset of taxi trajectories from San Francisco[Piorkowski et al., 2009]. It is composed of $4,127$ trajectories, all starting from the Caltrain Station and ending in an area of size 6.327×6.827 km in the center of the city. The second is composed of $19,423$ trajectories from taxis in the center of Porto [kaggle, 2015], leaving from the Sao Bento Station and ending in a delimited area of size 8.116×8.068 km. These two datasets are displayed on the left in Figure 2.2 and Figure 2.3.

These two datasets are quite different. In San-Francisco, the road network looks like a grid, most of the streets are either parallel or perpendicular to each other. In Porto the

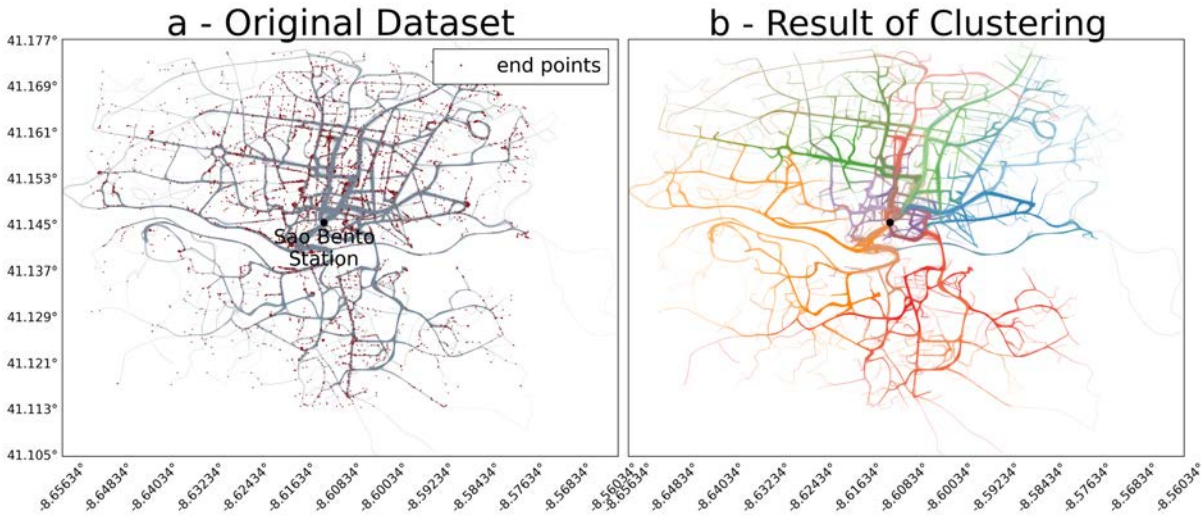


Figure 2.3 – Sao Bento Station, Porto Dataset and its Partitioning in 45 clusters

network is more irregular. These differences will enable us to test the scalability of our method and its capacity to be adapted on different datasets.

On figure 2.2 and 2.3 we can see the results of the clustering described in section 2.1.3 on the data sets from San Francisco and Porto. We can see that the clustering obtained results in a group of trajectories tracing the same path from the selected departure points.

2.1.5.2 Trajectory Classification

To evaluate the quality of our classification, we observe the percentage of trajectories that have been assigned to their true cluster.

Definition 37. *The quality criterion, Q_{class}, for the classification, is the percentage of well classified p-trajectories* $, \forall p \in [0, \dots 1]$, *defined as:*

$$Q_{class}(p) = \frac{\#\{T^i(p) | l^i_{guess} = l^i, T^i \in \mathcal{T}\}}{\#\mathcal{T}}$$

In Figure 2.4, we can observe the percentage of well classified trajectories, Q_{class} , for $p = 1$, *i.e*, when the trips are completed, with respect to the number of clusters of trajectories. These values are represented with a triangle in the Figure. We could have expected that the quality of the classification decreases when the number of clusters increases. But we can see that we obtain better results with 20 clusters than 5 clusters for both the *San Francisco* and *Porto dataset*. When the number of clusters of trajectories is low, the points that compose the trajectory are scattered. Hence the clusters of points found with the Gaussian Mixture have covariance matrices with high values resulting in low likelihoods and low scores. This means that a trajectory can have a low score with

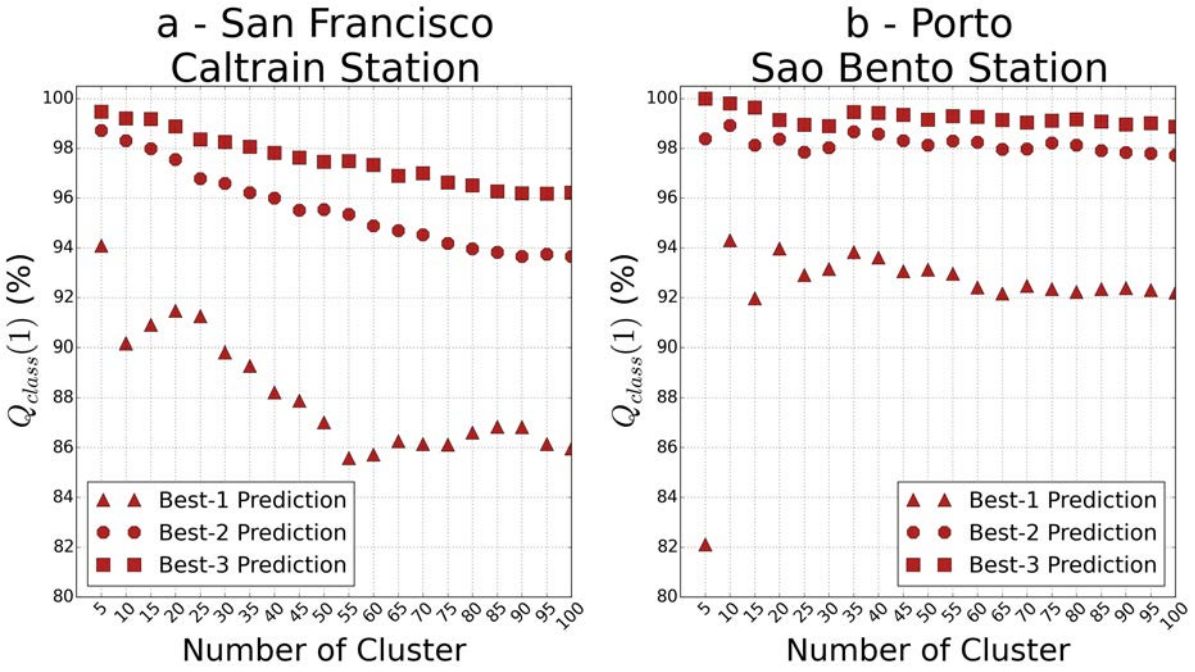


Figure 2.4 – Percentage of Trajectories Correctly Classified According to Number of Clusters. Compare Best-3 Prediction.

respect to its correct cluster. This explains the behavior of the quality criterion Q_{class}, for a number of clusters of trajectories between 5 and 25. The behavior of the quality criterion is more intuitive between 25 and 100 clusters for both datasets. We can observe that the percentage of well classified trajectories is always higher than 85% for the San Francisco dataset and higher than 91% for the *Porto dataset* between 10 and 100 clusters . Moreover, we have displayed in Figure 2.4 the percentage of trajectories for which the cluster they belong to appears among the first two cluster predictions (represented with the circle) and among the first three cluster predictions (represented with the square) according to our classification method. For the *San Francisco dataset* the percentage of trajectories for which the correct cluster of trajectories appears among the first two prediction is at least 95% when the number of clusters is below 55 and at least 93.5% otherwise. This percentage is greater than 96% when we looked at the best three predictions. For the *Porto dataset* the percentages is greater than 97.5% for the best two predictions and greater than 99% regardless of the clusters size. To asses this results it is relevant to look at the behavior of the *simple sore* of the correct cluster to see if in addition to being among the best predictions, the values of the scores of the correct cluster is high. For this reason, we consider a one-vs-all classification for every cluster of trajectories produced by our method.

In Figure 2.5, we can observe the *ROC* curves for these one-vs-all classifications and

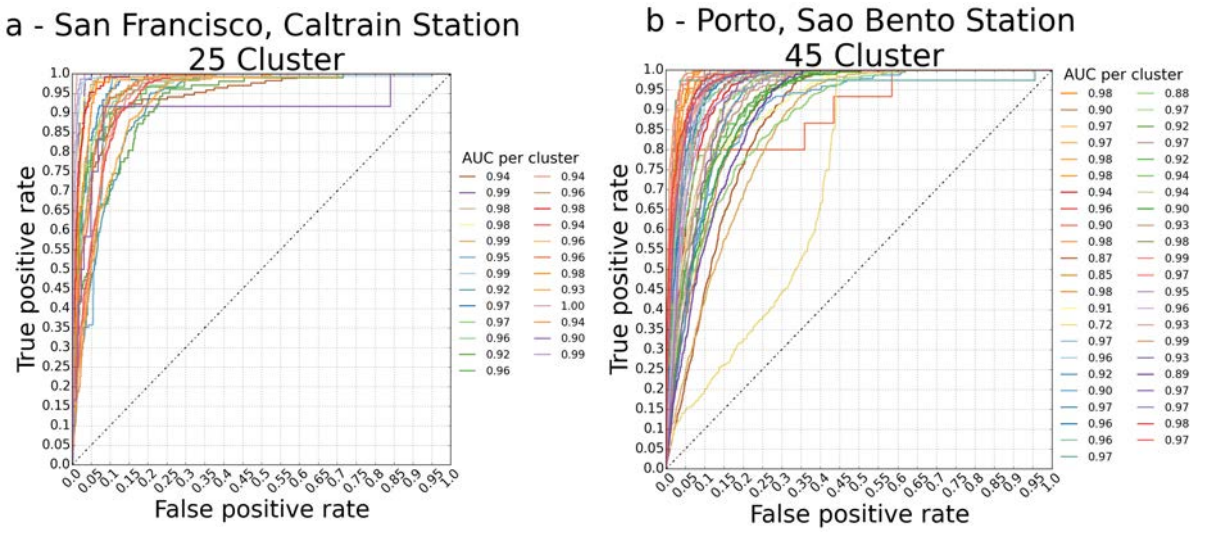


Figure 2.5 – ROC Curves and AUC for every clusters of trajectories.

for both datasets. We have selected a number of 25 clusters for the *San Francisco dataset* and a number of 45 clusters for the *Porto dataset* according to the results described in the following section. The legends of these plots indicate the AUC (Area Under Curves). All AUC are greater than 0.90, and 17 are greater than 0.95 for the *San Francisco dataset* while all but three are greater than 0.90 and 28 are greater than 0.95 for the *Porto dataset*. These results show that even if some trajectories are not assigned to the correct clusters, the *simple scores* for their correct cluster is always elevated. Hence we have seen that the classification procedure, which is the first step of our prediction method (c.f Figure 2.1), is effective and enables us to associate a trajectory to the cluster of trajectories it most likely belongs.

2.1.5.3 Final Destination Prediction

In this section, we present the results of our method for the prediction of taxi trips destination. To evaluate our method we used the mean of the *Haversine Distance*, which measures distances between two points on Earth based on their latitude and longitude.

Definition 38. *The Haversine Distance,* D_H *between two locations* $d1, d2 \in \mathbb{R}^2$ *is defined as:*

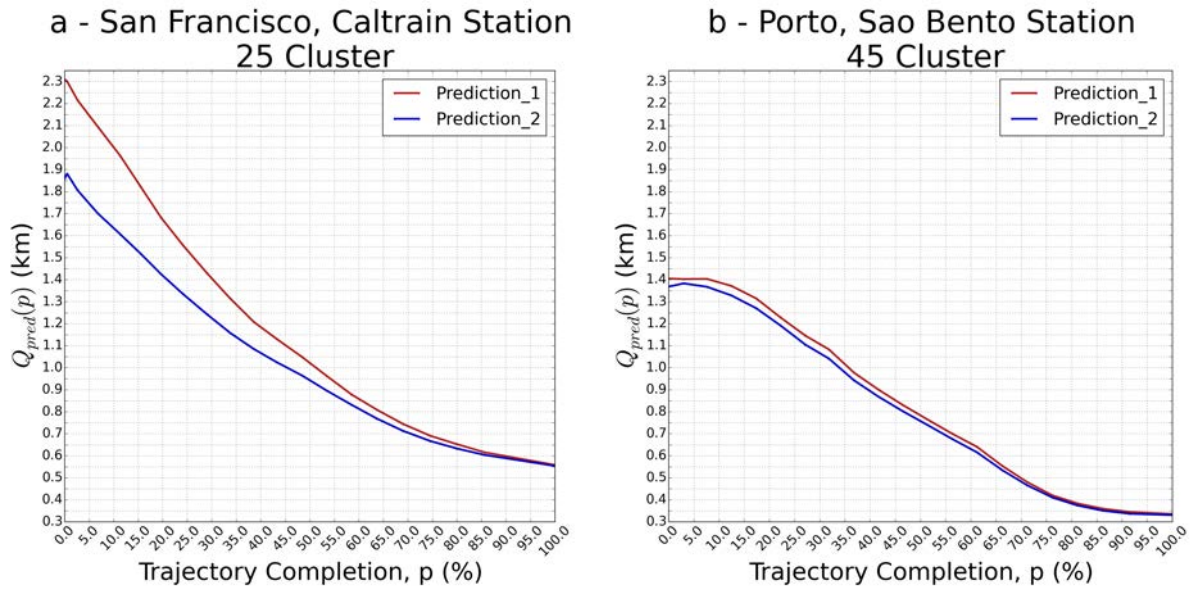


Figure 2.6 – Mean Error of Final Destination Prediction According to Trajectory Completion. Compare Method 1 and 2.

$$D_H(d1, d2) = 2 \cdot r \cdot \arctan\left(\sqrt{\frac{a}{1-a}}\right)$$
$$a = \sin^2\left(\frac{y_2 - y_1}{2}\right) + \cos(y_1)\cos(y_2)\sin^2\left(\frac{x_2 - x_1}{2}\right)$$

where (x_1, y_1) *and* (x_2, y_2) *are the longitude and latitude of* $d1$ *and* $d2$ *respectively and* $R = 6,371(km)$ *is the radius of the Earth. Hence, the Haversine distance returns the distance in Km between two locations on Earth.*

We can then define the quality criterion for the prediction of the final destination as the mean of the *Haversine* distance between the true location of the final destination, $p^c_{n_c}$, of the trajectory, T^c, and the location of the prediction, $d_{pred}(T^c)$.

Definition 39. *The quality criterion,* Q_{pred}*, is defined as :*

$$Q_{pred}(p) = \frac{\sum_{T^c \in \mathcal{T}} D_H(d_{pred}(T^c(p)) - p^c_{n_c})}{\#\mathcal{T}}$$

In figure 2.6 we can observe the results of the quality criterion, Q_{pred}, according to the trajectory completion. We compare the results of the two prediction methods, $pred_1$

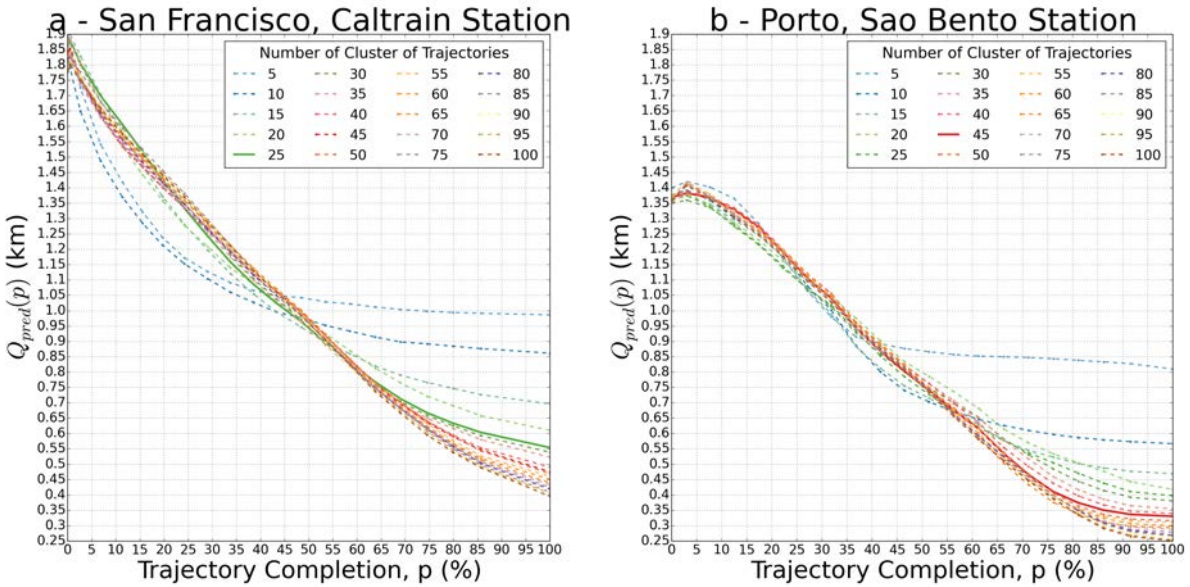


Figure 2.7 – Mean Error of Final Destination Prediction According to Trajectory Completion. Compare Number of Cluster for Method 2

and $pred_2$. Both datasets are displayed, San Francisco(a), on the left and Porto(b), on the right, for a number of clusters of 25 and 45 respectively. For San-Francisco, we can observe that the second method gives best results especially at the beginning of the trajectories where Q_{pred} is 400 meters better using $pred_2$. As the trajectories progress, the results continue to be better with $pred_2$ but the difference between the two methods decreases and after 50% of trajectory completion, the difference is less than 50 meters. This is expected because the more locations we know for a trajectory, the more confidently we can assign the trajectory to one cluster and one cluster only. Hence, the more locations we know for a trajectory, the more closely the results are using the two methods. For trajectories in Porto, if Q_{pred} also gives better results with $pred_2$, the difference between the method is insignificant. Nevertheless, we will still use the $pred_2$ to compare results according to the number of clusters.

In Figure 2.7, we can look at the same quality criterion, Q_{pred}, according to trajectory completion. We display these results for different numbers of clusters from 0 to 100. For dataset in San Francsico (a), and for trajectory completion between 0% and 50% the bests results are found for 5 and 10 clusters. At these completion rates, the trajectories are more easily assigned to their correct cluster, leading to best results. For completion rates between 50% and 100% the results found with 5 and 10 clusters are the worst, because these numbers of clusters do not well enough describe the space. The same conclusion can be made for 15 and 20 clusters. Up until 70% of trajectory completion, there is no strong differences for a number of clusters between 25 and 100. When all the trajectories

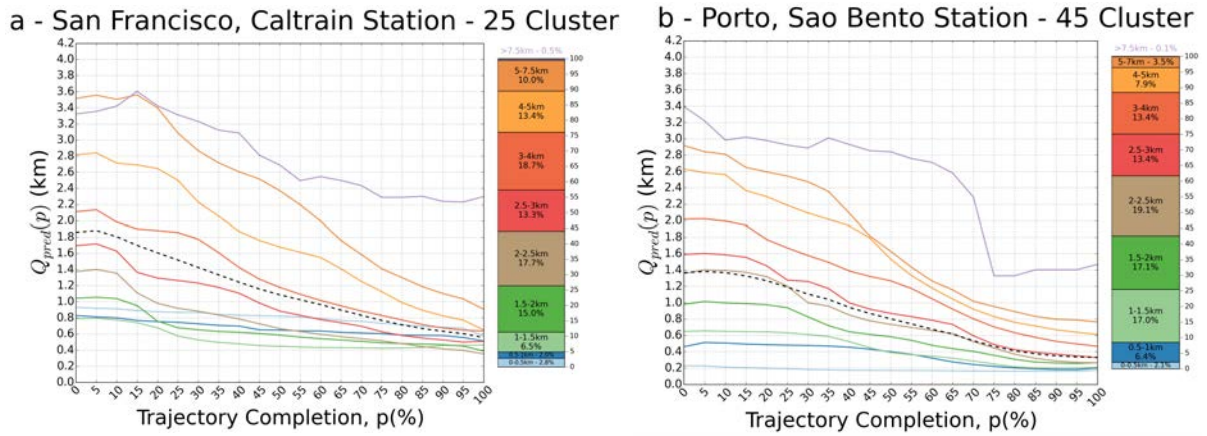


Figure 2.8 – Mean Error of Final Destination Prediction According to Trajectory Completion. Compare Length of Trip

are completed, the more clusters we have, the more precise the prediction is. However, we have a gain of precision of only 200 meters between 25 and 100 clusters. The more clusters of trajectories we have, the more Gaussian Mixture we need to estimate. Hence 25 clusters of trajectories is the best compromise to well describe the space and to do so within a reasonable computation time. For Porto dataset(b), the worst results are found for a number of 5 clusters, for trajectory completions between 0% and 15% and between 40% and 100%. For trajectory completions from 35% to 55%, the best results are for a number of clusters of 10 and 15, but they yield bad results after 65% and 80% trajectory completions. The same conclusion can be made for a number of clusters between 20 and 40. We can observe that the results stabilise when the number of clusters increases from a number of cluster of 45. The difference of Q_{pred} value for a number of clusters between 45 and 100 does not exceed 40 meters for trajectory completions from 0% to 80%. Similarly to San-Francsico, when all the trajectories are completed, the more clusters we have, the more precise the prediction is, but the gain of precision is low, only 100 meters between 45 and 100 clusters. Hence, 45 is the best choice for the Porto dataset.

In Figure 2.8, we can observe the evolution of the Q_{pred} criterion according to the trajectory completion for the number of clusters of trajectories selected and for different groups of trajectories within both datasets. Trajectories are grouped together according to the length of their piecewise representation. The legend color is displayed on the right of each plot along with the percentage of trajectories which composed each group. First of all, we can see some differences of the repartition of the total length of the trajectories between both datasets. For the *San Francisco Dataset*, there are more than 55% of the trajectories that have a length above 2.5km and more than 25% above 4km. For the *Porto Dataset*, more than 60% trajectories are below 2.5km while only 12% are above 4km. This value, along with the differences of sampling rate, can explain the différence between the

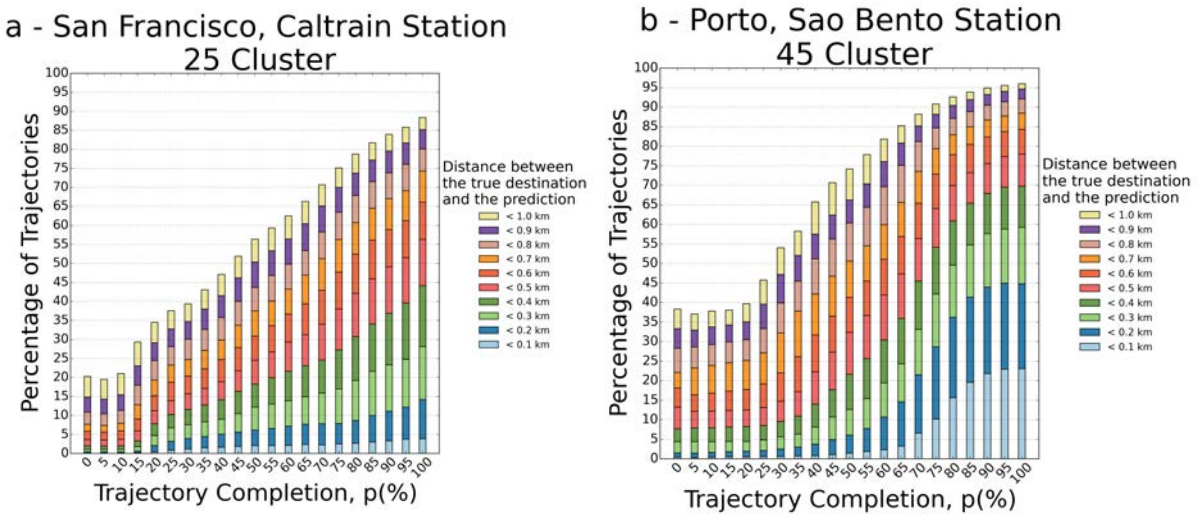


Figure 2.9 – Percentage of Trip under different distance error according to trajectory completion

mean values of the Q_{pred} criterion for both datasets. The major conclusion that we can draw from these plots is that the behavior of the Q_{pred} criterion is similar for a group of trajectories with same length for both datasets except for the two groups with length between 0 and 0.5km and between 0.5 and 1km. For these two groups, the performance of the prediction for the *San Francisco dataset* is worse than the prediction for the group of trajectories with lengths between 1 and 1.5km. These results can be explained by the small number of trajectories that compose these two groups (below 100). However, for all other groups the evolution of the Q_{pred} criterion is similar for both datasets. For these groups we can observe that the shorter the trajectories are, the better the prediction is. These result prove that as soon as there are enough representative trajectories within the learning set (about 500 trajectories) of a given type of trajectory, our algorithm is able to adapt to datasets with different characteristics and to provide similar results. We can also observe that the difference of error prediction is quite large from one group to another at the beginning of the trajectory, and is directly related to the total length of the trajectories. This is expected because at the beginning of the trajectories, we have little information about them, and the prediction of the final destination is almost the same for all trajectories. But this difference reduces quickly as trajectory progress. For example, if we compare the group of trajectories between 1km and 1.5km and the group of trajectories between 3km and 4km for both datasets, the difference of the Q_{pred} criterion between these two groups is about 1.2km when $p = 5\%$. This difference is about 0.85km for $p = 45\%$ and about 0.6km for $p = 65\%$. These results prove the capacity of our algorithm to adapt to different types of trajectories with different characteristics.

In Figure 2.9, we can see in detail the performance of the final destination prediction

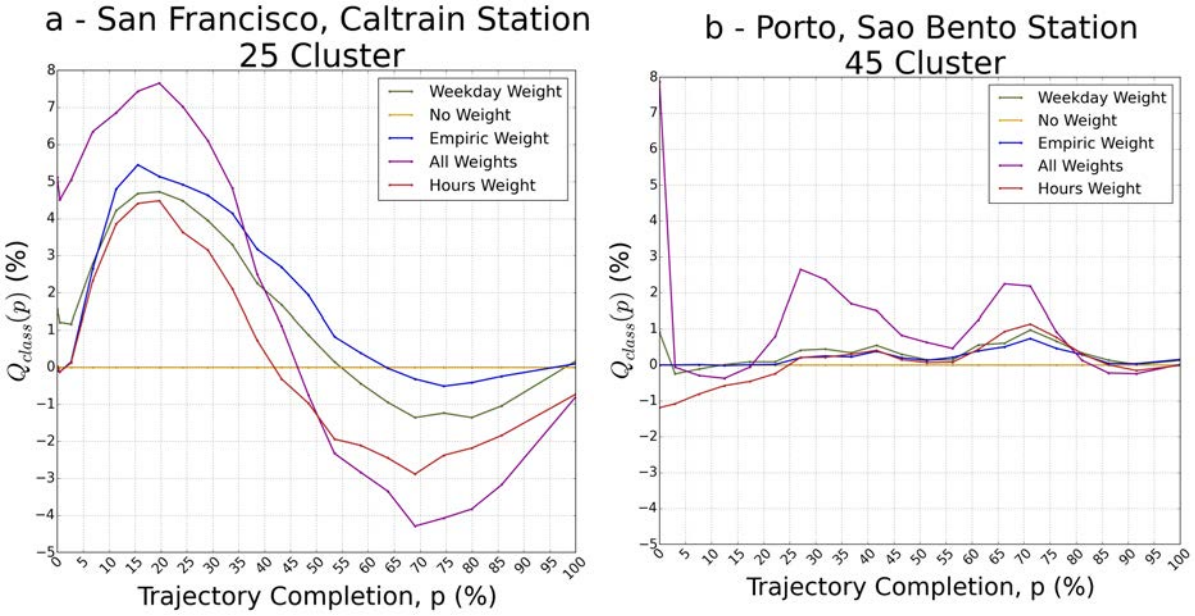


Figure 2.10 – Improvement of Trajectory Classification With auxiliary Information

for the number of clusters selected for both datasets. For different percentages of trajectory completion, we can observe which percentage of trajectories is below a given error distance between the final destination prediction and the true destination of the trajectories. We can observe that the prediction results do not evolve much for trajectory completion between 0% and 15% for the *San Francisco dataset* and between 0% and 20% for the *Porto dataset*, which means that at this point we do not have enough information fo improve the quality of the prediction. After these intervals, we can see that the prediction is regularly improved as the trajectory progress.

2.1.5.4 Effect of Auxiliary Information on Classification and Prediction

In Figure 2.10 and Figure 2.11, we observe the effect of different auxiliary weights described in Section 2.1.4.2 on both the quality criteria Q_{class} and Q_{pred}. We display the differences of these criteria with the different weights and the same criteria with no weights according to trajectory completion. For San-Francisco, we display the results for 25 clusters. We can observe that all weights result in an improvement on both the quality of the classification and the prediction of the final destination in the first part of the trajectories, for trajectory completion between 0% and $35\% - 45\%$. The mix of all the weights is yields the best results. The improvement of classification continues until 8% when trajectory completion is at 25% and the improvement of the Q_{pred} criteria is 225 meters when the trajectory starts and 100 meters at 25% of trajectory completion. The curves of both quality criteria, Q_{class} and Q_{pred} are not perfectly correlated. This is expected because Q_{class} shows the

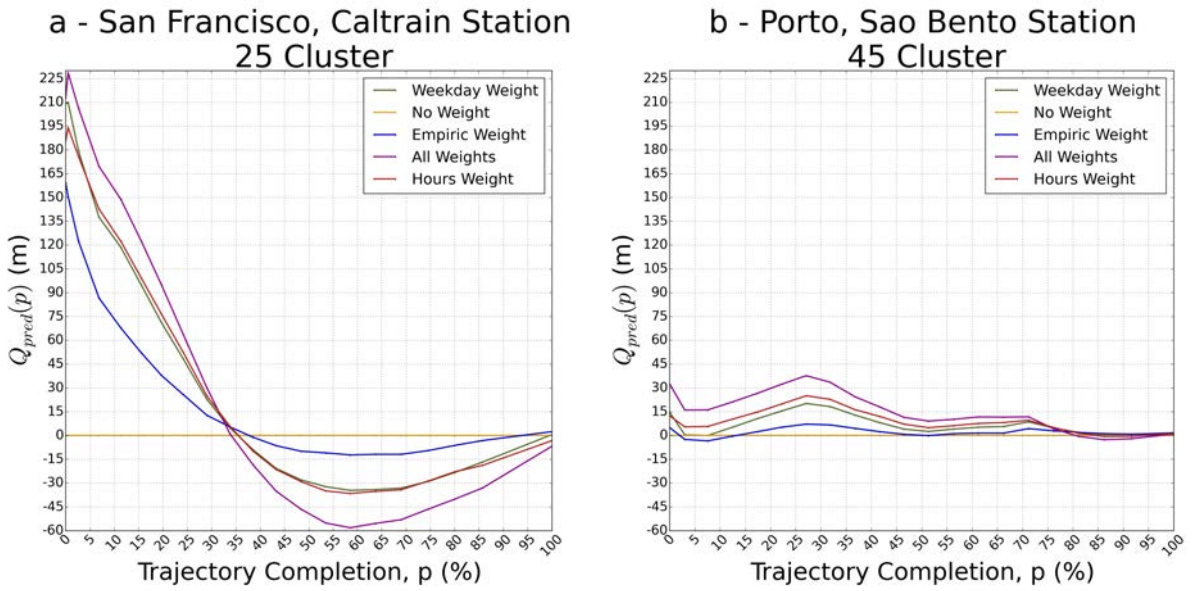


Figure 2.11 – Improvement of Prediction of Final Destination With auxiliary Information

rate of correct classification, while Q_{pred} displays the prediction quality found with $pred_2$, which uses information of different clusters and not only the first predicted cluster for the prediction. Beyond these completion rates, the auxiliary weights deteriorate according to the different quality criteria values. This means that when we have little information about the location of the trajectory, context information help to improve the destination prediction. Whereas when we have sufficient information about the trajectory location, we can confidently predict the correct clusters of trajectories the new trajectory most likely belongs to. Hence, adding auxiliary weight information deteriorates the result.

The results obtained are different using the Porto dataset. The results for Porto datasets are displayed for 45 clusters. The different weights improve the prediction, and the mix of all weights yields the best results, but the improvement is always less than 30 meters which is much less significant than with the San-Francisco dataset. Similarly, the classification is never improved more than 3%. Taxi trips in Porto are less influenced by auxiliary variables than taxi trips in San Francisco.

2.1.6 Discussion

2.1.6.1 Kaggle

We have tested our method on the test dataset from the Kaggle challenge. This competition is over but we can still submit an entry to see our score. We have applied our models on two different subsets of the learning datasets.

- The first has been build to match the trajectory within the test dataset. For that we have selected top 100 nearest neighbors of each trajectory within the test dataset according to the *SPD* distances from the test trajectories, to the trajectories within the original learning dataset. With this subset of the learning dataset, our final destination prediction method produces a mean error of 2.36623 kilometres on the test dataset and would have ranked us 38 out of 381 submissions.

- The second is a random subset taken within trajectories whose starting points are within an area including all starting points of the test trajectories. With this subset of the learning dataset, our final destination prediction method produces a mean error of 2.82021 kilometres on the test dataset and would have ranked us 242 out of 381 submissions.

We have drawn two conclusions from these results. First of all, if we prepare our learning datasets to match most of the the trajectories within the test dataset, as the other methods did, we proved that our method is very competitive with a ranking among the first deciles. Of course our method does not compete with the winning solution, which uses deep learning and produces an error of 2.03489 kilometres. However, once our model has been learned, our methodology can take into account new location of the test trajectories during its completion without needing to produce a new learning, which it is not the case withe the wining solution. Secondly, our method also produced good results considering that the learning dataset has not been developed to match the test dataset, which is the case in real applications where we cannot afford to fit the learning dataset to the trajectory we want to predict. Hence our model can be re-used directly for a different test dataset, and can also be used to predict the destination within the same trajectory, without requiring a new training, something which other methods do not allow.

2.1.6.2 Advantages of our models

Our method was designed to provide a forecast based on rigid models learnt using clusters of trajectories. It provides a deep understanding of the main streams and paths of vehicles in a city that reflect the behavior of drivers. We used this model as a prediction model and compared it to one used in one of the most competitive competitions in the machine learning field. We were not surprised to be outperformed by deep learning methods. Yet our forecast can be easily used to provide models of road behavior that explain the prediction obtained, as we have shown in the different Figures, Section 2.1.5. In Figure 2.12, one can observe an example of how our model for final destination prediction works for a trajectory selected randomly among the *San Francisco Dataset*. The probability of different final destination points for a trajectory at 6 different rates of trip completion is displayed in this figure. At each moment, we can observe the probability of belonging to each cluster of trajectories and their corresponding final destinations. The more likely

that the trajectory belongs to a cluster, the more visibly this cluster is displayed on the plot.

This Figure shows that at each trip completion, we are able to associate a trajectory, to the group of trajectories it ressembles. We have used these associations to predict the final destination of the trajectory, but many others features can be used for different objectives. Moreover we have seen that our method gives similar results in trajectory classification, Section 2.1.5.2, and in prediction of final destination, Section 2.1.5.3, for both studied datasets of trajectories in San Francisco and Porto. Taking into account the differences between the structure of the road network of theses two cities proves that our method can be adapted to different datasets, without requiring prior study of the dataset. However, the effect of auxiliary variables is different from one dataset to one another. These results show that the behavior of the drivers differs from one city to another. It could help traffic managers to better understand the traffic flow of a city.

2.1.7 Conclusion

In this paper, we proposed a data-driven method to predict the final destination of vehicle trips using a statistic learning procedure. Vehicle trajectories differ from other trajectories in that they are constrained to a road network, which differs from one place to another, and directly influences the behaviour of the users. The learning step of our method follows a two-step procedure which enables to capture the behaviour of the user. It first models the main paths taken by the users by clustering their complete trajectories. Then, it models main traffic flow patterns within each trajectory's cluster by a mixture of 2d-Gaussian distributions. This yields a data driven grid of locations which describes the all space. This model is finally used to predict the final destination of vehicle trips, by assigning the trajectory to the path to whom it belongs the most likely and extracting information from trajectories who follow this path. This prediction is based on the initial location of the trajectory. Since we model the whole path, the prediction can be accomplished at any time during trajectory completion. Such method is applied on two different datasets: trajectories of taxi trip moving on two different road networks from San-Francisco, United-State and from Porto, Portugal and proves that such predictions based on the structures of the paths, compete with methods very complex and not easily tractable such as deep learning methods. Hence we propose a new description of road traffic that can be used for other research. For example, we can use different information from trajectories inside the clusters to short term prediction, estimate arrival time, or detect abnormal behaviour. Our model provide a better understanding of behaviours of the cars drivers by pointing out the main paths. Hence it can help organise trip distribution of a city.

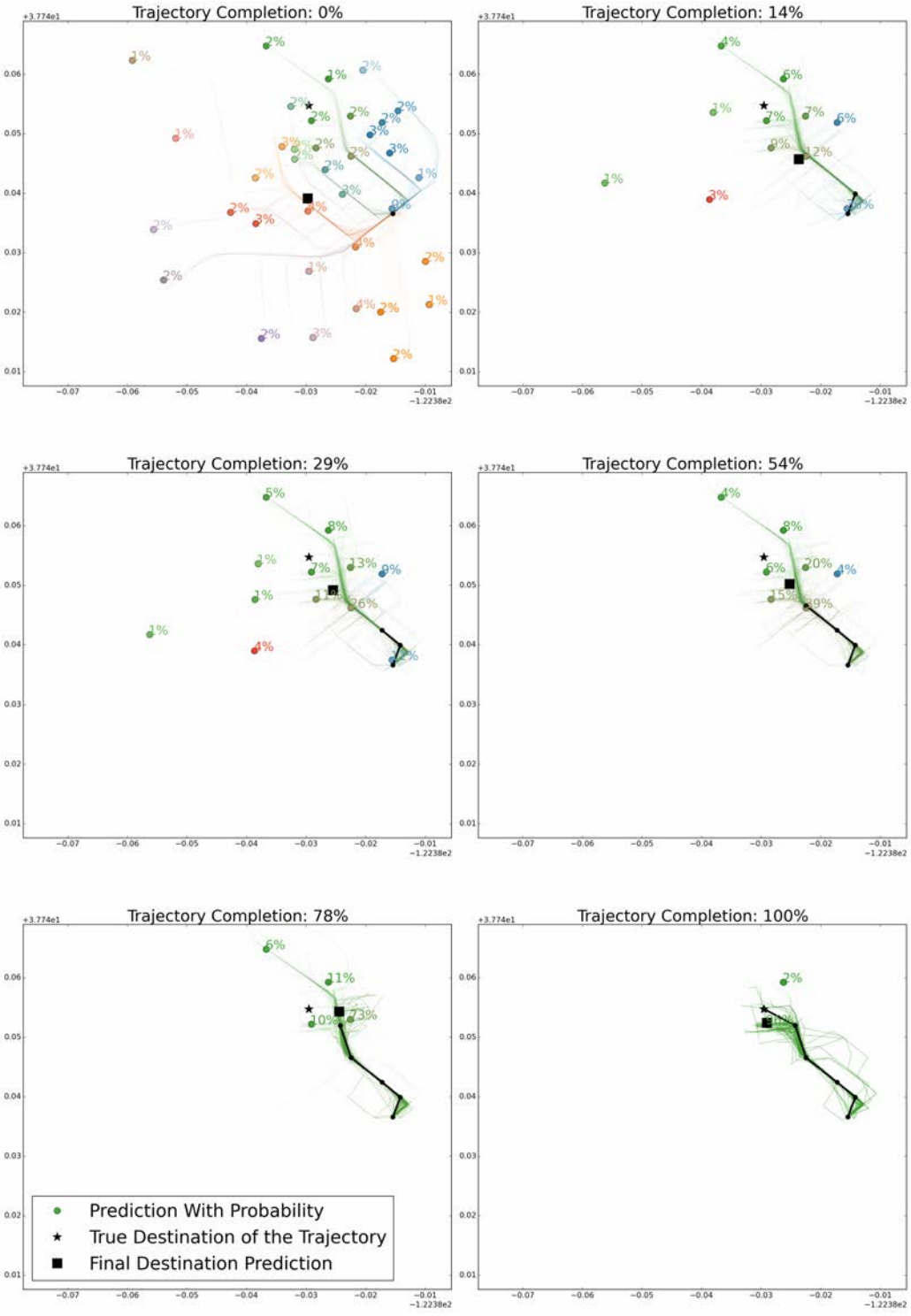


Figure 2.12 – Example of final destination prediction for a taxi trip In San Francisco

2.2 More prediction applications from this Model

The main goal of this section is to demonstrate how easily new methods can be defined, based on the model defined in the previous section. The model we defined, enables to structure the data in order to catch the main patter of the movement of the user of the road network. We already established a method, based on this new structure of the data, in order to predict the final destination of a taxi trajectory.

In this section, we present two new examples of how methods can be adapted on the model to solve two new problematics: predict the arrival time of a new trip, and the next location of a trip. The methods presented in this Section can still be improved. But they demonstrated how easily the model enables to define them.

2.2.1 New structure of the data

2.2.1.1 Clustering of locations

The notations are the same that the one used in the previous section, define in Table 2.1.

In Section 1.2, we defined a method to cluster a set of trajectories, producing K cluster of trajectories where the m^{th} cluster is defined as $\mathcal{T}^m = \{T^i | l^i = m\}$. We denoted l^i as the label of the i^{th} trajectory, *i.e.* $l^i = m$ implies that trajectory $T^i \in \mathcal{T}^m$.

From these clusters, we defined $\mathcal{P}^m = \{p_j^i | l^i = m\}$, $\forall m \in [1 \dots K]$ which is the set of all the points composing the trajectories within cluster $\mathcal{T}^m$. On section 2.1.4, we modelled each set of points $\mathcal{P}^m$ with a mixture of Gaussian. Hence, every point, $p \in \mathcal{P}^m$, is assumed to be generated from a weighted sum of k^m Gaussian densities $\phi_k^m(p)$, as described Definition (32):

$$\Phi^m(p) = \Phi(p|\Theta^m) = \sum_{k=1}^{k^m} \omega_k^m \cdot \phi_k^m(p).$$

One can think of mixture models as a clustering method, which from the set of points $\mathcal{P}^m$, produces a partition of the set in k^m set of points: $\mathcal{C}(\mathcal{P}^m) = \{\mathcal{P}_1^m, \dots, \mathcal{P}_{k^m}^m\}$ where each cluster of points, $\mathcal{P}_n^m$ is assumed to be generated by one of the Gaussian density, ϕ_n^m, composing the mixture.

Given the Gaussian Mixture $\Phi^m(p)$, composed of k^m components, the clustering procedure assigns a point p_j^i to the n^{th} component yielding the highest posterior probability, according to its density function, among all the density functions composing the Gaussian mixture :

$$\hat{n} = \max_{k \in [1 \dots k^m]} \phi_k^m(p_j^i) \tag{2.9}$$

We denote then $l_j^i = (m, n)$ as the label of the point p_j^i, where m denotes the cluster in which the trajectory T^j belongs to , and n the cluster of points at which the point p_j^i

has been assigned according to Equation (2.9).

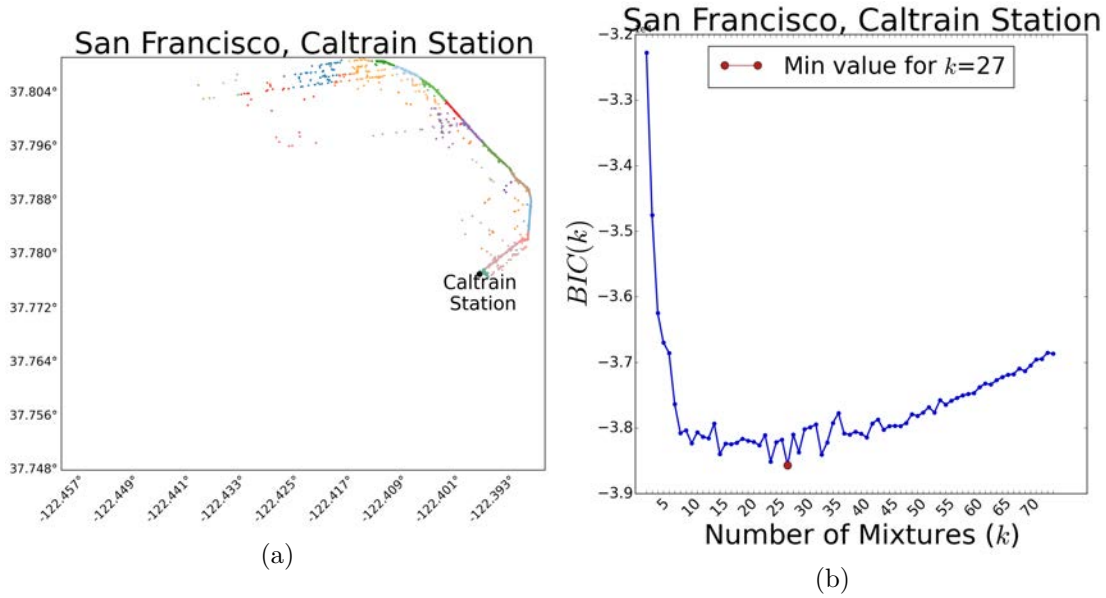
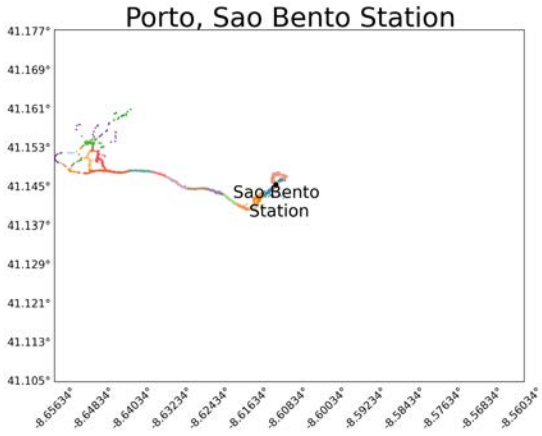


Figure 2.13 – (a) Locations from a cluster of Trajectory in San Francisco. (b) Evolution of the BIC criterion

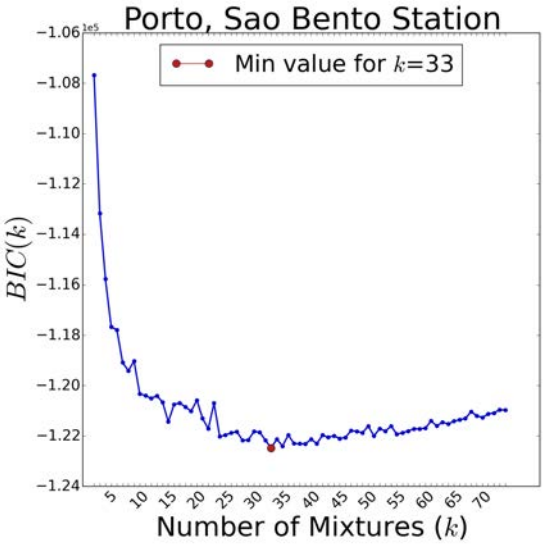
In Figures 2.13a and 2.14a we can observe the partition obtained on two different clusters of trajectories. The first one is a cluster of trajectories from the partition of the Caltrain dataset in 25 clusters. This cluster contains 156 trajectories. The second one is a cluster of trajectories from the partition of the Porto dataset in 45 clusters. This cluster contains 62 trajectories. The locations that composed the trajectories are then partition in clusters according to the method described above. In Figures 2.13b and 2.14b we can see the behaviour of the BIC criterion according to the number of cluster from 0 to 75 clusters. The minimum value is obtained for 27 clusters for the trajectory cluster in San Francisco and 33 clusters in Porto. Until the end of Section 2.2, we will use these two clusters and the trajectories which belong to it to illustrate some of our methods and results.

2.2.1.2 A new trajectory representation

This clustering of location method, desribed above, ebables to describe the all space with a set of *states*, where a state is a cluster of location. Once that all its points have been assigned to a state, with the Equation (2.9), a trajectory can be seen as a succession of *states*, rather than a succession of locations in $\mathbb{R}^2$. This enables us to define the *state trajectory*.



(a)



(b)

Figure 2.14 – (a) Locations from a cluster of Trajectory in Porto. (b) Evolution of the BIC criterion

Definition 40. *A state trajectory* TS^i *is defined as*
$TS^i : [(l_1^i, t_1^i), \ldots, (l_{n^i}^i, t_{n^i}^i)]$,
where $l_j^i = (m, n)$ *s.t.* $\mathcal{T}^m$ *is the cluster of trajectories at which* T^i *has been assigned and* $\mathcal{P}_n^m$ *the cluster of point at which* p_j^i *has been assigned.* n^i *is the length of the trajectory* T^i.

In Figures 2.15 and 2.16, we can see the state representation of the trajectories obtained of the trajectory from the cluster display in Figures 2.13a and 2.14a. The trajectory are displayed according to the hour at which they happened. Every horizontal lines represent a trajectory. These lines are composed of different colors. Each of these colors represent a unique state.

These Figures enable to understand the benefits of the state representation of the trajectory. We can see the time spend within the different state for different trajectories. It gives new features to compare each trajectory to one another. For example, we can extract the transit time and the time spend within each state or the main transition from one state to another. We can also detect main pattern by studying the of state the trajectories went through.

2.2.1.3 Assigning new trajectory to the new structure

In the previous Section, we defined a new representation for trajectory, the *state trajectory*, Definition 40. This new representation is built in a *data-driven* way. Indeed, the

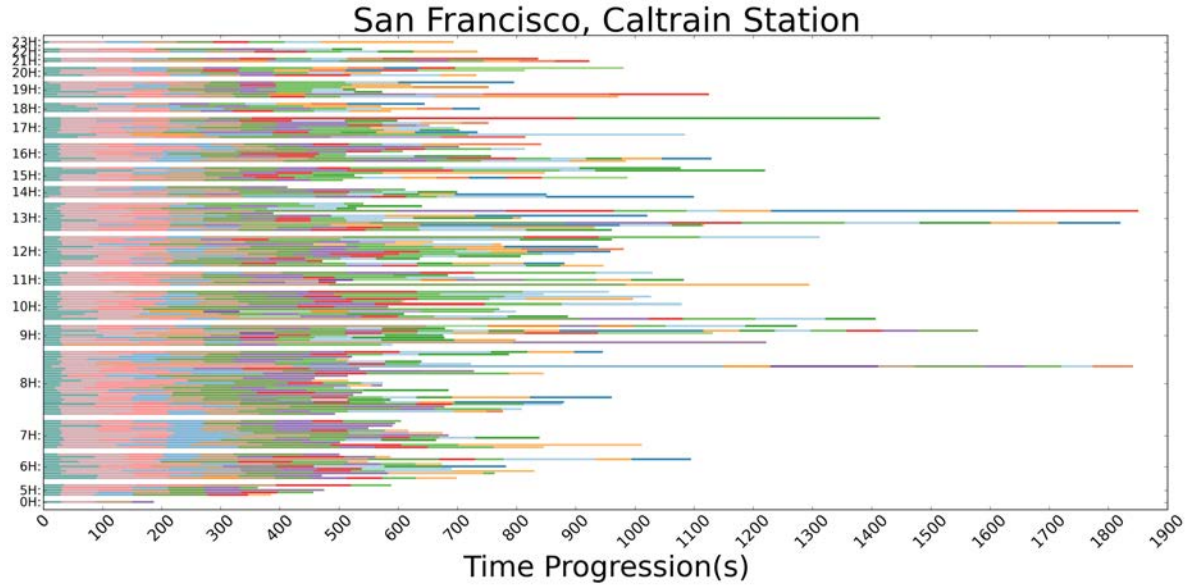


Figure 2.15 – State representation of the Trajectories from the trajectory cluster in Figure 2.13a

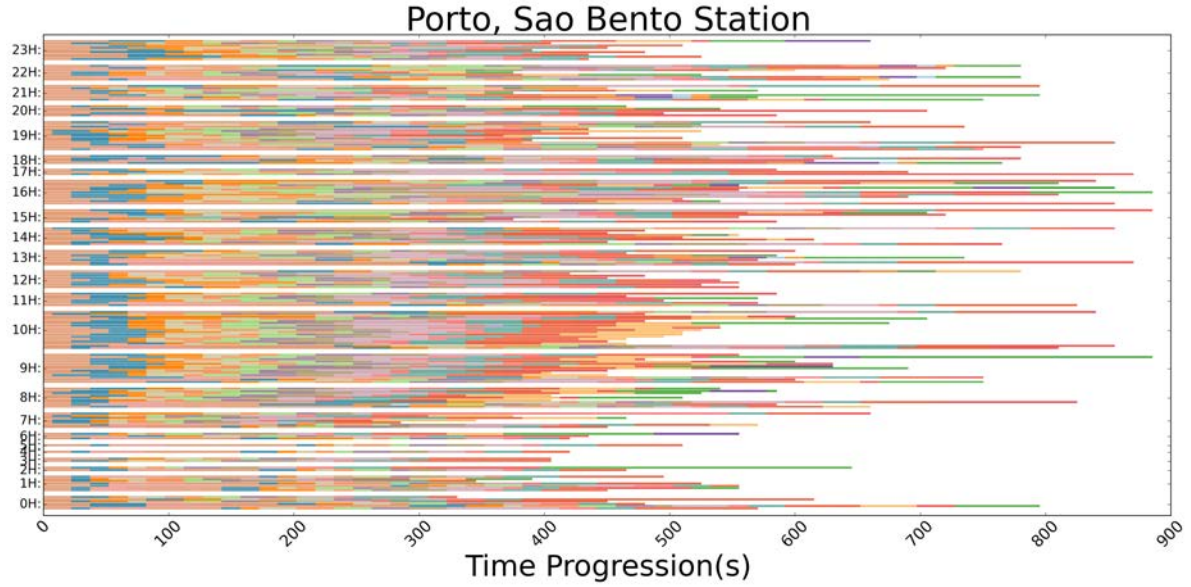


Figure 2.16 – State representation of the Trajectories from the trajectory cluster in Figure 2.14a

states of the new representation are generated with a a two-step clustering procedure (trajectories and locations) based on a given dataset. However, one of the main advantage

of this procedure, is that it enables to quickly convert a new trajectory, T^c, to its state representation, TS^c.

First, the trajectory classification procedure, defined section 2.1.4.1, enables to assigned the trajectory T^c, to the cluster of trajectories it belongs the most likely by computing the *simple score*, $s^m(T^c)$, define Equation (34), for all the cluster of trajectory. The trajectory is assigned to the cluster with the highest affinity score.

$$\widetilde{l}^c = \max_{m \in [1...K]} s^m(T^c) \tag{2.10}$$

Once the trajectory T^c has been assigned to a cluster of trajectory m, *i.e.* $\widetilde{l}^c = m$, we assigned each of its point, p_j^c to a state, (m, n), thanks to the Equation (2.9). We denote then $\widetilde{l}_j^c = (m, n)$ as the label of the point p_j^c.

Now that we have explained how to convert a new trajectory to its state representation, we will present, in the next sections, how this model can be used for two different objectives: arrival time prediction and next location prediction.

2.2.2 Arrival time prediction

In this section, we tackle the problem of predicting the arrival time of a taxi trip knowing the departure time which will give the travel time. For that, we remind first, some definition that will be used in this Section.

Based on the definition of a trajectory, Definition 29, let p_j^i be the j^{th} location of a Trajectory T^i, t_j^i the time at which p_j^i has been observed and $t_{n^j}^i$, the *arrival time* of the trajectory T_j^i. We define l_j^i the label of the point p_j^i, and $\widetilde{l}_j^c$ the predicted label of a point p_j^c. Finally we introduce the definition of the *number of occurrence*, occ_j^i, of a trajectory within a state at a given location, as the number of consecutive appearance of the trajectory within the current state:

Definition 41. *The number of occurrence, occ_j^i, of a trajectory, T^i within a state at the j^{th} observation is defined as:*

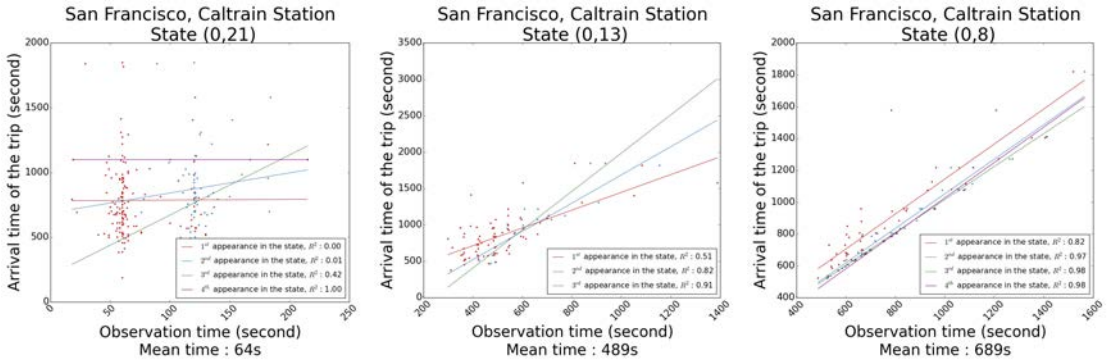
$$occ_j^i = \begin{cases} 1, & \text{if } l_{j-1}^i \neq l_j^i \\ \min\{k \mid l_{j-(k-1)}^i = l_j^i,\ l_{j-k}^i \neq l_j^i\} & \text{otherwise} \end{cases}$$

2.2.2.1 Data exploration

On Figures 2.17 and 2.18, we can observe on each subplot, the arrival time of taxi trips, according to the time at which the taxi have been observed within a given state for respectively the Caltrain Dataset and the Sao Bento Dataset. The colors of the different points in the plot represent the *number of occurrence* of a taxi within the state according to definition 41. For each dataset, three different states from the same cluster of trajectories

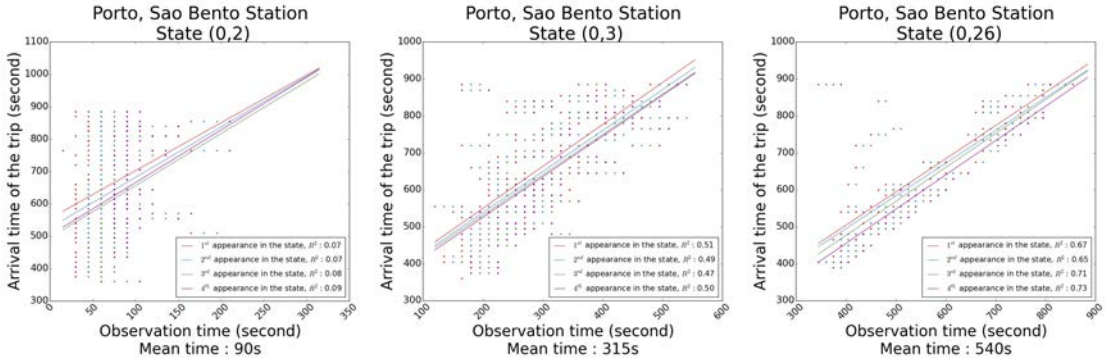
have been chosen at different mean time of trajectory completion. In Figure2.17, the mean time of observation within the state are 64, 489, 699 seconds from the left to the right, while the arrival time for the trajectories within this cluster goes from 186 to 1850 seconds for a mean time of 785 seconds. In Figure2.18 the mean time of observation within the state are 90, 315, 540 seconds from the left to the , while the arrival time for the trajectories within this cluster goes from 360 to 885 seconds for a mean time of 623 seconds.

For each cluster of Locations, and for each number of occurrence in the state, we fit a linear model computed with *Ordinary Least Square* between the time in the state and the arrival time in order to get an overview of the linear relation between the time at which the taxis have been observed within a state and the arrival time.



(a) (b) (c)

Figure 2.17 – Arrival time of the trip according to the transit time in the cluster for three different cluster.



(a) (b) (c)

Figure 2.18 – Arrival time of the trip according to the transit time in the cluster for three different cluster.

In figure 2.18a, the mean time of the observation within the state is 90 seconds. In this

state, there is clearly no linear relation between the time of the observation within the state and the arrival time. This state is one of the first state the taxi crossed during their trip. The end of their trip, and hence their arrival time, will be strongly influenced by the density of the traffic. In Figure 2.18c, the mean time of the observation within the sate is 540 seconds, and it is one of the last state cross by the taxi. At this moment, the trip of the taxi has been influenced by the traffic. Hence the time at which it is observed within the state gives a good information to predict the arrival time. We can even observed that the relation between the time within the state and the arrival time seems to be linear. We can also observed that the *number of occurence* seems to have an effect on the linear relation. The observation for the Caltrain Dataset, Figure 2.17 are similar. Hence we can assume that it is possible to predict the arrival time of the taxi trips, based on the time they have been observed within the different state and the *number of occurrence parameter*. The more the trip is closed to its end, the more information about the traffic we get.

Based on this observations, we propose, in the next section, to predict the arrival time of the trajectories with a linear model and we test different context variable to check if they can help anticipate the lack of information about traffic at the beginning of the trajectory.

2.2.2.2 Model for arrival time prediction

We propose in this section a method to predict the arrival time of the trip of a taxi, $t^i_{n^j}$, based on variables which changes at each step of the taxi's trajectory. The predicted values is hence the arrival time of a taxi, that we will denote as y^i.

Given the observation made in Section 2.2.2.1, we expected a linear combination between the arrival time of a trajectory and the input features within each state such as the time, t^i_j, at which the trajectory has been observed within the state or the *number of occurrence*, occ^i_j, of the trajectory within the state and context variable such at the hour of the day h^i or the day of the week d^i.

Considering $x^i_j = \{t^i_j, occ^i_j, h^i, d^i\}^\intercal$ as the feature vector, this relation between x^i_j and y^i can be formulated as :

$$y^i = x^{i\intercal}_j \beta + \epsilon \tag{2.11}$$

where ϵ is a random error and $\beta \in \mathbb{R}^4$ is the vector of the regression coefficients. We use *Lasso* regression method to estimate the coefficients. This solution promotes sparsity in the sense that models with few coefficients are preferred to large models. It used a *L1* penalty on the parameters. Hence for every state (m, n), we build a *Lasso* model by solving the following objective function:

$$\min_{\omega} \frac{1}{|N_{m,n}|} ||X_{m,n}\beta_{m,n} - Y_{m,n}||^2_2 + \alpha||\beta_{m,n}||_1 \tag{2.12}$$

where $N_{m,n}$ is the number of observation within the state, $X_{m,n} \in \mathbb{R}^{N_n^m \times 4}$ is the features matrix in which each vector correspond to a features vector of an observation, and $Y_{m,n} \in \mathbb{R}^{N_n^m}$ is the answer vector composed of the arrival time of all the observation. α is the regularization parameter, optimized with the BIC criterion.

For a new trajectory, T^c, whose p_j^c is the last observed location. We assume that we have assigned this location to the state (m, n) according to the method describe Section 2.2.1.3, *i.e.* $\widetilde{l}_i^c = (m, n)$. We can then predict the arrival time of the taxi trip, $\hat{y}_j^c$ with the following formula:

$$\hat{y}_j^c = x_j^{c\intercal}.\beta_{\widetilde{l}_i^c} \tag{2.13}$$

where $x_j^{c\intercal}$ is the features vector, and $\beta_{\widetilde{l}_i^c}$ the estimated coefficients of the model linked to the state $\widetilde{l}_i^c = (m, n)$.

2.2.2.3 Experimental results

To test the performance of our method, we use the same procedure that the one described in Section 2.1.5: a 10-cross validation method to calculate this error by learning on 90% the data, the training set $\mathcal{T}_{train}$, and forecasting the remaining 10%, the test set $\mathcal{T}_{test}$.

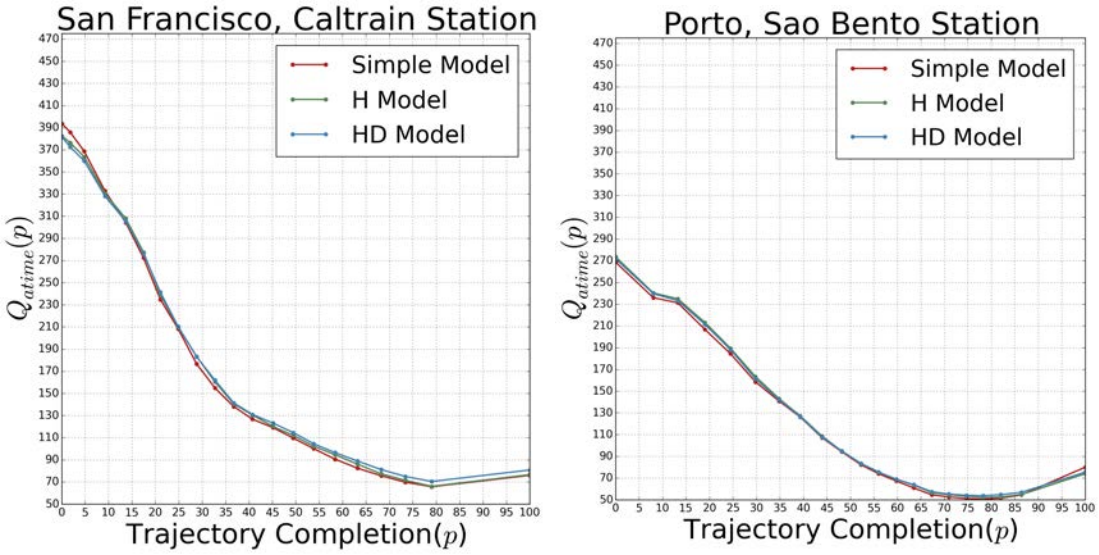
We refer the different models build within the different state described in the previous Section at the *Lasso HD Model*. Because it takes into account both, hours of the day, and day of the week, context variable. In the next Section, we compare this model to slightly different models where only the hours of the day is taking into account for the context features(*Lasso H Model*) , and where no context features are taking into account (*Lasso Model*).

We evaluate our method by looking at the quality criterion, Q_{at} which is the mean value of all the absolute differences between the prediction of the arrival time and the true arrival time of every trajectory T^c in the set $\mathcal{T}$.

Definition 42. *The Quality criterion Q_{at} is defined as:*

$$Q_{at}(p) = \frac{\sum_{T^c \in \mathcal{T}} |at_{pred}(T^c(p)) - t_{n^c}^c|}{\#\mathcal{T}}$$

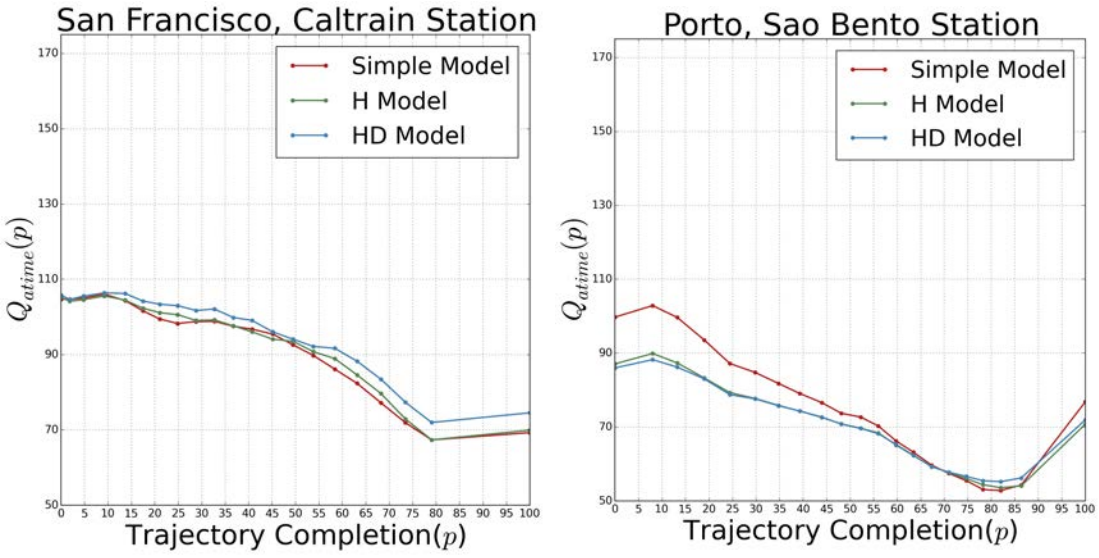
In Figure 2.19 we can observe the evolution of the quality criterion Q_{at} for the Caltrain dataset (a), and the Porto, Sao Bento dataset (b) according to trajectory completion for the prediction method of the arrival time described in the previous Section. We can observe that when the trajectory are at 40% of they completion the mean error of the predicted arrival time is 120 second. The minimum Q_{at} values reached by our prediction method is 70 seconds at 80% of trajectory completion for the *Caltrain Dataset* and 50 seconds also at 80 of trajectory completion for the *Sao Bento Dataset*. However we can see that taking



(a) (b)

Figure 2.19 – Mean Error of Arrival Time Prediction According to Trajectory Completion.

into account the hour of the day and the day of the week have no effect on the quality of the prediction.



(a) (b)

Figure 2.20 – Mean Error of Arrival Time Prediction According to Trajectory Completion when Tc is known.

In Figure 2.20 we can observe the evolution of the quality criterion Q_{at} for the Caltrain dataset (a), and the Porto, Sao Bento dataset (b) for the prediction method described in the previous Section but where the correct cluster of trajectories of each tested trajectories is supposed to be known. For trajectory completion between 0% and 40%, the results are naturally better than results where the correct cluster is not known. After 40% cluster the difference of between the value of the quality criterion Q_{at} when the correct cluster is know or not is less than 20 seconds. We can on see on Figure 2.20a that context variable seems to depreciate the results for the Caltrain Dataset. But the influence is slight and no strong conclusion can be drawn from this results. On the contrary, we can see on figure 2.20b that context variable seems to improve the results, especially at the beginning of the different trips. However, the improvement is never bigger than 15 seconds.

Theses results proves that the new structure we defined for the data enables us to define this method using easily understandable machine learning tools such that *Linear Regression*. They also help the exploration of the data. Moreover, we lack knowledge about traffic management to draw lessons from these explorations, but the results we get show that the model can be used as usefull tool for traffic data exploration.

2.2.3 Following location and state prediction

In this section, we propose a method in order to predict the following location of a vehicle during its trip.

We we will consider for that the state representation of the trajectory of its trip as defined, Definition 40. Hence for a trajectory T^i, its related state trajectory, TS^i is defined as: $TS^i : [(l^i_1, t^i_1), \ldots, (l^i_{n^i}, t^i_{n^i})]$. The states, (m, n) in which the trajectory lies, are obtained according to the clustering of locations procedure defined in Section 2.2.1.1. Hence $l^i_j = (m, n)$ implies that the trajectory T^i belong to the m^{th} cluster of trajectories, $\mathcal{T}^m$, such that $m \in [1, \ldots, K]$ and that the location p^i_j has been assigned to the state $\mathcal{P}^m_n$, such that $n \in [1, \ldots, k^m]$. The set of state, S is Hence the total number of states is defined by K_s and is equal to :

$$K_s = \sum_{m=1}^{K} k^m$$

2.2.3.1 First-order markov chain

To fulfill our objectives of predicting the following location of the trajectory, we want to predict first the following state of its state representation. For that, we consider modeling this trajectory, TS^i, by a first-order discre-time Markov chains having K_s states. This first-order discrete-time Markov chain satisfies the following relationship:

$$P(l^i_{j+1} = (m,n)|l^i_j = (m,n'), l^i_{j-1} = (m,n''), \ldots, l^i_0 = (m,n^0)) \\ = P(l^i_{j+1} = (m,n)|l^i_j = (m,n')), \quad (2.14)$$

The transition probabilities (or conditional probabilities),

$$a^m_{n',n} = P(l^i_{j+1} = (m,n)|l^i_j = (m,n')) \quad (2.15)$$

are called the one-step transition probabilities of the Markov Chain from the state (m,n) to the state (m,n'). These probabilities are written as $a^m_{n',n}$. We note that these probabilities satisfy the following two properties:

$$0 \leq a^m_{n',n} \leq 1, \ \forall (m,n),(m,n') \in S \text{ and } \sum_{n=1}^{k^m} = 1$$

We also consider that this Markov Chain, is time-homogeneous implying that it follow this property:

$$a^m_{n',n} = P(l^i_{j+1} = (m,n)|l^i_j = (m,n')) = P(l^i_j = (m,n)|l^i_{j-1} = (m,n')), \quad (2.16)$$

for all $j \in [0, \ldots, n^i]$. The probability of the transition is independent of the index j of the location of the trajectory.

At a first sight, the choice we made to model the state trajectory with a first-order Markov Chain is not intuitive. Indeed, for a vehicle trip, the next state of its trajectory does not depend only of its current location. For example it will unlikely come back to its previous state and more likely continue its way along the same direction. It is then necessary to take into account its previous state to know its direction and where he is going.

However, we suppose that within each cluster of trajectory, the first-order property of the Markov Chain is respected. Indeed, the cluster of trajectories we defined Section 1.2, regroups trajectories according to the path taken by the users. These path share common start location and final area. Hence the direction along these path is known and we can assume that the following state a vehicle will go through along this path depends only of its current state. Accordingly, we consider that the Markov property is respected for the transition between state within a trajectory cluster. We can then model the observation using a Markov Chain.

2.2.3.2 Estimation of the transition probabilities

We present in this section how we estimate the transition probabilities, defined in the previous Section, Equation (2.15). We consider the set, $\mathcal{TS}$ composing of N state trajectories, TS^j, where, n^j is the number of state that composed the trajectories. we infer the

transition probabilities empirically by simply counting how many times we saw a transition between two state. Hence the transition probability, $a^m_{n',n}$, from a state (m,n) to a state (m,n') is estimated as follow:

$$a^m_{n',n} = \frac{\#\{(l^i_j, l^i_{j+1}) | l^i_j = (m,n'), l^i_{j+1} = (m,n)\}}{\#\{l^i_j | l^i_j = (m,n')\}}, \tag{2.17}$$

For all $i \in [1,\dots,N]$ and for all $j \in [1,\dots,n^i]$. We note that, according to this estimation, the transition between state is possible only if the state are generated from the same cluster of trajectories. Indeed, within the set,$\mathcal{TS}$, the cluster of trajectory are supposed to be known and a trajectory belongs to one cluster only. Hence, the transition probability between the states (m,n) and (m',n') are equal to zero if $m \neq m'$.

By creating Markov chain within cluster of trajectory, we solve the fact that the transition between the current state to the following state of a vehicle trip depends only on the current state, and does not not depend on the previous state. But we can suppose that the transition probability depends on some context variable such as the *number of occurrence*, occ^i_j, of a trajectory within a state at the j^{th} observation as defined Definition 41, or the hours at which the trip takes place. To take these variables into account, we define two new ways to compute approximate the transition probability taking into account theses variables.

We denote as, $a(h)^m_{n',n}$, the *hours estimation*, at the h^{th} hours of the day, of the transition probabilities defined Equation (2.15). It is empirically estimated by counting how many times we saw a transition between two state at hour h for $h \in [0,\dots,23]$:

$$a(h)^m_{n',n} = \frac{\#\{(l^i_j, l^i_{j+1}) | l^i_j = (m,n'), l^i_{j+1} = (m,n), h^i = h\}}{\#\{l^i_j | l^i_j = (m,n'), h^i = h\}}, \tag{2.18}$$

For all $i \in [1,\dots,N]$ and for all $j \in [1,\dots,n^i]$ and where h^i is the hour at which the trajectory TS^i takes place.

In the same way, $a(occ)^m_{n',n}$, the *occurrence estimation*, where the *number of occurrence* is equal to *occ*, of the transition probabilities defined Equation (2.15). It is empirically estimated by counting how many times we saw a transition between two state where the *number of occurence* for the trajectory within the state (m,n') is equal to *occ* for $occ \in \mathbb{N}$:

$$a(occ)^m_{n',n} = \frac{\#\{(l^i_j, l^i_{j+1}) | l^i_j = (m,n'), l^i_{j+1} = (m,n), occ^i_j = occ\}}{\#\{l^i_j | l^i_j = (m,n'), occ^i_j = occ\}} \tag{2.19}$$

For all $i \in [1,\dots,N]$ and for all $j \in [1,\dots,n^i]$ and where occ^i is the *number of occurrence* of the trajectory,TS^i within the state at (m,n') at its j^th observation.

2.2.3.3 Models for following state prediction

In this section we present the method to predict the following state of a given trajectory, TS^i during its completion using the Markov model we define Section 2.2.3.1. This method will enable us to use the Markov chain properties to guess its most likely following state, $\widetilde{l}^i_{j+1}$, assuming that its current state $l^i_j = (m, n')$, is supposed to be known. According to the estimation we made for the transition probability, Section 2.2.3.2, the following state of the trajectory, will be a state generated from the same cluster of trajectories, $\mathcal{T}^m$. Hence , the predicted following state $\widetilde{l}^i_{j+1}$ will be the state $(m, \widetilde{n})$ such that $\widetilde{n}$ is estimated as:

$$\begin{aligned}\widetilde{n} &= \underset{n\in[1\ldots k^m]}{\arg\max}\, P(l^i_{j+1} = (m, n)|TS^i_{0:j}) \\ &= \underset{n\in[1\ldots k^m]}{\arg\max}\, P(l^i_{j+1} = (m, n)|l^i_j = (m, n')) \\ &= \underset{n\in[1\ldots k^m]}{\arg\max}\, a^m_{n',n}\end{aligned} \tag{2.20}$$

The equation (2.20) requires that we get the state representation TS^i of the trajectory T^i for the observation available. Hence the first task, when we want to predict the following state of a trajectory T^i is to convert it to its corresponding state representation TS^i for the observation that are known. At a given time t^i_j, we get the different locations of the trajectory $T^i_{0:j} : [(p^i_1, t^i_1), \ldots, (p^i_j, t^i_j)]$. To convert this trajectory to its state representation we propose two different methods. The first method, (that we denote as **V1** in the Experimental results, Section 2.2.3.5) , attributes the most likely state to a location observation among all existing states. As reminded Section 2.2.1.1, the different states are generated with Gaussian mixtures and each state is assumed to be generated by one of the Gaussian density, ϕ^m_n ,composing the mixtures. We can then assigns a point to the state yielding the highest posterior probability, according to its density function, among all the density functions. Hence for all location p^i_j of T^i, its state l^i_j is determined by the equation:

$$(m, n') = \underset{\substack{m\in[1,\ldots,K]\\ n'\in[1\ldots k^m]}}{\max} \phi^m_k(p^i_j) \tag{2.21}$$

The main drawback of this first method, is that several clusters of locations can overlap in a given area, given that clusters of locations are generated for each set of points that form cluster of trajectory. Hence a location can easily be assigned to its wrong state.

To overcome this problem, we use for that the methodology defined Section 2.2.1.3, that we denote as **V2** in the Experimental results, Section 2.2.3.5. It consists at first assigning the new trajectory, to the cluster of trajectories it belongs the most likely and then assigned each of its point to a the state it belong the most likely among the state generate from the cluster of trajectory the trajectory has been assigned.

Hence, once that the trajectory T^i has been converted to its state representation according to either method V1 or V2, we can predict its following state using the Equation 2.20.

2.2.3.4 Following prediction location

The methods established in the previous Section enable to predict the most likely following state of a vehicle trip during its accomplishment. Once we have predicted the following state, $\widetilde{l}^i_{j+1} = (m, n)$, we want to predict the next location. For that we test two different methods. The first method, predicts the next location $\widetilde{p}^i_{j+1}$, as the mean of the location within the predicted next state:

$$\widetilde{p}^i_{j+1} = \bar{p}^m_n = \frac{1}{\#\mathcal{P}^m_n} \sum_{p \ s.t. \in \mathcal{P}^m_n} p \tag{2.22}$$

The second methods, predicts the next location $\widetilde{p}^i_{j+1}$, by taking into account the real current location, p^i_j and the speed, v^i_j, at which the vehicle is moving from location p^i_{j-1} to location p^i_j. We assume that the vehicle keeps the same speed v^j_i until the next location, which means that it will move along a distance of $d^i_{j+1} = v^j_i \cdot \delta_t$, where δ_t is the time between two locations. We also assume that the vehicle is moving in the direction on the mean location of the next state $\bar{p}^m_n = (\bar{lon}^m_n, \bar{lat}^m_n)$. Taking into account that we know the distance until the next location and the direction taken, we can predict the next location $\widetilde{p}2^i_{j+1} = (\widetilde{lon}2^i_{j+1}, \widetilde{lat}2^i_{j+1})$ using this formula:

$$\begin{aligned}
\widetilde{lat}2^i_{j+1} &= \arcsin(\sin(lat^i_j) \cdot \cos(\delta r^i_{j+1}) + \cos(lat^i_j) \cdot \sin(\delta r^i_{j+1}) \cdot \cos(\theta)) \\
\widetilde{lon}2^i_{j+1} &= lon^i_j \\
&\quad + \arctan 2\Big(\sin(\theta) \cdot \sin(\delta r^i_{j+1}) \cdot \cos(lat^i_j),\ \cos(\delta r^i_{j+1}) - \sin(lat^i_j) \cdot \sin(\widetilde{lat}2^i_{j+1}) \Big)
\end{aligned} \tag{2.23}$$

where $\delta r^i_{j+1} = d^i_{j+1}/R$ is the angular distance, R being the earth's radius. θ is the bearing, $s.t$:

$$\begin{aligned}
\theta = \arctan 2\Big(&\sin(\bar{lon}^m_n - lon^i_j) \cdot \cos(\bar{lat}^m_n), \\
&\cos(lat^i_j) \cdot \sin(\bar{lat}^m_n) - \sin(lat^i_j) \cdot \cos(, \bar{lat}^m_n) \cdot \cos(\bar{lon}^m_n - lon^i_j) \Big)
\end{aligned}$$

2.2.3.5 Experimental results

To test the different methods, we use one more time a 10-cross validation to learn the different transition probability. We test it on the two same datasets: Caltrain and Sao Bento

Dataset. It is useful to recall that, these two datasets differ by their sampling time rate between two locations, δ_t. For the Caltrain dataset, $\delta_t = 60$seconds, while $\delta_t = 15$seconds for the Sao Bento Dataset. Predicting the next state of trajectory within these two datasets mean to predict the next state, 60 seconds forward for the Caltrain dataset and only 15 seconds forward for Sao Bento. These characteristics have two consequences. First, the number of locations that composes a trajectory and thus, the cluster of trajectories, are bigger within the Sao Bento dataset than the Caltrain Dataset, which implies a bigger number of clusters of locations within each cluster of trajectories. Knowing that we have retained a number of 25 clusters of trajectories for the Caltrain Dataset and a number of 45 cluster of trajectories for the Porto Dataset, we obtain a total number of 364 clusters of locations to describe the all space of the Caltrain Dataset and 1311 cluster of Locations for the Porto Dataset. Secondly, the quality of the next location prediction, will of course be simpler for the Porto Dataset, because its time interval between two observations is shorter than with within the Caltrain Dataset.

To evaluate the quality of our prediction, we define two new criteria. The first one, $Q_{NextState}(p)$, measures the ability of our different methods, to guess the correct following state of the trajectory of the vehicle trip during its completion. It is defined as the percentage of well predicted next state $\widetilde{l}^i_{n^i(p)+1}$, at a given rate, p of trajectory completion for all the tested trajectories.

Definition 43. *The quality criterion, $Q_{NextState}$ is defined as*

$$Q_{NextState}(p) = \frac{\#\{T^i(p) | \widetilde{l}^i_{n^i(p)+1} = l^i_{n^i(p)+1}, T^i \in \mathcal{T}\}}{\#\mathcal{T}}.$$

We also defined the quality criterion $Q_{NextLoc}(p)$ which evaluates the capacity of our method to predict the next location of the vehicle trip during its completion. It is defined at the mean of all the distances between the predicted location $\widetilde{p}^i_{n^i(p)+1}$ and the true location $p^i_{n^i(p)+1}$, at a given rate, p of trajectory completion for all the tested trajectories.

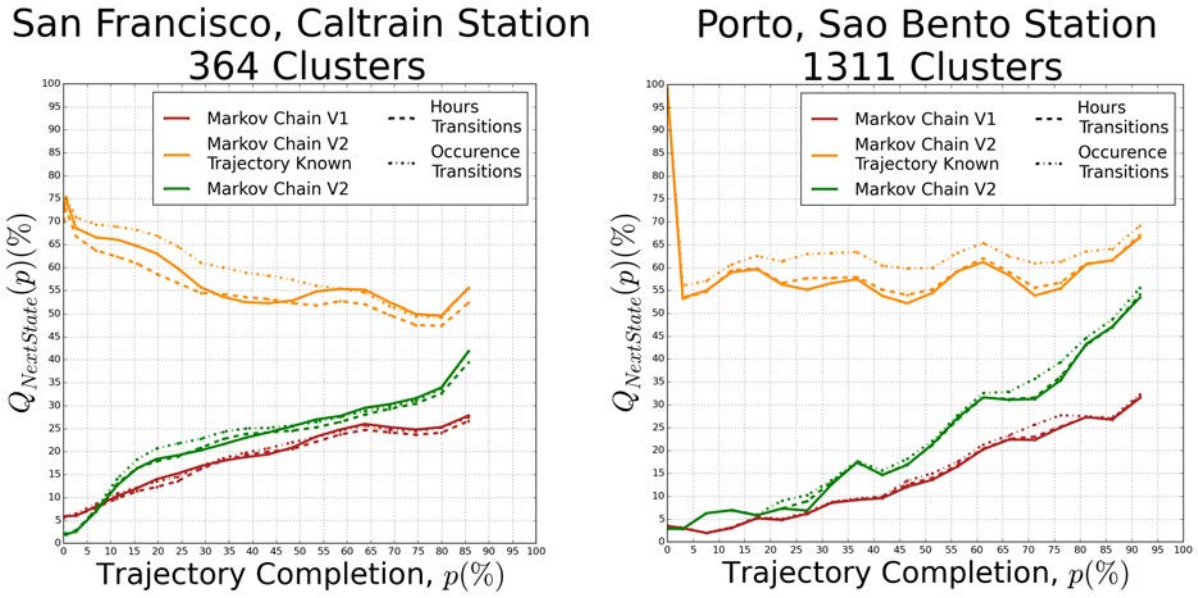
Definition 44. *The quality criterion, $Q_{NextLoc}$, is defined as*

$$Q_{NextLoc}(p) = \frac{\sum_{T^c \in \mathcal{T}} D_H(\widetilde{p}^i_{n^i(p)+1} - p^i_{n^i(p)+1})}{\#\mathcal{T}}$$

These two new quality criteria enable us to test the different methods defined based on Markov model.

On Figures 2.21, 2.22 and 2.23, the results for the two quality criteria are displayed. For each dataset we observe three different methods based on the Equation 2.20. In red, the current state has been estimated among all location's states according to the method **V1** defined Section 2.2.3.3. In green, the trajectory cluster is first estimated, then the current cluster is estimated among the cluster of location within the cluster of trajectory

according to the method **V2** defined Section 2.2.3.3. Finally in yellow, we can observe the results, where the true cluster of the trajectory is assumed to be known. The 'solid' line shows the results obtain with the basic transition probability, Equation 2.15. The effect of the hours, and the time spent in the current state are displayed with respectively a 'dashed' line and a 'dashdot' line.



(a) (b)

Figure 2.21 – Evolution of the quality criterion $Q_{NextState}$ according to the trajectory completion and for the different Markov Chain Methods. (a) Caltrain, (b) Sao Bento.

On Figures 2.21a and 2.21b we can observe the evolution of the quality criterion $Q_{NextState}$ according to the trajectory completion for respectively the Caltrain dataset and the Sao Bento Dataset. For both datasets the quality of the next state prediction is better when the cluster of trajectory is first predicted. This results is intuitive. At a given location, several clusters of location from different clusters of trajectory may overlap. Hence, a location can easily be attributed to the wrong cluster of location. Predicting first the cluster of trajectory enables to best estimated the current location, given that there are fewer clusters of locations which overlap within the same cluster of trajectory. This also confirms the quality of our method of trajectory classification. We can observe that for both methods, the quality criterion, $Q_{NextState}$, increases according to the trajectory. This can be explained by the characteristics of the generated datasets which are composed of trajectories which share a common start area. Hence, the clusters of trajectories they belong appear to be increasingly well-separated when their trips progress. On the contrary, we can see that when the cluster of trajectory associated to the trajectory is known, $Q_{NextState}$ remains constant during the trajectory completion for the Sao

Bento dataset, and are even decreasing for the Caltrain dataset. This results let us thinks that for different dataset, *i.e.*, with different origins and destinations or, on the contrary, with same origins and destinations, the quality criterion $Q_{NextState}$ may remain constant according to trajectory completion.

According to the effects of the different transition probabilities studied, we can conclude that taking into account the hours at which the trip takes place does not improve the quality of the next state prediction. We have seen above, that knowing the hour at which the trip takes places influences the repartition of the trips within trajectory clusters. But the way the vehicles are moving within clusters of trajectories are not influenced by the hours at which the trip takes place. On the contrary, we can see that taking into account the time spent into the cluster, improves the prediction of the next state. This results seems reasonable, indeed, for a location within a state, the probability to stay in the same state at the next time is higher is the time spend in the current state is small. The more the time spend in the cluster increase, the more high the probability to move to another state is. This result is especially visible on the yellow curves, *i.e.*, when the good trajectory is known. The improvement is less meaningful for the two method for which the trajectory is unknown. This can be explained by the fact, that it is harder to be sure of the time spend within a cluster of locations in this case because the estimation of the current state is not constant. Hence if the time spend within a trajectory can improve the quality of the next state prediction, it is not easy to applied it.

We can see that the value of the $Q_{NextState}$ state, is more or less the same for both datasets. Caltrain has 3 times fewer clusters of location that Porto Dataset. But the prediction of the next state is made 60 seconds ahead for Caltrain Dataset, and only 15 seconds ahead for Porto Dataset. This explains why the values are more or less the same for both datasets. Hence it is hard to adress some conclusions based on this quality criterion. To complete this analysis, we look at the results of the second quality criterion, $Q_{NextLoc}$.

On Figures 2.22 and 2.23 we can observe the evolution of the $Q_{NextLoc}$ according to the percentage of trajectory completion for respectively the Caltrain Dataset, and the Sao Bento Dataset. We compare here the results with both the different methods of next state prediction and the two different methods of next location prediction. The color code for the next state prediction is the same that the one described above. The darker colors show the results obtained with the firs method of next location prediction, and the lighters the second method. First of all, we can see that the order of magnitude of $Q_{NextLoc}$ differs from one dataset to another. The values of $Q_{NextLoc}$ are contained between 175 meters and 575 meters for the Caltrain Dataset and between 50 and 225 meters, for the Sao Bento datasets. This difference is due to the different sampling rates for the two datasets.

We compare first the values of the $Q_{NextLoc}$ criteria obtained with the first method of next location prediction, *i.e.* when the predicted location is the mean of the locations that composed the next state. We first can seen that the values of $Q_{NextLoc}$ are not improved

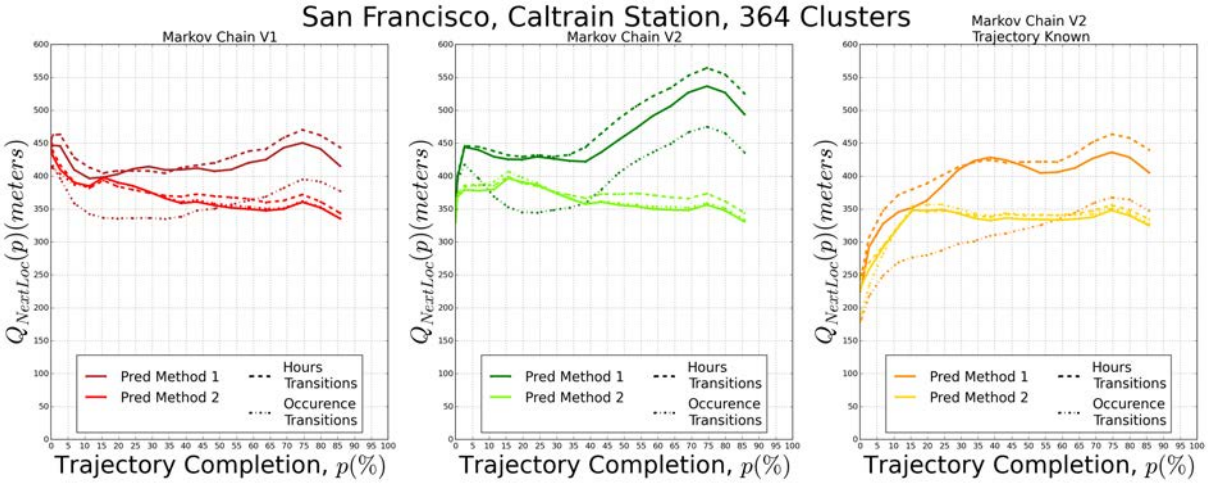


Figure 2.22 – Evolution of the quality criterion $Q_{NextLoc}$ according to the trajectory completion and for the different Markov Chain Methods, Caltrain.

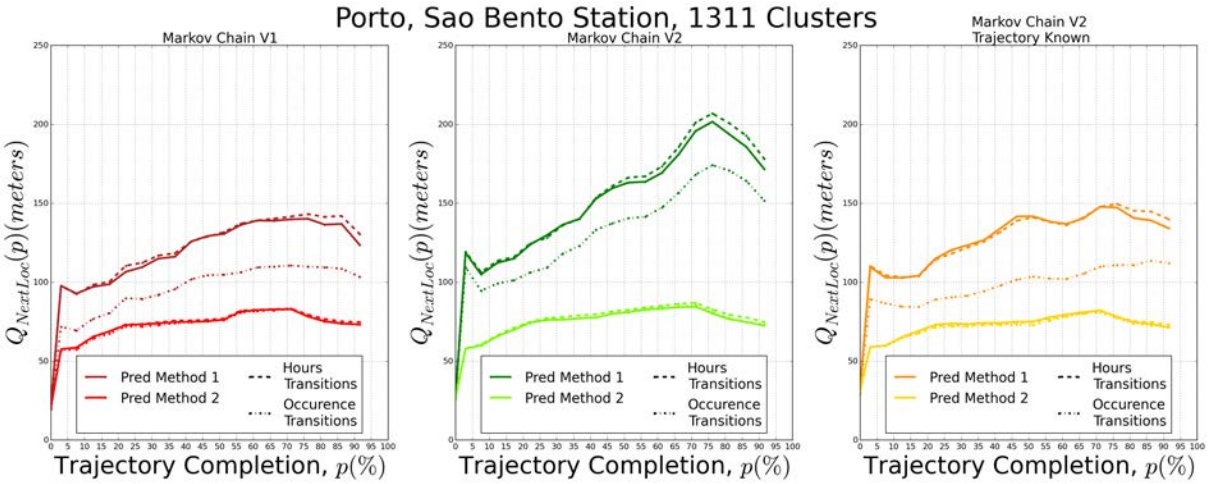


Figure 2.23 – Evolution of the quality criterion $Q_{NextLoc}$ according to the trajectory completion and for the different Markov Chain Methods, Sao Bento.

where the trajectory of cluster if first guessed. This means that for a short term prediction, to be in the right state does not necessarily improved the prediction of the next location. Once again, this result depends of the dataset studied. The different cluster of trajectories all start from the same origin. Hence this result may differs for trajectory of cluster with opposite directions. This result is enhanced by the result obtained here the cluster of trajectory is supposed to be known. If the results are better, they do not significantly improve it. As for the $Q_{NextState}$ criteria, taking into account the hours of the trip, does not improve the result. The $Q_{NextLoc}$ criteria appears to be slightly worst. On the contrary,

taking into account the time spend in the stat enables to improve the value of the $Q_{NextLoc}$ criteria of about 100 meters for the Caltrain dataset, and about 25 meters for the Sao Bento dataset.

The results obtained with the second method of next prediction location, when the mean of the locations of the next state are considered as the direction taken by the vehicle and not directly as the next location considerably improve the quality of the prediction. These results are especially visible on the Sao Bento Dataset. However with this method, both the effect of the hours and the time spend within the current state do not effect the quality of the result.

We can see that the behaviours of the $Q_{NextLoc}$ criterion differs according to the next prediction location method. With the first method, the $Q_{NextLoc}$ criterion increases according to trajectory completion, while it remains constant with the second method. This is due to the fact that the state have various length, according to the density of the locations of the clusters they represent. When we move away from the origin, the density of the locations are sparser. Hence a point in a state far from the origin can be far from the mean of this state. This shows the limit of the first method of next location prediction. On the contrary the second method is not subject to this constraint.

2.3 Conclusion

We proposed in this Chapter, a data-driven method which enables to capture the behaviour of the user. Based on the partition of the trajectories obtained Chapter 1, we model the main traffic flow patterns within each trajectory's cluster by a mixture of 2d-Gaussian distributions. This yields a data driven grid of locations which enables to consider the trajectory as a succession of state instead of a succession of location. This new way to represent the data enables to easily compare the trajectory between them but also to extract various kind of features. We show in this chapter, how this new structure enables to use machine learning and prediction methods in a generic way, which is impossible by considering the data in their original form. In fact the applications defined in this Chapter, are not establish to deal with a given test dataset where the progress of the trajectories is fixed but for any trajectory with any characteristics. Based on well-known mathematical tools such as *Linear Regression* or *Markov Model*, the model enables us to solve various problems linked to road traffic such as: final destination prediction, arrival time prediction or next location prediction. These applications are quite simple, but they demonstrate how easily the model enables the used of well known algorithm.

Obviously, this list of applications is not exhaustive. There are several objectives that we did not achieve due to lack of time but which can be developed by taking advantage of the characteristics of the model. We propose here some application that could be developed based the model.

- *Interaction with the environment.* We did not use, in the application developed in

this chapter, information of the environment. These kind of information are precious for advertiser. Town centers of city such as San Francisco or Porto are full of places of interest such that store, restaurant, or commercial center. We have seen that our model can easily assign a location to a state and thus assign one of this point of interest to a state. Hence by predicting the future state of a vehicle trip and by learning the habits of the individual, we can propose personal interaction with this commercial center. T

- *Anomaly detection.* In fleet management, for example, One of the issue is to detect anomaly in the trajectory of their driver, *i.e.* if a driver takes a wrong direction. We have shown that the model is able to assign to a new trajectory a probability to belong to cluster of trajectory and then to a path. Hence if we knew the right cluster the trajectory is supposed to belong, we can detect is its probability to belong to its cluster is high, or if there is a anomaly.

- *Traffic management.* Finally, by detecting the main traffic flow and congestion area, the model can serve the development of trip distributions and then propose solution for traffic manager and public services.

All this development has been made with the *Python* language. The amount of data we deal with (see Section I.1) leads us to consider new technologies like *Spark* developed in order to handle big volume of data. However this technology is recent and no benchmark are yet been realized in order to compare it to the one commonly used by the statistician community such as *Python* or *R*. Hence before adapting the model and their applications on *Spark* we would rather test it on simpler applications. Accordingly, we have tested this technology on three different use case consisting of the application of a machine learning algorithm on dataset whose size and characteristic match industrial case. The development of these comparison, and their results are the subject of the next Chapter.

2.A Appendix

2.A.1 Hidden markov model for next state and location prediction

In the Section 2.2.3, we presented different methods to predict the next location of a vehicle during its trip based on first-order Markov models. We considered for that the state representation of the trajectory of its trip as defined, Definition 40. Hence for a trajectory T^i, its related state trajectory, TS^i is defined as: $TS^i : [(l^i_1, t^i_1), \ldots, (l^i_{n^i}, t^i_{n^i})]$. The states, (m, n) in which the trajectory lies, are obtained according to the clustering of locations procedure defined in Section 2.2.1.1, where $m \in [1, \ldots, K]$, $n \in [1, \ldots, k^m]$ and $K_s = \sum_{m=1}^{K} k^m$ is the total number of states. The conversion from a trajectory to its

state trajectory, is carried out according to the method described Section 2.2.3.3, where one observation is assigned to a state which is considered as its true state.

The main drawback with this assumption is that we loose the information of the probability for current location to belong to the other cluster. Hence we may want to take advantage of all the probability distribution of the current state in order to predict the following state. To accomplish this task we present new models, based on the properties of the Hidden Markov model.

2.A.1.1 Hidden markov model

A Hidden Markov model is a Markov chain for which the state is only partially observable. Its mathematical fundation has been developed by Baum [Baum and Petrie, 1966]. This model is required when observations are related to the state of the system, but they are typically insufficient to precisely determine the state.

Hence we model a trajectory T^i, with a Hidden Markov Model, where the locations, $p^i_j,\ \forall j \in [0,\dots,n^i]$ are the observations of the model, and where the states of the system, are the state (m,n) described above. The parameters of the model are denoted as $\lambda = \{A, B, \Pi\}$, where A is the state transition matrix, B is the emission probability matrix, and Π is the vector of initial state probabilities. This model follow the first order property, Equation (2.14) as well as the time-homogeneous property, Equation (2.16). Hence the definition of the transition probabilities, $a^m_{n',n}$ are the same that the one described Equation (2.15). The Hidden Markov Model differs from the first-order Markov model described Section 2.2.3.1 by the fact that the state are not directly visible. We defined for that the emission probability, $b^m_n(p^i_j)$, which is the probability for the observation p^i_j to be generated by the state (m,n):

$$b^m_n(p^i_j) = P(p^i_j | l^i_j = (m,n)) \tag{2.24}$$

Hence, given the observation, $[p^i_0,\dots,p^i_j]$ and the parameters, $\lambda = \{A, B, \Pi\}$ of the Hidden Markov Model enables us to first computing the optimal state sequences, $[l^i_0,\dots,l^i_j]$ of the trajectory and to predict its most likely following state. In the following Section, we describe first how the parameters have been estimated, and how we solve these prediction problems

2.A.1.2 Estimation of the the parameters

We present in this section how we estimate the parameters, $\lambda = \{A, B, \Pi\}$, of the Hidden Markov Model. We consider the set, $\mathcal{TS}$ composing of N state trajectories, TS^j, where, n^j is the number of state that composed the trajectories.

The transition probabilities are inferred as described in Section 2.2.3.2. The simple, *hours* and *occurrence estimation* of the transition probability we defined will enable us to test the effect of different context variables.

We recall that the different states of our model are generated with Gaussian mixtures and each state is assumed to be generated by one of the Gaussian density, ϕ_n^m ,composing the mixtures. Hence we use the property of this Gaussian density to estimate the emission probability, $b_n^m(p_j^i)$, for the observation p_j^i to be generated by the state (m, n). Hence we consider that this probability is equal to the probability of the observation given the parameter of the Gaussian density, ϕ_n^m, divided by the sum of the probability of the same observation given the parameter of all the others Gaussian density :

$$b_n^m(p_j^i) = \frac{\phi_n^m(p_j^i)}{\sum_{m' \in [1...K]} \sum_{n' \in [1...k^m]} \phi_{n'}^{m'}(p_j^i)} \tag{2.25}$$

For the estimation of the initial state probability, we have considered that it follows a discrete uniform distribution. This assumption is made in order to make the computation simpler.

2.A.1.3 Models for following state and location prediction

Now that we have estimated the parameters of the Hidden Markov Model, The objective is then to predict the next state $\widetilde{l}_{j+1}^i$, given the observation $T_{0:j}^i$:

$$\widetilde{l}_{j+1}^i = \underset{\substack{m \in [1...K] \\ n \in [1...k^m]}}{\arg\max} P(l_{j+1}^i = (m, n) | T_{0:j}^i) = \underset{\substack{m \in [1...K] \\ n \in [1...k^m]}}{\arg\max} \gamma_n^m(T_{0:j}^i) \tag{2.26}$$

To compute this probability, we used the *forward algorithm* which will enables us to determine the the optimal states sequence, given sequence of observations. This probability is estimated as follow:

for all couple (m, n) , $s.t.\ m \in [1, \ldots, K], n \in [1, \ldots, k^m]$:

$$\begin{aligned}
P(l_{j+1}^i = (m, n)/T_{0:j}^i) &= \sum P(l_{j+1}^i = (m, n), l_j^i = (m, n')/T_{0:j}^i) \\
&= \sum_{n' \in [1...k^m]} \underbrace{P(l_{j+1}^i = (m, n)/l_j^i = (m, n'))}_{\text{Markov Assumption}} P(l_j^i = (m, n')/T_{0:j}^i) \\
&= \sum_{n' \in [1...k^m]} a_{n',n}^m \underbrace{\frac{P(l_j^i = (m, n'), T_{0:j}^i)}{P(T_{0:j}^i)}}_{\text{Bayes}} \\
&\propto \sum_{n' \in [1...k^m]} a_{n',n}^m \underbrace{P(l_j^i = (m, n'), T_{0:j}^i)}_{\text{Forward Likelihood}} \\
&\propto \sum_{n' \in [1...k^m]} a_{n',n}^m \alpha_{n'}^m(T_{0:j}^i) = \widetilde{\gamma}_n^m(T_{0:j}^i)
\end{aligned}$$

The $\propto$ symbol means "proportional to". Indeed, the value of $P(T^i_{0:j})$, is the same for all state (m,n) for which the value of $P(l^i_{j+1}=(m,n)$ is estimated. Hence it is not necessary to compute it.

The "forward likelihood", can be computed as follow:

$$
\begin{aligned}
\alpha^m_{n'}(T^i_{0:j}) &= P(l^i_j=(m,n'),T^i_{0:j}) \\
&= \sum_{n''\in[1\ldots k^m]} P(l^i_j=(m,n'),l^i_{j-1}=(m,n''),T^i_{0:j}) \\
&= \sum_{n''\in[1\ldots k^m]} P(l^i_{j-1}=(m,n''),T^i_{0:j-1})P(l^i_j=(m,n'),p^i_j/l^i_{j-1}=(m,n''),T^i_{0:j-1}) \\
&= \sum_{n''\in[1\ldots k^m]} \alpha^m_{n''}(T^i_{0:j-1})P(l^i_j=(m,n')/l^i_{j-1}=(m,n''),T^i_{0:j-1}) \\
&\quad \times P(p^i_j/l^i_j=(m,n'),l^i_{j-1}=(m,n''),T^i_{0:j-1}) \\
&= \sum_{n''\in[1\ldots k^m]} \alpha^m_{n''}(T^i_{0:j-1}) \underbrace{P(l^i_j=(m,n')/l^i_{j-1}=(m,n''))}_{\text{Markov Assumption}} \underbrace{P(p^i_j/l^i_j=(m,n'))}_{\substack{\text{Assuming that current observation}\\ \text{depends only of current state}}} \\
&= \Big[\sum_{n''\in[1\ldots k^m]} \alpha^m_{n''}(T^i_{0:j-1})a^m_{n'',n'}\Big] b^m_{n'}(p^i_j) \\
&= \widetilde{\gamma}^m_{n'}(T^i_{0:j-1})b^m_{n'}(p^i_j)
\end{aligned}
$$

Hence, with this recurrence relationship, we can compute, $\widetilde{\gamma}^m_n(T^i_{0:j})$ with the algorithm 1

Algorithm 1 Forward prediction

INITIALISATION

Data $:= p^i_0$

$\alpha^m_{n'}(T^i_0) = \pi^m_{n'} b^m_{n'}(p^i_0)\ \forall(m,n')\ , s.t.\ m\in[1,\ldots,K], n'\in[1,\ldots,k^m]$

$\widetilde{\gamma}^m_n(T^i_0) = \sum_{n'\in[1\ldots k^m]} a_{n'n}\alpha^m_{n'}(T^i_0)\ \forall(m,n)\ , s.t.\ m\in[1,\ldots,K], n\in[1,\ldots,k^m]$

$\widetilde{l}^i_1 = \arg\max_{\substack{m\in[1\ldots K]\\ n\in[1\ldots k^m]}} \widetilde{\gamma}^m_n(T^i_0)$

for j **from** 1 **to** n^i **do**

Data $:= T^i_{0:j}$

$\alpha^m_{n'}(T^i_{0:j}) = \widetilde{\gamma}^m_{n'}(T^i_{0:j-1})b^m_{n'}(p^i_j)\forall(m,n')\ , s.t.\ m\in[1,\ldots,K], n'\in[1,\ldots,k^m]$

$\widetilde{\gamma}^m_n(T^i_{0:j}) = \sum_{n'\in[1\ldots k^m]} a^m_{n',n}\alpha^m_{n'}(T^i_{0:j})\forall(m,n')\ , s.t.\ m\in[1,\ldots,K], n\in[1,\ldots,k^m]$

$\widetilde{l}^i_{j+1} = \arg\max_{\substack{m\in[1\ldots K]\\ n\in[1\ldots k^m]}} \widetilde{\gamma}^m_n(T^i_{0:j})$

end for

In the same way as for the markov chain, we propose another method to compute the next state prediction. Instead of computing the forward likelihood for all cluster of trajectory. We first guess the trajectory cluster, $\widetilde{l}^i$, thanks to the classification rules define equation 2.5. After that we use the Algorithm 1 but we do not compute the forward lielihood for all $m \in [1, \dots K]$ but only for $m = l^i$ and we use the conditional emission probability $b_n^m(p_j^i | l^i = m)$ rather $b_n^m(p_j^i)$.

$$l^i = \underset{m \in [1,\dots K]}{\arg\max}\, s^m(T^i_{0:j})$$

$$\widetilde{l}^i_{j+1} = \underset{n \in [1 \dots k^{l^i}]}{\arg\max}\, P(l^i_{j+1} = (l^i, n) | T^i_{0:j}) = \underset{n \in [1 \dots k^{l^i}]}{\arg\max}\, \gamma^{l^i}_n(T^i_{0:j}) \tag{2.27}$$

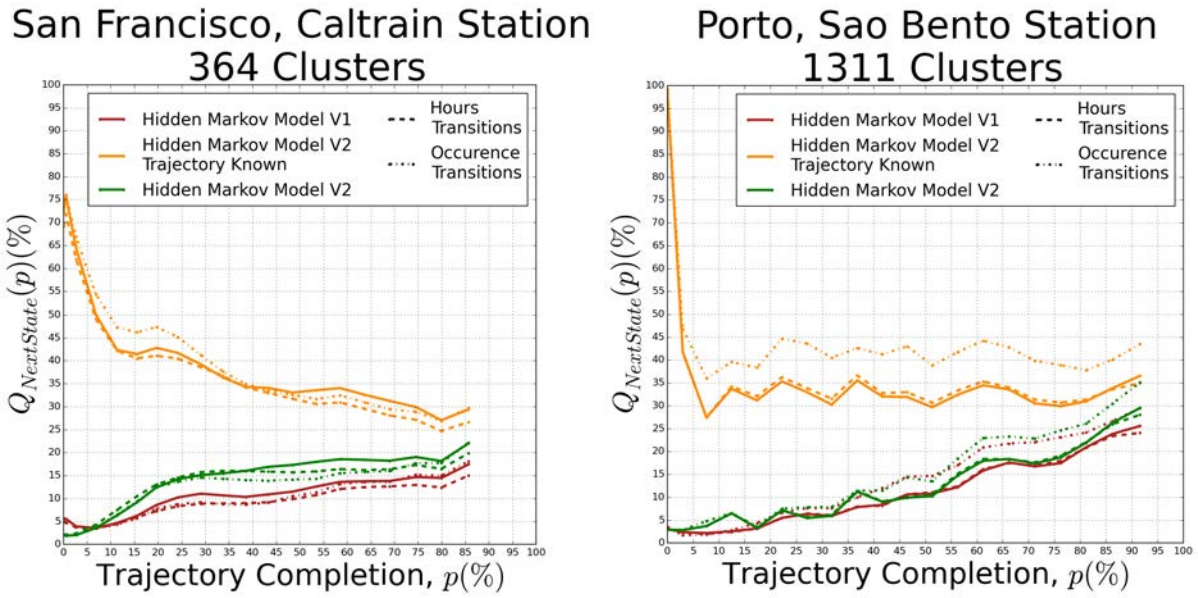
Once the next state of a vehicle has been predicted. We use the different methods established Section 2.2.3.4 to predict the next location of the trajectory.

2.A.1.4 Experimental results

On Figures 2.24, 2.25 and 2.26, the results for the two quality criteria, $Q_{NextState}$ and $Q_{NextLoc}$, are displayed for the different methods established from the Hidden Markov model theory. In red we can see the results of the first method, where all states are considered at each step according to the Equation 2.26. In green, the trajectory cluster is first estimated, then the next state is estimated among the clusters of location within the cluster of trajectory according to the Equation 2.27. Finally in yellow, we can observe the results, where the true cluster of the trajectory is assumed to be known.

On Figures 2.24a and 2.24b we can observe the evolution of the quality criterion $Q_{NextState}$ according to the trajectory completion for respectively the Caltrain dataset and the Sao Bento Dataset. These plots confirm two results found with the Markov Chain model. First, the $Q_{NextState}$ criterion gives better results when we first predict the cluster of trajectory. Secondly, taking into account the hours at which the trips take place does not improve the quality of the prediction. The results are different regarding the effect of the number of occurence within the current state. If the results found on the Sao Bento dataset confirm that taking into account this variable enables to improve the quality, the results are worse for the Caltrain dataset.

On Figures 2.25 and 2.26,we can observe the evolution of the $Q_{NextLoc}$ according to the percentage of trajectory completion for respectively the Caltrain Dataset, and the Sao Bento Dataset. Here again, these results confirm the results found with the Markov Chain model. The benefits of first predicting the cluster of trajectory are less visible looking at the $Q_{NextLoc}$ criteria than the $Q_{NextSate}$ criteria. Taking into account the number of occurrence within the state enables to improve the value of the $Q_{NextLoc}$ criteria of approximately 50 meters for the Sao Bento dataset. On the contrary and as for the $Q_{NextSate}$ criterion, it degrades the values of the $Q_{NextLoc}$ criteria. Finally it confirms that the results obtained



(a) (b)

Figure 2.24 – Evolution of the quality criterion $Q_{NextState}$ according to the trajectory completion and for the different Hidden Markov Model Methods. (a) Caltrain, (b) Sao Bento.

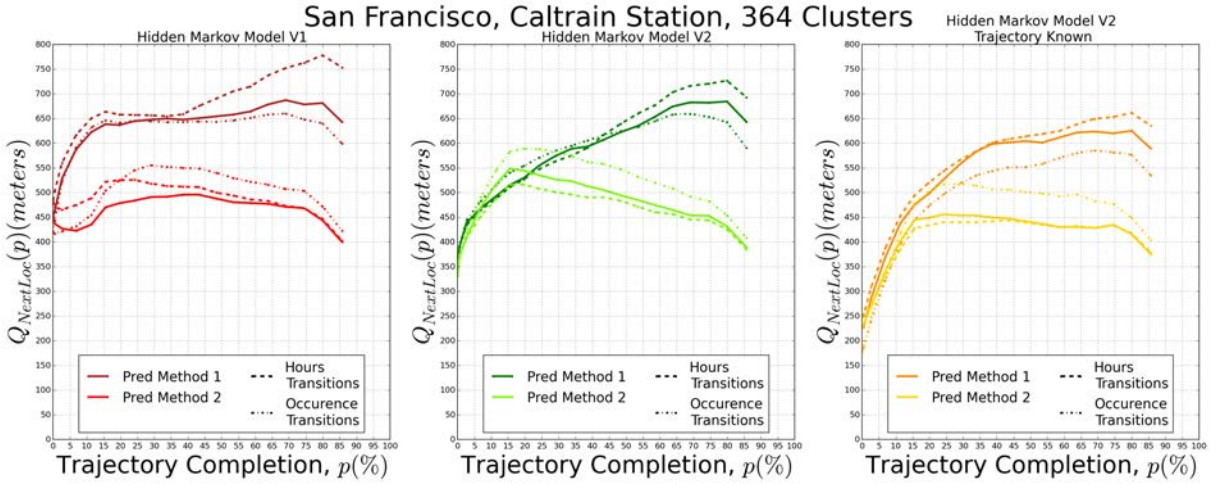


Figure 2.25 – Evolution of the quality criterion $Q_{NextLoc}$ according to the trajectory completion and for the different Hidden Markov Model Methods, Caltrain.

with the second method of next prediction location improve the quality of the prediction. These results are especially visible on the Sao Bento Dataset. However with this method, both the effect of the hours and the time spenr within the current state do not effect the

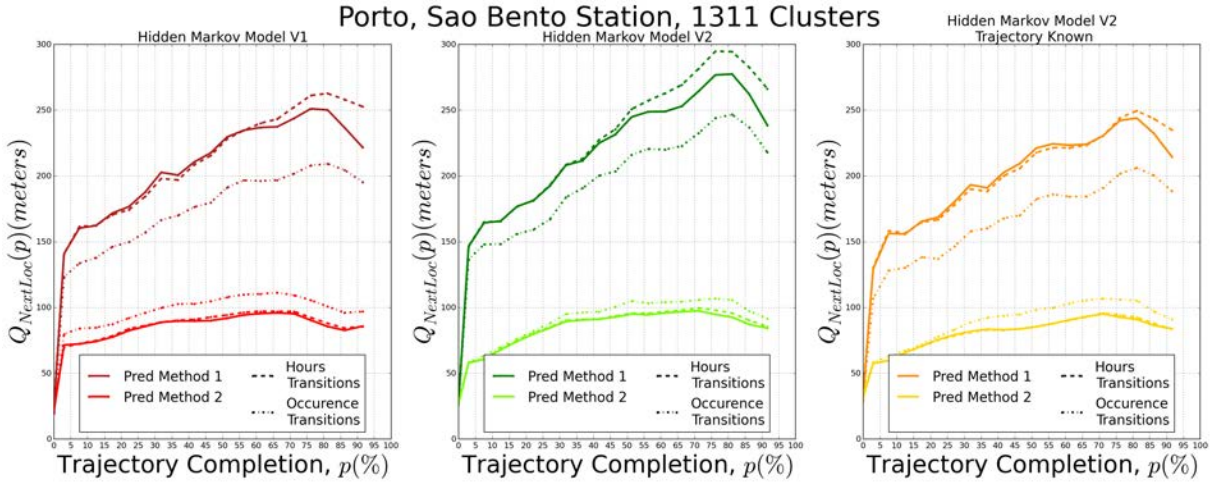


Figure 2.26 – Evolution of the quality criterion $Q_{NextLoc}$ according to the trajectory completion and for the different Hidden Markov Model Methods, Sao Bento.

quality of the result.

But the main conclusion here is that, for both dataset, and for both criteria, the results obtained with the methods established from the Hidden Markov theory are worse that the one found with the Markov Chain theory. Hence we can think that idea of taking into account the probability distribution of the current state among all the existing state is not a good solution and that we better should consider the most likely current state at the true current state of the trajectory. But this conclusion should be qualified by two important idea. First of all, the Hidden Markov Theory is more difficult to be tested. We choose here the "forward algorithm" to estimate the hidden state of the variable. In this section we compute the different transitions probabilities empirically, and the emission probabilities are computed using the distribution of the different GMM models from which the different cluster has been generated. But another solution consists of estimating these parameters according to the *Baum–Welch algorithm*. This algorithm adapts the well known EM algorithm to find the maximum likelihood estimate of the parameters of a hidden Markov model given a set of observation. Secondly, the main idea of the Hidden Markov is to take into account the probability distribution of the current state across all the existing state to predict the next state. However, in our case, the cluster of location has been defined within each trajectory cluster. Hence it is impossible to move from a cluster of location to another if they come from different clusters of trajectories. This reduces significantly the benefits of the Hidden State models. A solution to overcome this limitation should be to generate state which could be common to different clusters of trajectory. This implies to either:

- Apply a clustering method on the set of all the observations that composed all the trajectories of the set $\mathcal{T}$, regardless of the clusters of trajectories the trajectories

belong. The cluster of observation obtained would then be composed of observation from trajectories which belongs to different cluster of trajectory.

- Or we can merge clusters of locations obtained with the different Gaussian Mixtures, according to the parameter of their Gaussian Density.

Hence, the are several possible directions of work to improve the results obtained for the Hidden Markov methods, which required to find a new way to cluster the location. We have performed some work in order to accomplish the cluster of the all set of locations. Some notions are presented in the next Section.

2.A.2 Clustering of locations

In this section, we present some work accomplished in order to perform the clustering of locations. We want that the clustering produces cluster of locations which catch the flow of the vehicle. For example if two locations are physically close to each other, but the trajectories they belong to are moving in opposite directions, we want these locations to belong to different clusters.

2.A.2.1 Distance base clustering

The first way consist in applying some well-known clustering algorithms, among the one described Section 1.1.1. For this purpose we need to define a distance between the locations that composed the trajectories, able to taking into account the direction of this locations. We defined for that do different distances:

- **Euclidean Distance on** $\mathbb{R}^4$. This methods consist to apply the Euclidean distance on Vector $x \in \mathbb{R}^4$. For every location, $p_j^i \in \mathbb{R}^2$ which belongs to a trajectory T^i as defined in Section 29, we associate to this locations a vector x_j^i which is equal to:

$$x_j^i = (lon_j^i, lat_j^i, dlon_j^i, dlat_j^i),$$

  where, lon_j^i and lat_j^i are respectively the longitude and the latitude of the locations p_j^i, and $dlon_j^i$ and $dlat_j^i$ are the differences according to the longitude and the latitude to the next location of the trajectory, *i.e.*, $dlon_j^i = lon_{j+1}^i - lon_j^i$ and $dlat_j^i = lat_{j+1}^i - lat_j^i$

- **Spin Distance**. We define the spin distance as:

$$D_{Spin}(p_j^i, p_{j'}^{i'}) = \left[||p_j^i - p_{j'}^{i'}||_2 - cos(dp_j^i, dp_{j'}^{i'}).\frac{||dp_j^i||_2 + ||dp_{j'}^{i'}||_2}{2} \right]_+$$

  where $p_j^i = (lon_j^i, lat_j^i)$ and $dp_j^i = (dlon_j^i, dlat_j^i)$. This distance enables to taking both into account the physical distance between the location, and the direction of this

locations. Indeed, The greater the angle between the two directions of the locations, the smaller the value of $cos(dp^i_j, dp^{i'}_{j'})$. Hence the physical distance between the locations is penalized if their direction are too differents

We applied different clustering algorithms such that, *K-means*, *Hierarchical Clustering* and *Affinity propagation* on the distance matrices, computed on the set of locations with these two distances. But the results we obtained were not satisfying for different reasons. Indeed for both distance the clusters obtained where not enough separated according to the direction of their locations.

2.A.2.2 DBSCAN algorithm

We encountered some difficulties to define a metric which enables to compare both physical distance between locations and direction of these locations. Hence we decide to look for an algorithm enables to deal with this objectives separately.

We propose for that a new algorithm that is a modification of the *DBSCAN* algorithm established by Ester *and al.* [Ester et al., 1996]. This algorithm enables to group points according to two criteria which are their *density* and their *physical reachability*. We propose to adapt this algorithm by adding a third criteria, which is the *direction reachability*.

To explain our methods, we remind the definitions in [Ester et al., 1996] that enables to define the *DBSCAN algorithm*.

- *eps* : is the maximum distance for which two locations are considered in the same neighbourhood.
- $MinPts$: if the minimum number of points required to form a cluster.

.

Definition 45. *The eps-neighborhood of a point. The Eps-neighborhood of a point p, denoted by $N_{eps}(p)$, is defined as:*

$$N_{eps}(p) = \{q \in D | \overline{p,q} \leq eps\}$$

Definition 46. *directly density-reachable. A point p is directly density-reachable from a point q wrt. eps, MinPts if*

- $p \in N_{eps}(p)$
- $|N_{eps}(p)| \geq MinPts$

Definition 47. *density-reachable. A point p is density- reachable from a point q wrt. eps and MinPts if there is a chain of points $p_1, \ldots, p_n, p_1 = q, p_n = p$ such that p_{i+1} is directly density-reachable from p_i.*

Definition 48. *density-connected. A point p is density-connected to a point q wrt. eps and MinPts if there is a point o such that both, p and q are density-reachable from o wrt. eps and MinPts.*

Definition 49. *Cluster. Let $\mathcal{P}$ be a set of points. A cluster $\mathcal{P}^m$ wrt. eps and MinPts is a non-empty subset of $\mathcal{P}$ satisfying the following conditions:*

- *$\forall p, q$: if $p \in \mathcal{P}^m$ q is density-reachable from p wrt. eps and $MinPts$, then $q \in \mathcal{P}^m$.*
- *$\forall p, q \in \mathcal{P}^m$ p is density-connected to q wrt. eps and $MinPts$).*

The clusters are then formed by propagation. A cluster stop extending where there is no point density-reachable from any points within the cluster. Hence this cluster algorithm is perfectly designed for partitioning where the desired cluster are well physically separated as in the example Figure 1.3.This is obviously not the case for locations. Indeed, we can see on the top left of the Figures 2 and 3 that there is no group of locations correctly well physically delimited. This is logic because according the road network, all the road are density-reachable through the other road.

Hence, the idea of was to add a third criterion in order to define a cluster. Beside that the cluster have to fulfil a *density* threshold and a *Phyisical reachability*, we also want that locations within a cluster have the same *direction*. Hence we define a new *direction criteria*, eps_d, which is the maximum angle between two directions of locations in order to consider that these locations have the same direction.

This allows us to give a new definition of *epss-neighborhood*, where *epss* comprises both *eps* and and eps_d criteria.

Definition 50. *epss-neighborhood. The epss-neighborhood of a point p, $N_{epss}(p)$ wrt $epss = eps, eps_d$ is defined ad :*

$$N_{epss}(p) = \{q \in D | \overline{p,q} \leq eps \ \& \ \widehat{D_p, D_q} \leq eps_d\}$$

The others definitions remind the same except that we use the definitions of the *epss-neighborhood*, Definition 50 instead of the definition of the *eps-neighborhodd*, Definition 45.

This modification of the *DBSCAN* algorithm suffer the same limitations that the original. It performs good clustering as long as the desired cluster are well delimiter in both *physical distance* and *directions*. Hence, this algorithm did not provide result on the dataset *Caltrain* and *Sao Bento* because the direction of the locations are not precise enough. Indeed, we estimate these directions according to the next locations of these points, thus according to the speed of these locations, the estimations of the directions of two locations can be really different event if they are moving in the same direction at a given time. Moreover, the choice of the value of the different criteria is subjective and depends of the density of the road network.

Chapitre 3

Apprentissage sur données massives : Trois cas d'usage avec R, Python et Spark

3.1 Introduction

3.1.1 Objectif

L'objectif de cet article est d'ouvrir une réflexion sur le choix des meilleures mises en œuvre de techniques d'apprentissage face à des données massives. Cet objectif ambitieux se heurte à de nombreuses difficultés : il présuppose une définition de ce que sont des *données massives*, donc les spécificités de l'environnement de travail et le choix de critères de comparaison.

De façon triviale, des données deviennent massives lorsqu'elles excèdent la capacité de la mémoire vive (RAM) de l'ordinateur (quelques giga-octets), puis réellement massives lorsqu'elles excèdent celle du disque dur (quelques tera-octets) et doivent être distribuées sur plusieurs disques voire machines. Cette définition dépend évidemment de l'environnement matériel mais les principaux problèmes méthodologiques et algorithmiques sont soulevés lorsque les données sont effectivement distribuées sur plusieurs machines. Le principal objectif est alors d'éviter des transferts coûteux de données entre ordinateurs ou même entre disques et unités centrales ; analyse et calculs sont déportés sur le lieu de stockage et doivent se limiter à une seule lecture des données.

L'environnement de travail peut-être un poste de travail personnel avec plus ou moins de mémoire, de processeurs, de cartes graphiques (GPU), l'accès à un puissant serveur de calcul intensif ou encore, c'est la caractéristique originale de la prise en compte de données massives, à un *cluster* de calcul réel ou virtuel, privé ou loué. Pouvoir transférer les calculs d'un poste personnel utilisé pour la réalisation d'un prototype, au déploiement, passage à

l'échelle, d'un grand *cluster* à l'aide du même code est un élément essentiel d'efficacité car le coût de développement humain l'emporte largement sur celui de location d'un espace de calcul.

Les évaluations et comparaisons classiques de performances en apprentissage sur des données de faible taille se compliquent encore avec l'explosion combinatoire du nombre d'implémentations des algorithmiques ainsi qu'avec la diversité des technologies de parallélisation et celle des architectures matérielles. Le nombre de paramètres à considérer rend ces comparaisons beaucoup trop complexes ou soumises à des choix bien arbitraires. Face à cette complexité, l'objectif est nécessairement réduit. Loin de pouvoir discuter la réelle optimalité des technologies d'analyse de données massives, on se propose d'aborder les intérêts respectifs de quelques stratégies autour de trois cas d'usage devenus classiques afin d'illustrer, ne serait-ce que la complexité des problèmes et la pertinence de certains choix.

L'objectif est donc réduit à la comparaison des implémentations, dans trois environnements très répandus ou en pleine expansion : R, Python (*Scikit-learn*) et *Spark-MLlib*, de trois méthodes parmi les plus utilisées : les forêts aléatoires ([Breiman, 2001]) en classification supervisée, la factorisation non négative de matrices (NMF) pour apprendre des recommandations et enfin la régression logistique faisant suite au pré-traitement d'un corpus volumineux de textes.

D'autres approches, d'autres technologies, d'autres méthodes, certes importantes et faisant par exemple appel à l'apprentissage profond (*deep learning*) pour l'analyse d'images, à des méthodes d'optimisation stochastique ou de décision séquentielle... nécessiteraient des développements spécifiques conséquents ; elles sont volontairement laissées de côté.

Il s'agit simplement d'illustrer, sur des cas d'usage, le choix qui se pose au statisticien de passer, ou non, à un environnement technologique plus complexe : Python *Scikit-Learn* ([Pedregosa et al., 2011]), *Spark* ([Zaharia et al., 2012]) que celui : R ([R Core Team, 2016b]) qu'il maîtrise depuis de longues années.

Tous les scripts des programmes utilisés en R, Python et Pyspark sont disponibles sur le site `github/wikistat.fr` sous la forme de *notebooks* ou calepins *Jupyter*.

Les auteurs tiennent à remercier l'équipe de la société *Hupi* de Bidart et son directeur M. Moréno qui ont mis à notre disposition la plateforme matérielle (*cluster*) utilisée pour aborder l'environnement *Spark Hadoop* et qui ont également assuré un support efficace pour exploiter au mieux les technologies pour données distribuées.

3.1.2 Nouvelle Science des Données

La fouille de données (*data mining*) des années 90 a largement promu de nouveaux métiers et surtout de nouvelles suites logicielles commerciales intégrant gestion, transformations et analyse statistique des données pour s'adresser à de plus vastes marchés, principalement le marketing. L'objectif était déjà la valorisation (gestion de la relation client) de données acquises par ailleurs et sans planification expérimentale spécifique à une démarche

statistique traditionnelle. Une deuxième étape, en lien avec l'accroissement du volume et la complexité des données, fut l'explosion de leur très grande dimension avec la multitude des omiques (génome, transcriptome, protéome...) produite par les technologies de séquençage. Ce fut l'expansion de la Bioinformatique et, pour le statisticien confronté au fléau de la dimension, la nécessaire recherche de modèles parcimonieux. Une troisième étape est liée au développement d'internet, du commerce en ligne et des réseaux sociaux. Ce sont les principales sources de production et d'analyse de données massives (*big data analytics*). Comme pour la fouille des données des années 90, mais avec une ampleur sans commune mesure, l'histoire se répète avec l'explosion du marché des espaces publicitaires (*advertasing*) en ligne (*Google, Facebook...*) et celles des services (*Amazon Web Service...*) associés.

[Friedman, 1997] soulevait déjà la question : *Is data mining an intellectual discipline ?* Les enjeux actuels font *changer d'échelle* pour s'interroger sur la naissance plus prestigieuse d'une « nouvelle» *Science, des Données*, accompagnée d'un considérable battage médiatique. Celui-ci glorifie les exploits de la ruée vers le nouvel eldorado des investissements et emplois mais stigmatise aussi les risques éthiques, juridiques ou les fiascos annoncés. Cette évolution, tant historique que méthodologique, et son influence sur les programmes académiques est décrite par [Besse and Laurent, 2016b]. Elle n'est pas reprise ici mais nous en retenons le principal élément. Celui-ci s'apparente à l'émergence d'un nouveau paradigme provoquée par les changements d'échelles de volume, variété, vélocité des données qui bousculent les pratiques, tant en Statistique qu'en Apprentissage Machine.

L'estimation d'un modèle prédictif, qu'il soit statistique ou d'apprentissage supervisé, est la recherche d'un compromis optimal entre biais (*erreur d'approximation*) et variance (*erreur d'estimation*) donc d'un modèle parcimonieux évitant le sur-ajustement. À cette question centrale en Apprentissage, vient s'ajouter un nouveau problème d'*Optimisation* sous la forme d'un troisième terme d'erreur. Les ressources (mémoire, temps) sont contraintes ; comment minimiser le terme d'erreur d'optimisation dû, soit à un sous-échantillonnage pour satisfaire aux contraintes de mémoire, soit à la limitation des calculs ou méthodes utilisables sur une base d'apprentissage très volumineuse et distribuée ? Une fonction objectif globale à minimiser pourrait être définie à la condition d'évaluer le coût induit par des erreurs de prévision dues à l'échantillonnage, à mettre en balance avec le coût de l'estimation avec données massives sur des serveurs les supportant. Le problème général étant bien trop complexe, nous nous restreignons à des comparaisons plus qualitatives.

3.1.3 Contenu

La section 2 introduit l'environnement technologique de référence *Spark* ([Zaharia et al., 2012]) associé à la gestion de données massive *Hadoop distributed file system*. L'importance de la notion de *résilience* des bases de données est illustrée par l'exemple de l'algorithme élémentaire de [Forgy, 1965] modifié pour satisfaire aux contraintes des fonctionnalités

MapReduce d'*Hadoop*. Sont introduites également les librairies *MLlib, SparkML* développées pour exécuter les méthodes de modélisation et apprentissage dans ce contexte.

Les sections suivantes abordent trois cas d'usage sur des données publiques ou rendues publiques : reconnaissance de caractères (MNIST), recommandation de films (MovieLens), catégorisation de produits (Cdiscount) avec l'objectif de comparer les performances (temps, précision) des environnements R, Python et Spark sur ces exemples de taille raisonnable.

Les comparaisons ont été opérées sur des machines de faible taille / coût : Lenovo X240 (Windows 7) avec processeur 4 cœurs cadencé à 1,7 GHz et 8Go de RAM, MacBook Pro (OS X Yosemite) avec processeur 4 cœurs cadencé à 2,2 GHz et 16Go de RAM, cluster (Linux, Spark) avec au plus 1 nœud maître et 8 nœuds excécuteurs de 7Go de RAM. Ces configurations ne sont pas très réalistes face à des données massives. Néanmoins les résultats obtenus, sont suffisamment explicites pour illustrer les contraintes du compromis taille des données *vs.* précision pour une taille de mémoire fixée.

La dernière section conclut sur l'opportunité des choix en présence.

3.2 Principaux environnements technologiques

Les choix de méthodes offertes par les librairies et les performances d'une analyse dépendent directement de l'environnement matériel et logiciel utilisé. La profusion des possibilités rendent ces choix difficiles. Il serait vain de chercher à en faire une synthèse, mais voici quelques éléments de comparaison parmi les environnements les plus pratiqués ou au mois les plus médiatisés et accessibles car sous la licence de l'*Apache Software Fondation*.

3.2.1 Nouveau modèle économique

Rappelons tout d'abord que le déluge des données, conséquence de la *datafication* de notre quotidien, entraîne un profond changement des modèles économiques en lien avec l'analyse de ces données. Changement qui impacte directement les développements et donc disponibilités des environnements accessibles aux entreprises et aux établissements académiques. Les équipements trop chers sont loués, les langages et logiciels sont libres mais les concepts et méthodes complexes à assimiler et mettre en œuvre sont des services (formations, plateformes) monnayables. Plus généralement, tout un ensemble de ces services et une nomenclature associée se développent avec l'industrialisation, la commercialisation du *cloud computing* : *software as a service* (SaaS), *infrastructure as a service* (IaaS), *platform as a service* (PaaS), *desktop as a service* (DaaS)...

À titre d'illustration, citons seulement quelques entreprises surfant sur la vague des nouvelles technologies : *Enthought* (Canopy) et *Continuum analytics* (Anaconda) proposent des distributions libres de Python et, c'est important, faciles à installer ainsi que des environnements plus élaborés payants et de la formation associée. *Horthonworks* et

Cloudera diffusent des environnements de *Hadoop* incluant *Spark*. Les créateurs ([Zaharia et al., 2012]) de *Spark* ont fondé *databricks* : *Data science made easy, from ingest to production*, pour principalement vendre de la formation et une certification. Trevor Hastie et Ron Tibshirani conseillent *Oxdata* qui développe (*H20*) avec notamment une forme d'interface entre R et le système *Hadoop*.

Dans le même mouvement ou cherchant à ne pas perdre trop de terrain par rapport à *Amazon Web Services*, les grands éditeurs ou constructeurs intègrent, avec plus ou moins de succès, une offre de service à leur modèle économique d'origine : *Google Cloud platform, IBM Analytics, Microsoft Azure, SAS Advanced Analytics...*

3.2.2 *Hadoop, MapReduce, Spark*

Hadoop (*distributed file system*) est devenu la technologie systématiquement associée à la notion de données considérées comme massives car distribuées. Largement développé par *Google* avant de devenir un projet de la fondation *Apache*, cette technologie répond à des besoins spécifiques de centres de données : stocker des volumétries considérables en empilant des milliers de cartes d'ordinateurs et disques de faible coût, plutôt que de mettre en œuvre un supercalculateur, tout en préservant la fiabilité par une forte tolérance aux pannes. Les données sont dupliquées et, en cas de défaillance, le traitement est poursuivi, une carte remplacée, les données automatiquement reconstruites, sans avoir à arrêter le système.

Ce type d'architecture génère en revanche un coût algorithmique. Les nœuds de ces serveurs ne peuvent communiquer que par couples (clef, valeur) et les différentes étapes d'un traitement doivent pouvoir être décomposées en étapes fonctionnelles élémentaires comme celles dites de *MapReduce*. Pour des opérations simples, par exemple de dénombrement de mots, d'adresses URL, des statistiques élémentaires, cette architecture s'avère efficace. Une étape *Map* réalise, en parallèle, les dénombrements à chaque nœud ou exécuteur (*workers*) d'un *cluster* d'ordinateurs, le résultat est un ensemble de couples : une clef (le mot, l'URL à dénombrer) associée à un résultat partiel. Les clefs identiques sont regroupées dans une étape intermédiaire de tri (*shuffle*) au sein d'une même étape *Reduce* qui fournit pour chaque clef le résultat final.

Dans cette architecture, les algorithmes sont dits *échelonnables* de l'anglais *scalable* si le temps d'exécution décroît linéairement avec le nombre d'exécuteurs dédiés au calcul. C'est immédiat pour des dénombrements, des calculs de moyennes, ce n'est pas nécessairement le cas pour des algorithmes itératifs complexes. La méthode des k plus proches voisins n'est pas échelonnable au contraire des algorithmes de classification non-supervisée par réallocation dynamique (*e.g.* Forgy, k-means) qui opèrent par itérations d'étapes *MapReduce*.

L'exemple de l'algorithme de [Forgy, 1965] est très révélateur.

- **Initialisation** de l'algorithme par définition d'une fonction de distance

et désignation aléatoire de k centres.

- **Jusqu'à** convergence :
  - L'étape **Map** calcule, en parallèle, les distances de chaque observation aux k centres courants. Chaque observation (vecteur de valeurs) est affectée au centre (clef) le plus proche. Les couples : (clef ou numéro de centre, vecteur des valeurs) sont communiqués à l'étape *Reduce*.
  - Une étape intermédiaire implicite **Shuffle** adresse les couples de même clef à la même étape suivante.
  - **Pour** chaque clef désignant un groupe, l'étape **Reduce** calcule les nouveaux barycentres, moyennes des valeurs des variables des individus partageant la même classe c'est-à-dire la même valeur de clef.

Principal problème de cette implémentation, le temps d'exécution économisé par la parallélisation des calculs est fortement pénalisé par la nécessité d'écrire et relire toutes les données entre deux itérations.

3.2.3 *Spark*

C'est la principale motivation du développement de la technologie *Spark* ([Zaharia et al., 2012]) à l'université de Berkeley. Cette couche logicielle au-dessus de systèmes de gestion de fichiers comme *Hadoop* introduit la notion de base de données *résiliente* (*resilient distributed dataset* ou RDD) dont chaque partition reste, si nécessaire, présente en mémoire entre deux itérations pour éviter réécriture et relecture. Cela répond bien aux principales contraintes : des données massives ne doivent pas être déplacées et un résultat doit être obtenu par une seule opération de lecture.

Techniquement, ces RDDs sont manipulées par des commandes en langage *Java* ou *Scala* mais il existe des API (*application programming interface*) acceptant des commandes en Python (*PySpark*) et en R. *Spark* intègre beaucoup de fonctionnalités réparties en quatre modules : *GRAPHX* pour l'analyse de graphes ou réseaux, *streaming* pour le traitement et l'analyse des flux, *SparkSQL* pour l'interrogation et la gestion de bases de tous types et la librairie *MLlib* pour les principaux algorithmes d'apprentissage. En plein développement, cet environnement comporte (version 1.6) des incohérences. *SparkSQL* génère et gère une nouvelle classe de données *DataFrame* (similaire à R) mais qui n'est pas connue de *MLlib* qui va progressivement être remplacée par *SparkML* dans les versions à venir...

Dans la présentation qui en est faite, *Spark* apparaît donc comme un cadre général (*framework*) permettant de connecter la plupart des technologies développées sous forme de projet *Apache*. Tous types de fichier (JSON, csv, RDDs...) et types de données structurées ou non, tous types de flux de données (objets connectés, tweets, courriels...) peuvent

ainsi être gérés, agrégés par des opérations classiques de sélection, fusion à l'aide de commandes utilisant une syntaxe dérivée de SQL (*structured querry language*) avant d'être utilisés pour des modélisations.

3.2.4 Librairie *MLlib*

Cet article est focalisé sur l'utilisation de quelques méthodes de transformation et modélisation sous *Spark* dont celles de *MLlib* décrites par [Pentreath, 2015]. Cette librairie regroupe les algorithmes d'apprentissage adaptés à des bases de données résilientes et qui supportent donc le *passage à l'échelle* du volume des données. Un programme mis au point sur de petits échantillons et un poste de travail personnel peut en principe s'exécuter en l'état sur un *cluster* de calcul (*e.g. Amazon Web Service*) pour analyser des données massives même si la *scalabilité* n'est pas strictement atteinte.

Pour cette librairie en plein développement et aussi en migration vers *SparkML*, seule la documentation en ligne, malheureusement fort succincte, est à jour concernant la liste des méthodes disponibles et leurs options d'utilisation. En voici un rapide aperçu (version 1.6) :

Statistique de base : Univariée, corrélation, échantillonnage stratifié, tests d'hypothèse, générateurs aléatoires, transformations (standardisation, quantification de textes avec hashage et TF-IDF pour *vectorisation*), sélection ($\chi 2$) de variables (*features*).

Exploration multidimensionnelle Classification non-supervisée (k-means avec version en ligne, modèles de mélanges gaussiens, LDA (*Latent Dirichlet Allocation*), réduction de dimension (SVD et ACP mais en Java ou Scala pas en Python), factorisation non négative de matrice (NMF) par moindres carrés alternés (ALS).

Apprentissage Méthodes linéaires : SVM, régression gaussienne et binomiale ou logistique avec pénalisation L1 ou L2 ; estimation par gradient stochastique, ou L-BFGS ; classifieur bayésien naïf, arbre de décision, forêts aléatoires, boosting (*gradient boosting machine* en Scala).

Les sections qui suivent proposent une étude comparative des environnements R, Python (*Scikit-Learn*) et *Spark MLlib* sur trois cas d'usage de données publiques. Les données ne sont pas excessivement volumineuses mais sont considérées comme telles en mettant en œuvre des technologies d'algorithmes distribués capables en principe de passer sans modification à des échelles plus importantes. Elles le sont pas ailleurs suffisamment pour mettre en défaut des implémentations pas ou mal optimisées comme la librairie *randomForest* de R.

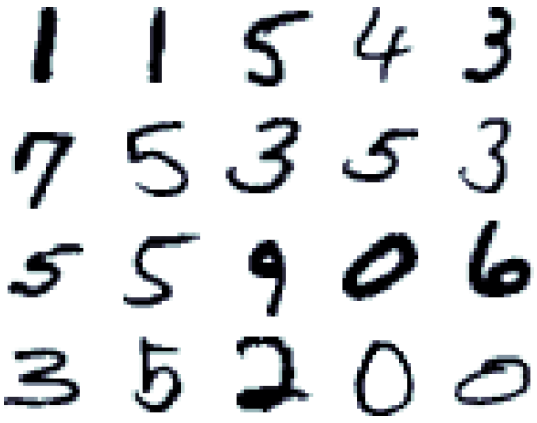


FIGURE 3.1 – *Quelques exemples d'images de caractères de la base MNIST.*

3.3 Reconnaissance de caractères (MNIST)

La reconnaissance de caractères manuscrits, notamment les chiffres de codes postaux, est un vieux problème de classification supervisée qui sert depuis de nombreuses années de base de comparaison entre les méthodes (cf. figure 3.1). Le site *MNIST DataBase* fournit des données ([Lecun et al., 1998]) et une liste de références, de 1998 à 2012 proposant des stratégies d'analyse avec des taux d'erreur de 12% à 0,23%. Les méthodes les plus récentes s'affrontent toujours sur ce type de données : [Bruna and Mallat, 2013] ... [Lee et al., 2016] ainsi que dans un concours *Kaggle* se terminant en décembre 2016.

De façon très schématique, plusieurs stratégies sont développées dans une vaste littérature sur ces données.

- Utiliser une méthode classique : k plus proches voisins, forêt aléatoire... sans trop raffiner mais avec des temps d'apprentissage rapides conduit à un taux d'erreur autour de 3%.

- Ajouter ou intégrer un pré-traitement des données permettant de recaler les images par des distorsions plus ou moins complexes.

- Construire une mesure de distance adaptée au problème car invariante par rotation, translation, homothétie, puis l'intégrer dans une technique d'apprentissage adaptée comme les k plus proches voisins. [Simard et al., 1998] définissent une distance dite tangentielle entre les images de caractères possédant ces propriétés d'invariance.

- D'autres pistes peuvent être explorées notamment celles des machines à noyaux par [Müller et al., 2001] qui illustrent ce type d'apprentissage par les données de MNIST sans semble-t-il faire mieux que la distance tangentielle.

- Les réseaux de neurones, renommés apprentissage profond (*deep learning*), et implémentant une architecture modélisant une convolution (*convolutional neural network*) impliquant les propriétés recherchées d'invariance, sont les algorithmes les plus compétitifs et les plus comparés ([Wan et al., 2013]). Noter le coût de calcul assez prohibitif pour l'estimation et l'optimisation de ces architectures complexes qui nécessitent des moyens matériels et des librairies sophistiquées reprenant toutes ce même jeu de données dans leur tutoriel. C'est le cas de *torch* en langage *Lua* ou de *Lasagne* (Theano) en Python. Même chose pour *H2O*, dont le tutoriel, très commercial, optimise les paramètres à partir de l'échantillon test malgré un risque évident de sur-apprentissage, ou encore pour *Tensor Flow* rendu récemment (12-2015) accessible par *Google*.

- Toujours en connexion avec le *deep learning* et illustrée par les mêmes données, [Bruna and Mallat, 2013] utilisent une décomposition, invariante par transformations, des images sur des bases d'ondelettes par *scattering*.

- ...

Attention, l'objectif de cette section n'est pas de concourir avec les meilleures solutions publiées. Les données sont simplement utilisées pour comparer les performances des implémentations de solutions élémentaires dans l'objectif d'un passage à l'échelle volume.

3.3.1 Les données

Les données représentent 60 000 images en niveaux de gris de chiffres manuscrits écrits par plusieurs centaines de personnes sur une tablette. Un ensemble de pré-traitements restreint les dimensions des images à $28 \times 28 = 784$ pixels. Celles-ci ont ensuite été normalisées par des transformations élémentaires. L' échantillon test indépendant est constitué de la même manière de 10 000 images.

Ces données ne sont pas réellement massives, elles tiennent en mémoire, mais leur volume suffit à montrer les limites, notamment en temps de calcul ou occupation mémoire, de certains algorithmes.

3.3.2 Solutions

Trois environnements sont comparés : R, Python (*Scikit-learn*) et *Spark* (*MLlib*). Une simple classification non supervisée (k-means) a également été testée sur les données. Les résultats sont équivalents entre les environnements à quelques remarques près. L'implémentation de R signale une difficulté à converger mais fournit avec le même temps d'exécution des résultats similaires à ceux de *Scikit-learn*. En revanche, l'algorithme des k plus proches voisins de R a des temps d'exécution rédhibitoires sur ces données alors que l'implémentation de *Scikit-learn* est lente mais raisonnable. Comme cet algorithme ne

s'adapte pas au le passage à l'échelle, il est abandonné les comparaisons pour se focaliser sur celui des *forêts aléatoires* ([Breiman, 2001]).

Les implémentations de R et *Scikit-learn* de ce dernier algorithme sont très voisines en terme de paramétrages et de valeurs par défaut. Des forêts de 250 arbres sont comparées. En prendre plus n'améliore pas notablement la qualité de prévision, 200 suffisent en fait. Les autres valeurs par défaut des paramètres, notamment le nombre (racine carrée du nombre de pixels) de variables tirées aléatoirement : `mtry` de *ranger*, `max_features` de *Scikit-learn*, `featureSubsetStrategy` de *MLlib*, sont satisfaisantes.

Pour satisfaire aux contraintes de *MapReduce* l'implémentation de *MLlib* est basée sur un algorithme de construction d'arbres issu du projet Google PLANET ([Panda et al., 2009]). Ce dernier dispose d'un nouveau paramètre : `maxBins` (= 32 par défaut) qui contrôle le nombre maximal de divisions prises en compte pour la recherche d'un nœud optimal. Ce paramètre provoque le regroupement des modalités d'une variable qualitative qui en comporte trop ou la transformation d'une variable quantitative en une variable ordinale. Il permet de contrôler les temps d'exécution de la construction de chaque arbre et leur complexité. Les données MNIST étant finalement assez "binaires" (présence ou non d'un pixel), ce paramètre n'a quasiment aucun effet sur ces données à moins de lui donner une valeur très faible.

Cette implémentation de *random forest* dans *MLlib*, soulève des problèmes concrets plus délicats. Alors qu'en principe pour réduire le biais, les arbres sont estimés sans limitation ni élagage, des problèmes de gestion de mémoire liés à la configuration matérielle, imposent de contrôler la profondeur maximum des arbres ou (`maxDepth`). En cas de manque de mémoire, Python (sous Windows, Mac OS ou Linux) gère ce problème par une extension (*swapping*) de ma mémoire, certes coûteuse en temps de calcul, sur le disque. En revanche ce même problème provoque une erreur et un arrêt intempestif de *Spark*. C'est donc par essais / erreurs successifs qu'il est possible de déterminer les valeurs des paramètres : nombre et profondeurs des arbres, acceptables sans erreur par chacun des nœuds du cluster. Des forêts de cent arbres de profondeur `maxDetph = 15` sont estimées.

3.3.3 Discussion

La précision des prévisions entre R et Python est analogue (cf. figures 3.2, 3.3), en revanche, les temps d'exécution, dépendent fortement du choix de la librairie R. Celle historique (*randomForest*), et intégrant le programme fortran de Breiman et Cutler interfacé avec R par [Liaw and Wiener, 2002], s'avère très lente (facteur 20) en comparaison à d'autres implémentations écrites en C. La dernière en date (*ranger* de [Wright and Ziegler, 2016]) concurrence l'implémentation en Python et Cython de *Scikit-learn*. La différence entre les deux, au profit de Pyhton, vient de ce que cette dernière implémentation sait gérer la parallélisation (4 cœurs pour l'ordinateur utilisé) en fixant le paramètre `n_jobs=-1`, même sous Windows, contrairement à *ranger* qui ne peut le faire que sous un système issu d'Unix.

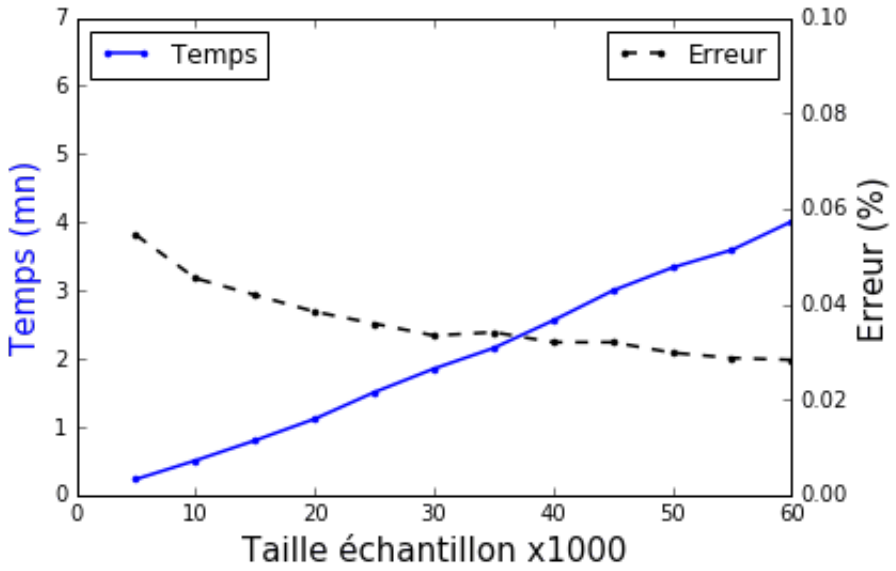


FIGURE 3.2 – *MNIST : forêts aléatoires avec R (ranger). Évolution du temps d'apprentissage et de l'erreur estimées sur l'échantillon test en fonction de la taille de l'échantillon d'apprentissage*

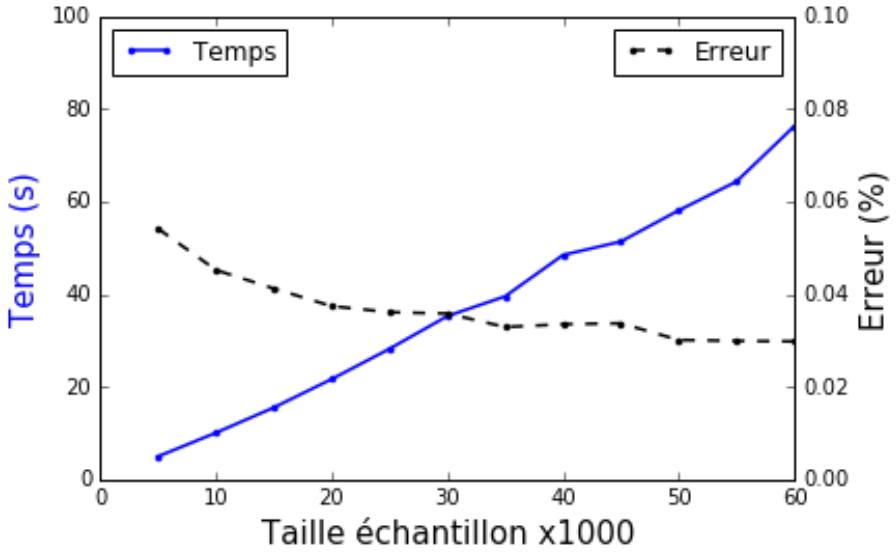


FIGURE 3.3 – *MNIST : forêts aléatoires avec Python (Scikit-learn). Évolution du temps d'apprentissage et de l'erreur estimées sur l'échantillon test en fonction de la taille de l'échantillon d'apprentissage.*

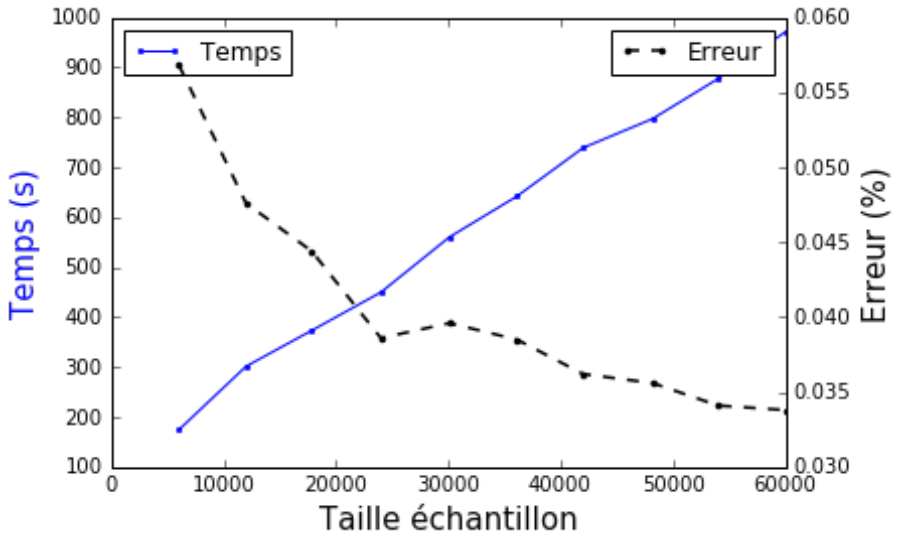


FIGURE 3.4 – *MNIST : forêts aléatoires avec Spark (MLlib). Évolution du temps d'apprentissage et de l'erreur estimées sur l'échantillon test en fonction de la taille de l'échantillon d'apprentissage.*

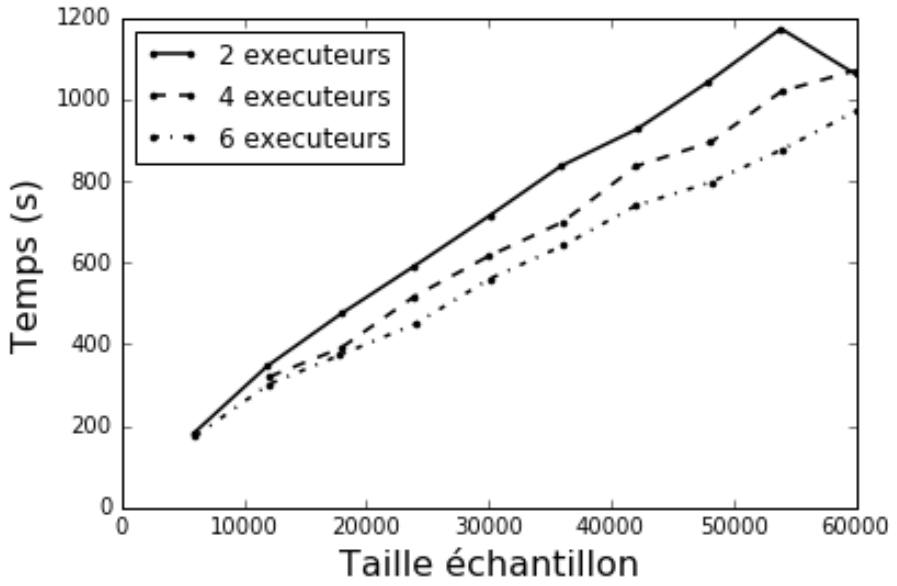


FIGURE 3.5 – *MNIST : Forêts aléatoires avec Spark (MLlib). Évolution du temps d'apprentissage pour plusieurs versions du cluster Spark avec 2, 4 ou 6 exécuteurs.*

Les contraintes dues à la gestion de la mémoire de l'environnement *Spark* rendent difficiles l'obtention de résultats strictement similaires en terme de qualité de prévision (cf. figure 3.4). La profondeur maximum des arbres influence fortement la qualité des estimations pour l'implémentation de *MLlib*. Le réglage de ce paramètre dépend directement de l'espace mémoire alloué à chaque exécuteur au moment de la déclaration du *cluster* par la commande `spark_context` ; une valeur de mémoire trop faible engendre une erreur qui arrête brutalement l'exécution. Seuls 100 arbres de profondeur maximum 15 sont estimés afin d'être sûr d'une exécution sans erreur. C'est une remise en cause de la tolérance aux pannes visées par *Spark* dans la version utilisée qui permet certes de gérer des données volumineuses mais pas, avec l'algorithme proposé, de mémoriser le modèle si celui-ci est trop complexe (nombre d'arbres, profondeur). Il aurait fallut disposer d'un *cluster* plus conséquent (plus de mémoire par exécuteur) pour obtenir des résultats comparables en précision et en temps. D'autre part, le temps d'exécution décroît bien avec le nombre d'exécuteurs (cf. figure 3.5) mais pas avec le facteur attendu pour atteindre la *scalabilité* de l'algorithme.

Enfin, remarquons que la croissance du temps d'apprentissage est linéaire en fonction de la taille de l'échantillon mais qu'au delà d'une taille d'échantillon de 50000, 55000 la précision des résultats semble arriver à un plateau (cf. figure 3.3), surtout avec un nombre (250) relativement élevé d'arbres. Il aurait sans doute été utile de tester un sous-échantillonnage *sans remise* plutôt que le *bootstrap* classique des forêts aléatoires. Ceci doit permettre en principe de construire des modèles d'arbres encore moins corrélés entre eux et donc de réduire la variance des prévisions mais cette option d'échantillonnage n'est pas prévue dans les implémentations utilisées. Sans cette dernière possibilité, considérer des échantillons d'apprentissage encore plus volumineux n'est pas une priorité pour améliorer la précision de ce problème de reconnaissance de caractères, ce sont d'autres types de modélisation, possédant des propriétés d'invariance, qu'il faut déployer.

En résumé, les expérimentations élémentaires réalisées sur ces données montrent qu'une architecture intégrée avec 8Go de mémoire et exécutant les implémentations *Python Scikit-learn* ou *R ranger* de *random forest* donne finalement de meilleurs résultats (temps, précision) qu'une architecture distribuée de 6 fois 7Go.

3.4 Recommandation de films

3.4.1 Marketing et systèmes de recommandation

Commerce en ligne

La rapide expansion des sites de commerce en ligne a pour conséquence une explosion des besoins en marketing quantitatif et *gestion de la relation client* (GRC) spécifiques à ce type de commerce. Ce domaine d'application est le principal moteur de développement des technologies liées au traitement des données massives et le premier fournisseur d'exemples.

La GRC en marketing quantitatif traditionnel est principalement basée sur la construction de modèles de scores : d'appétence pour un produit, d'attrition (*churn*) ou risque de rompre un contrat. Le commerce en ligne introduit de nouveaux enjeux avec le marché considérable de la publicité en ligne ou *recommandation*.

Systèmes de recommandation

La sélection ou recommandation automatique d'articles dans des approcches appelées encore *systèmes de recommandation* fait appel à des algorithmes dit de *filtrage*. Certains concernent des méthodes *adaptatives* qui suivent la navigation de l'internaute, son flux de clics, jusqu'à l'achat ou non. Ces approches sont basées sur des algorithmes de bandits manchots et ne font pas l'objet de cet exemple. D'autres stratégies sont définies à partir d'un historique des comportements des clients, d'informations complémentaires sur leur profil, elles rejoignent les méthodes traditionnelles de marketing quantitatif. D'autres enfin sont basées sur la seule connaissance des interactions clients $\times$ produits à savoir la présence / absence d'achats ou un ensemble d'avis recueillis sous la forme de notes d'appréciation de chaque produit consommé. On parle alors de filtrage collaboratif (*collaborative filtering*).

Filtrage collaboratif

Ce dernier cas a largement été popularisé par le concours *Netflix* où il s'agit de proposer un film à un client en considérant seulement la matrice très creuse $\boldsymbol{X}$: clients $\times$ films, des notes sur une échelle de 1 à 5. L'objectif est donc de prévoir le goût, la note, ou l'appétence d'un client pour un produit (livre, film...), qu'il n'a pas acheté, afin de lui proposer celui le plus susceptible de répondre à ses attentes.

Le filtrage collaboratif basé sur les seules interactions client $\times$ produits : présence / absence d'achat ou note d'appréciation fait généralement appel à deux grandes familles de méthodes :

Méthodes de voisinage fondées sur des indices de similarité (corrélation linéaire ou des rangs de Spearman...) entre clients ou (exclusif) entre produits :

- Basée sur le client avec l'hypothèse que des clients qui ont des préférences similaires vont apprécier des produits de façon similaire. Trouver un sous-ensemble S_i de clients qui notent de manière similaire au client i ; prévoir la note manquante : produit j par client i, par combinaison linéaire des notes de ce sous-ensemble S_i sur le produit j.
- Basée sur le produit avec l'hypothèse que les clients préfèreront des produits similaires à ceux qu'ils ont déjà bien notés. Trouver un sous-ensemble S_j de produits notés de façon similaire par le client i ; prévoir la note manquante : produit j par client i, par combinaison linéaire des notes de ce sous-ensemble S_j du client i.

Modèle à facteurs latents basé sur une décomposition de faible rang avec une éventuelle contrainte de régularisation, de la matrice très creuse $\boldsymbol{X}$ des notes ou avis clients $\times$ produits. La note du client i sur le produit j est approchée par le produit scalaire de deux facteurs $\boldsymbol{W}_i$ et $\boldsymbol{H}_j$ issus de la décomposition.

Complétion de matrice . À la suite du succès du concours *Netflix*, [Candes and Tao, 2010] ont formalisé de façon plus générale le problème de la recherche de facteurs latents comme celui de la complétion d'une matrice. Notons $P_\Omega(\boldsymbol{X})$ la "projection" de la matrice $\boldsymbol{X}$ qui consiste à remplacer toutes les valeurs inconnues (absence de note) par des valeurs nulles.

$$\min_{\boldsymbol{M}} \left(||P_\Omega(\boldsymbol{X} - \boldsymbol{M})||_2^2 + \lambda ||\boldsymbol{M}||_* \right) \tag{3.1}$$

où $||.||_2$ désigne la norme de Froebenius et $||.||_*$ celle nucléaire (somme des valeurs singulières) qui introduit une pénalisation afin de réduire le rang mais avec des propriétés de convexité. Le rôle de $P_\Omega(\boldsymbol{X})$ est important, l'ajustement n'est calculé que sur les seules valeurs connues de $\boldsymbol{X}$.

La littérature est très abondante sur le sujet qui soulève plusieurs questions dont :

- l'évaluation d'un système de recommandation car il n'est pas possible de savoir si le client a acheté sous l'effet de la proposition commerciale. Seul le montage complexe d'expérimentations (dites *AB testing*) permet de comparer deux stratégies sous forme de deux sites distincts auxquels sont aléatoirement adressés les internautes.
- Initialisation (*cold start problem*) des termes de la matrice avec l'arrivée de nouveaux clients ou la proposition de nouveaux produits.

Par ailleurs, des systèmes hybrides intègrent ces données d'interaction avec d'autres informations sur le profil des clients (âge, sexe, prénom...) ou encore sur la typologie des produits (genre, année...).

3.4.2 Complétion de matrices

La littérature sur ce sujet a connu un développement trop important pour espérer en faire une revue synthétique en quelques lignes. Nous nous limitons volontairement aux seules méthodes implémentées dans des librairies facilement accessibles des environnements R, Python et *Spark* considérés.

sofImpute

[Mazumder et al., 2010] puis [Hastie et al., 2015] proposent deux algorithmes de complétions implémentés dans la librairie *softImpute* de R pour résoudre le problème d'optimisation (3.1).

Le premier ([Mazumder et al., 2010]) itère des décompositions en valeurs singulières (SVD) seuillées pour reconstruire les valeurs manquantes. Le seuillage de la SVD consiste à annuler les valeurs singulières de la décomposition inférieure à une valeur λ forçant ainsi des solutions de plus faible rang. Des astuces permettent de réduire les temps de calcul. Il n'est pas nécessaire de calculer tous les termes de la SVD et la faible densité des matrices permet des économies substantielles de mémoire. D'autre part les SVD successives peuvent bénéficier de l'itération précédente comme initialisation.

Le deuxième algorithme est basé sur une MMMF (maximum margin matrix factorization) de [Srebro et al., 2005]. La matrice $\boldsymbol{X}$ à compléter est de dimension $(n \times p)$. Le critère à optimiser est donc

$$\min_{\boldsymbol{A}_{n\times r},\boldsymbol{B}_{p\times r}} ||P_\Omega(\boldsymbol{X} - \boldsymbol{A}\boldsymbol{B}^T)||_2^2 + \lambda \left(||\boldsymbol{A}||_2^2 + ||\boldsymbol{B}||_2^2\right),$$

où $r < \min(n, p)$ désigne le rang des matrices $\boldsymbol{A}$ et $\boldsymbol{B}$. Le critère n'est pas convexe mais dont la minimisation est résolue au moyen d'un algorithme de moindres carrés alternés (ALS). La minimisation est recherchée itérativement et alternativement en $\boldsymbol{A}$ puis $\boldsymbol{B}$. Si $\boldsymbol{A}$ est fixée, chercher la meilleure matrice $\boldsymbol{B}$ revient à résoudre p régressions *ridge* : chaque colonne $\boldsymbol{X}_j$ est expliquée par les r colonnes de $\boldsymbol{A}$ et pour les valeurs inconnues de $\boldsymbol{X}_j$, les lignes correspondantes de la matrice $\boldsymbol{A}$ sont ignorées de la $j$ème régression. La procédure équivalente est mise en place en fixant $\boldsymbol{B}$ et calculant n régressions *ridge*. La décomposition obtenue sous la forme $\boldsymbol{X} = \boldsymbol{A}\boldsymbol{B}^T$ est donc une approximation à partir des seules valeurs observées comportant des valeurs manquantes. Ces valeurs à compléter sont obtenues alors par le résultat du produit $\boldsymbol{A}\boldsymbol{B}^T$. Bien entendu cela suppose que les valeurs manquantes ne sont pas des colonnes ou des lignes entières.

L'algorithme proposé par [Hastie et al., 2015] hybride les deux précédents : moindres carrés alternés et décomposition en valeurs singulières. Les expérimentations sur des données simulées et celles de *MovieLens* montrent une convergence plus rapide de la fonction objectif.

Malheureusement, les auteurs ne donnent pas de résultats (RMSE) sur la précisions de la recommandation pour les données *MovieLens* et ceux fournis pour les données *Netflix* ne sont pas très convaincants. Une amélioration du score de base (*baseline*) de 0,1%, à comparer avec les 10% obtenus par la solution gagnante du concours.

Complétion par NMF

L'option `ALS` de *softImpute* approche finalement le problème de complétion par une factorisation. Malheureusement, [Hastie et al., 2015] n'introduisent pas dans leur comparaison, ni dans leurs références, d'autres approches de factorisation bien que la NMF (non negative matrix factorisation) soit largement citée et utilisée en filtrage collaboratif pour la recherche de facteurs latents sous la contrainte que toutes les valeurs soient positives ou nulles.

Problème d'optimisation. Comme la décomposition en valeurs singulières (SVD), la factorisation non négative de matrice (NMF) décompose une matrice rectangulaire $(n \times p)$ en le produit de deux matrices de faible rang. La solution de la SVD est unique, obtenue par des algorithmes efficaces et génère des facteurs orthogonaux permettant des représentations graphiques. En revanche la NMF fournit des résultats mieux adaptés à des notes, des effectifs, mais plusieurs algorithmes sont en concurrence qui convergent vers des optimums locaux voire même peuvent se bloquer sur une solution sous-optimale sur le bord du cône.

Soit $\boldsymbol{X}$ une matrice $(n \times p)$ ne contenant que des valeurs non négatives et sans ligne ou colonne ne comportant que des 0 ; r un entier choisi relativement petit devant n et p. La factorisation non-négative de la matrice $\boldsymbol{X}$ est la recherche de deux matrices $\boldsymbol{W}_{n\times r}$ et $\boldsymbol{H}_{r\times p}$ ne contenant que des valeurs positives ou nulles et dont le produit approche $\boldsymbol{X}$.

$$\boldsymbol{X} \approx \boldsymbol{W}\boldsymbol{H}.$$

Le choix du *rang* de factorisation $r << min(n, p)$ assure une réduction drastique de dimension et donc des représentations parcimonieuses. Évidemment, la qualité d'approximation dépend de la parcimonie de la matrice initiale.

La factorisation est résolue par la recherche d'un optimum local au problème d'optimisation :

$$\min_{\boldsymbol{W},\boldsymbol{H}\geq 0} \left[L(\boldsymbol{X}, \boldsymbol{W}\boldsymbol{H}) + P(\boldsymbol{W}, \boldsymbol{H})\right].$$

L est une fonction perte mesurant la qualité d'approximation et P une fonction de pénalisation optionnelle ; L est généralement soit un critère de moindres carrés (LS ou norme de Frobenius des matrices ou norme trace), soit la divergence de Kullback-Leibler (KL) ; P est une pénalisation optionnelle de régularisation utilisée pour forcer les propriétés recherchées des matrices $\boldsymbol{W}$ et $\boldsymbol{H}$; par exemple, la parcimonie des matrices ou la régularité des solutions dans le cas de données spectrales.

Non seulement la solution est locale car la fonction objectif n'est pas convexe en $\boldsymbol{W}$ et $\boldsymbol{H}$ mais en plus la solution n'est pas unique. Toute matrice $\boldsymbol{D}_{r\times r}$ non négative et inversible fournit des solutions équivalentes en terme d'ajustement :

$$\boldsymbol{X} \approx \boldsymbol{W}\boldsymbol{D}\boldsymbol{D}^{-1}\boldsymbol{H}.$$

Algorithmes. De nombreuses variantes algorithmiques et sur la forme des pénalisations ont été publiées et implémentées généralement en Matlab, parfois en C, quelques unes spécifiques en R ; [Berry et al., 2007] proposent un tour d'horizon de certaines tandis que [Gaujoux and Seoighe, 2010] en ont implémentées dans R pour rendre facilement possible la comparaison des résultats. Trois familles d'algorithmes sont généralement citées :

- *Standard NMF algorithm with multiplicative update*,
- *Alternate Least Square (ALS) algorithm*,

- Descente du gradient.

Chacun de ces algorithmes peut par ailleurs être initialisé de différentes façons :

- plusieurs initialisations aléatoires de $\boldsymbol{W}$ et $\boldsymbol{H}$, le meilleur ajustement est conservé,
- *non-negative double singular value decomposition (NNSVD)*,
- une classification (k-means) des lignes ou des colonnes,
- parts positives de matrices issues d'une analyse en composantes indépendantes (ACI),
- ...

Entre le choix de la fonction objectif : fonction perte (LS ou KL) et l'éventuelle pénalisation (L^1, L^2, régularité), le choix de l'algorithme ou d'une de ses variantes, le choix de l'initialisation... cela fait beaucoup d'options à comparer, tester. Comme toujours avec une nouvelle méthode et la pression de publication, de très nombreuses variantes apparaissent avant qu'une sélection *naturelle* n'opère pour aboutir à des choix plus efficaces et consensuels d'options en fonction du type de données traitées.

[Berry et al., 2007] décrivent très brièvement les principes de ces différents algorithmes et commentent leurs propriétés : convergence, complexité.

L'algorithme initial de [Lee and Seung, 1999] (*Multiplicative update algorithms*) peut converger vers un point stationnaire pas nécessairement minimum local, voire un point de la frontière même pas point stationnaire. Ces cas sont heureusement rares en pratique mais la convergence est considérée comme lente, demandant plus d'itérations que ses concurrents alors que chaque itération nécessite de nombreux calculs ($O(n^3)$). Les algorithmes de descente du gradient posent des questions délicates concernant le choix des deux pas de descente. La dernière famille d'algorithmes : moindres carrés alternés (ALS), exploite le fait que si le problème n'est pas à la fois convexe en $\boldsymbol{W}$ et $\boldsymbol{H}$, il l'est soit en $\boldsymbol{W}$ soit en $\boldsymbol{H}$. Il suit le principe ci-dessous et possède de bonnes propriétés (convergence, complexité).

NMF par ALS

- $\boldsymbol{W}$ =random(n, r)
- **FOR** $i = 1$ à `Maxiter`
  - Résoudre en $\boldsymbol{H}$: $\boldsymbol{W'WH = W'X}$
  - Mettre à 0 les termes négatifs de $\boldsymbol{H}$
  - Résoudre en $\boldsymbol{W}$: $\boldsymbol{HH'W' = HX'}$
  - Mettre à 0 les termes négatifs de $\boldsymbol{W}$

L'un des inconvénients du (*Multiplicative update algorithms*) originel est que si un élément des matrices $\boldsymbol{W}$ ou $\boldsymbol{H}$ prend la valeur 0, il reste à cette valeur, n'explorant ainsi pas de solutions alternatives. L'ALS est lui plus souple en permettant d'échapper à de mauvaises solutions locales.

Critères de choix. Les auteurs proposent différents critères pour aider aux choix des méthodes, algorithmes et paramètres, notamment celui du rang r de factorisation, pouvant intervenir au cours d'une étude. L'évaluation de la "stabilité" de plusieurs exécutions de NMF repose sur des critères (silhouette, consensus, corrélation cophénétique) issues des méthodes de classification non supervisée. Pour adapter ces critères à la NMF, la notion de classe d'une observation (resp. d'une variable) est remplacée par la recherche du facteur, ou élément de la base (colonne de $\boldsymbol{W}$ resp. de $\boldsymbol{H}$), pour laquelle l'observation (resp. la variable) a obtenu la plus forte contribution. Comme pour le choix d'une dimension, d'un nombre de classes, seules des heuristiques sont proposées dans la littérature pour le difficile choix de r pour lequel il n'y a pas de critère nettement tranché.

En revanche, dans l'exemple traité de recherche d'une recommandation par facteurs latents, une approche de type supervisée est possible. Il suffit d'extraire de la matrice initiale un sous-ensemble de notes, nombre d'achats, qui constituera un sous-échantillon de validation pour optimiser le rang des matrices et l'éventuel paramètre de pénalisation. Plusieurs méthodes, plusieurs approches peuvent également être comparées sur un échantillon test construit de la même façon. Le concours *Netflix* était construit sur ce principe avec une fonction perte quadratique (RMSE).

Implémentations. La librairie *NMF* de R implémente 11 méthodes ; 9 sont basées sur l'algorithme initial de [Lee and Seung, 1999] (*Multiplicative update algorithms*) avec différentes options de perte (LS, KL) et de pénalisation ou d'arrêt, deux sont basées sur les moindres carrés alternés (ALS) avec contrainte de parcimonie sur les lignes ou les colonnes. Systématiquement, l'option est offerte, et encouragée, de lancer plusieurs exécutions à partir de plusieurs initialisations aléatoires pour sélectionner les options "optimales" puis, une fois les choix opérés, pour retenir la meilleure parmi un ensemble d'exécutions.

Un algorithme spécifique (projection du gradient) et principalement utilisé pour de la décomposition d'images par NMF est accessible dans la librairie *Scikit-learn* pour matrices denses ou creuses.

Attention, ces algorithmes ne sont pas adaptés à la complétion de matrice. Appliqués à une matrice creuse de notes, ils visent à reconstruire précisément les valeurs manquantes considérées comme nulles et n'atteignent pas l'objectif recherché de compléter les valeurs manquantes.

MLlib implémente l'algorithme NMF par ALS avec deux options. Dans celle par défaut, la fonction objectif est une norme L_2 (Froebenius) entre la matrice et sa reconstruction

mais cette norme, comme dans le problème (3.1) de complétion, n'est calculée *que* sur les valeurs (notes) observées. Des contraintes (L_1 et L_2) introduisent une régularisation pour renforcer la parcimonie de la factorisation qui opère donc une complétion. Les paramètres sont à optimiser par validation croisée. La documentation n'est pas explicite sur le déroulement précis de l'algorithme mais cite [Koren et al., 2009] qui le sont un peu plus. La deuxième option concerne des effectifs de ventes, de chargements de films. Les valeurs nulles ne sont pas des données manquantes.

3.4.3 MovieLens

Les données

Des données réalistes croisant plusieurs milliers de clients et films, sont accessibles en ligne. Il s'agit d'une extraction du site `movielens.org` qui vous aide à choisir un film. Quatre tailles de matrices très creuses sont proposées contenant seulement les appréciations connues d'un client sur un film.

`100k` 100 000 évaluations de 1000 utilisateurs de 1700 films.

`1M` Un million d'évaluations par 6000 utilisateurs sur 4000 films.

`10M` Dix millions d'évaluations par 72 000 utilisateurs sur 10 000 films.

`20M` Vingt deux millions d'évaluations par 138 000 utilisateurs sur 27 000 films.

Complétion

Les programmes ont été testés, sur la matrice de 100 000 notes puis directement appliqués à celle de 22 millions de notes. Les notes sont stockées dans une matrice de format creux ou *sparse*. Il s'agit simplement de mémoriser la liste des triplets contenant le numéro de la ligne (usager), de la colonne (film) pour lesquels la note est connue. La plus grande matrice se réfère donc à des dimensions de 138k lignes pour 27k colonnes ($3,710^9$ éléments) mais comme moins de 1% des notes sont connues, elle tient en mémoire avec format creux.

L'évaluation de la complétion de matrice se ramène à un problème, ou tout du moins à la même démarche d'évaluation, qu'un problème d'apprentissage. La fonction perte utilisée pour mesure les reconstructions des notes est la racine de l'écart quadratique (RMSE). Un sous-ensemble de notes est extrait (10 % soit près de deux millions de notes) pour jouer le rôle d'échantillon test. L'optimisation du rang de la factorisation et celle de la pénalisation peut être calculée par validation croisée mais compte tenu des tailles des matrices, l'extraction préalable d'un échantillon de validation est préférable. Seules deux approches ont été testées car *Scikit-learn* ne propose pas d'algorithme de complétion.

La première approche a consisté à utiliser la librairie *softImpute* (option `ALS`) de R pour plusieurs valeurs des paramètres (rang et pénalisation) afin de compléter l'échantillon

TABLE 3.1 – *SoftImpute (ALS) appliqué aux données MovieLens. Temps (minutes) de calcul et RMSE en fonction du rang maximum de la factorisation et du paramètre de régularisation.*

Rang Max	λ	Temps	RMSE
4	1	5.6	1.07
10	10	12.6	1.020
10	20	12.2	1.033
15	10	19.4	1.016
20	1	26.9	1.020
20	10	26.1	1.016
20	15	24.4	1.018
20	20	27.0	1.016
30	20	40.1	1.020

test puis calculer le RMSE. Les résultats (processeur 4 cœurs séquencé à 1,7GHz, 8Go de RAM sous Windows) sont résumés dans le tableau 3.1.

La deuxième approche met en œuvre la NMF de la librairie *MLlib*. Une procédure sommaire de validation croisée sur les données 100k a montré qu'un faible rang, entre 4 et 8 était préférable alors que le paramètre de pénalisation (L_1) est simplement laissé à sa valeur par défaut (0.01).

Discussion

Même avec une démarche qui pourrait conduire à du sur-apprentissage, la librairie de R *softImpute* conduit à des résultats décevants sur les données *MovieLens* (cf. tableau 3.1) alors que la fonction NMF de *MLlib* conduit, sans effort d'optimisation élaboré, à des résultats satisfaisants (RMSE de 0,82), parmi les meilleurs trouvés dans les blogs de discussion sur le sujet.

La figure 3.6 représente le RMSE et le temps de calcul pour la factorisation en fonction du nombre de notes prises en compte dans la matrice concernée. Il apparaît que le temps de calcul croît approximativement un peu plus vite que linéairement avec ce nombre de notes tandis que l'erreur continue de décroître mais de moins en moins rapidement.

La figure 3.7 montre par ailleurs que le temps de factorisation décroît avec le nombre d'exécuteurs du *cluster*. Ce ne sont pas les facteurs attendus pour une décroissance linéaire du temps en fonction de ce nombre d'exécuteurs mais le gain est significatif.

En résumé, la librairie *Scikit-learn* ne propose pas d'algorithme de complétion et est absente de la comparaison alors que *Spark MLlib* conduit aux meilleurs résultats, bien meilleurs en temps et précision que l'implémentation réalisée dans *softImpute* de R. Parmi

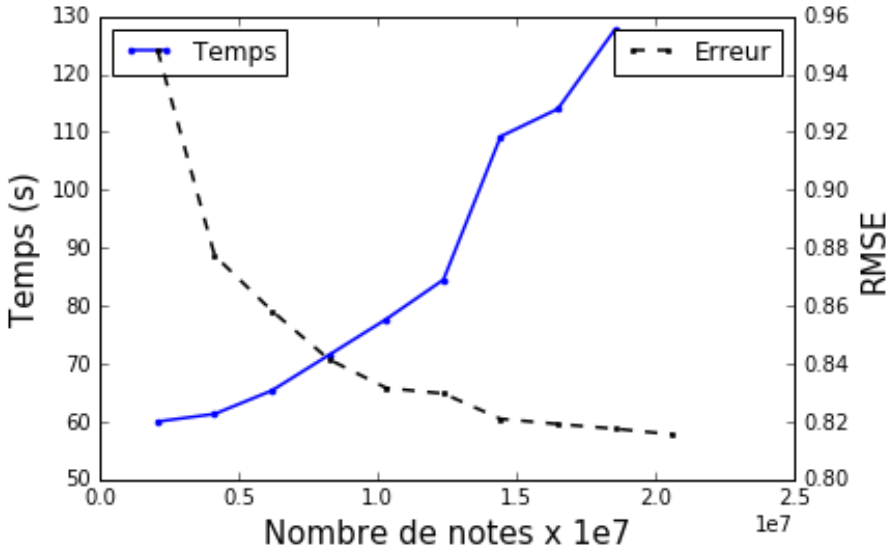


FIGURE 3.6 – *MovieLens : complétion par NMF (MLlib). Évolution du temps d'exécution de la factorisation et de l'erreur de complétion (RMSE) de l'échantillon test de notes en fonction du nombre de notes (taille de la matrice creuse) prises en compte dans la factorisation.*

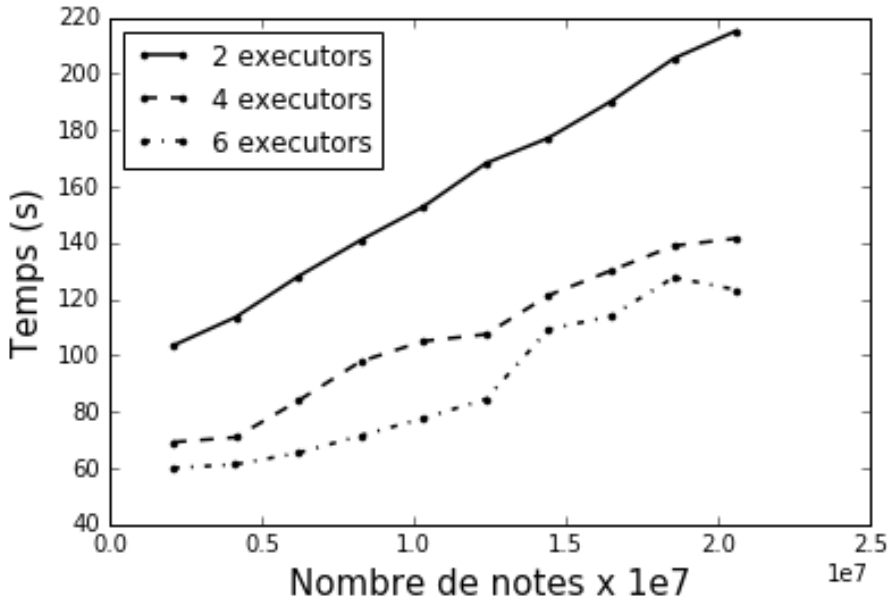


FIGURE 3.7 – *MovieLens : complétion par NMF (MLlib). Évolution du temps d'exécution de la factorisation en fonction du nombre de notes (taille de la matrice creuse) pour plusieurs versions du cluster avec 2, 4 ou 6 exécuteurs.*

une littérature déjà importante, ce n'est sans doute pas le meilleur algorithme qui est implémenté dans *MLlib* mais celui-ci (NMF par ALS) s'avère efficace et surtout bien adapté à l'architecture distribuée en association à *Spark* pour archiver, manipuler des matrices très creuses.

3.5 Catégorisation de produits

3.5.1 Objectifs

Fouille de texte

Il s'agit d'un problème récurrent du commerce en ligne qui se présente sous la forme suivante. Un commerçant partenaire d'un site en ligne souhaite proposer l'ensemble d'un catalogue d'articles à la vente, chacun décrit par un court texte en langage naturel. Pour assurer le maximum de visibilité des produits, ce site doit assurer une catégorisation homogène des produits malgré leurs origines très variées : les commerçants partenaires. A partir de la liste des articles déjà présents sur le site (base d'apprentissage), il s'agit de déterminer la catégorie d'un nouvel article, c'est-à-dire permettre de l'introduire dans l'arborescence des catégories et sous-catégories du site ; c'est donc encore d'un problème de discrimination ou classification supervisée mais appliquée à de la fouille de textes.

Étapes

Le traitement se décompose en trois étapes bien distinctes. La première est un pré-traitement ou nettoyage des données. La deuxième est une *vectorisation* ou quantification des textes, de façon à remplacer les mots par des nombres d'occurrences ou plutôt par les valeurs prises par une liste de variables (*features*) mesurant des fréquences relatives d'une liste déterminée de regroupements de mots. Enfin, une fois construite une matrice, généralement creuse, différentes méthodes d'apprentissage sont testées dans la 3ème étape afin de prévoir, au mieux, la catégorie des articles d'un échantillon test.

La préparation des données (*data munging*) est très souvent l'étape la plus délicate et la plus *chronophage* de la Science des Données de la vraie vie, par opposition à des données rendues publiques et souvent exploitées pour illustrer ou comparer des méthodes. Ces dernières, comme celles de l'exemple de reconnaissance des caractères, sont très « propres » : pas de données manquantes, ou trop atypiques, d'erreur de codage... Souvent négligé des présentations pédagogiques, le pré-traitement est néanmoins primordial pour assurer la qualité et le pouvoir prédictif des nouvelles variables ainsi construites et qui influent directement sur la pertinence des modèles. D'autre part, ces phases d'extraction, nettoyage, recodage, transformation... des données, conduisent très souvent à une réduction drastique de leur volume. Des données initialement massives, il ressort une matrice souvent

adaptée à la mémoire d'un plus ou moins gros ordinateur et ces pré-traitements peuvent rendre inutile une architecture distribuée pour la suite des analyses.

3.5.2 Préparation des textes

Voici la liste des traitements généralement opérés sur des données textuelles.

Nettoyage Suppression des caractères mal codés et de ponctuation, transformation des majuscules en minuscules, en remarquant que ces transformations ne seraient pas pertinentes pour un objectif de détection de pourriels.

Suppression des mots vides (*stop words*) ou mots de liaison, articles qui n'ont *a priori* pas de pouvoir discriminant.

Racinisation ou *stemming*. Les mots sont réduits à leur seule racine afin de réduire la taille du dictionnaire.

Hashage Une fonction de hashage est appliquée pour transformer chaque mot en un index unique en ajoutant une *astuce* ou *hashing trick*. Le nombre de valeurs possibles (modulo une division entière) prises par la fonction de hashage est un paramètre `n_hash`. Ainsi, les mots sont automatiquement et arbitrairement regroupés pour aboutir à un nombre prédéterminé de codes possibles. Plus précisément, la fonction de hashage h est définie sur l'espace des entiers naturels et à valeurs $i = h(j)$ dans un ensemble fini $(1, \ldots, \texttt{n\_hash})$ des variables ou *features*. Ainsi le poids de l'indice i, du nouvel espace, est l'association de tous les poids d'indice j tels que $i = h(j)$ de l'espace original. Ici, les poids sont associés d'après la méthode décrite par [Weinberger et al., 2009]. La fonction h n'est pas générée aléatoirement. Ainsi pour un même fichier d'apprentissage (ou de test) et pour un même entier `n_hash`, le résultat de la fonction de hashage est identique.

Xgram La fonction de hashage est appliquée aux mots (`unigram`) ou aux couples (`bigram`) de deux mots consécutifs. Ce deuxième choix permet de lever beaucoup d'ambiguïté du langage mais risque de faire exploser le volume du dictionnaire. C'est encore un paramètre à optimiser.

TF-IDF Il permet de faire ressortir l'importance relative de chaque mot m (ou couples de mots consécutifs) dans un texte-produit ou un document d, par rapport à la liste entière des documents. La fonction $TF(m, d)$ compte le nombre d'occurrences du mot m dans le document d. La fonction $IDF(m)$ mesure l'importance du terme dans l'ensemble des documents ou descriptifs en donnant plus de poids aux termes les moins fréquents car considérés comme les plus discriminants (motivation analogue à celle de la métrique du $\chi 2$ en analyse des correspondances). $IDF(m) = \log \frac{D+1}{f(m)+1}$

(version *smooth* adoptée dans *Scikit-learn* et *MLlib*) où D est le nombre de documents, la taille de l'échantillon d'apprentissage, et $f(m)$ le nombre de documents ou descriptifs contenant le mot m. La nouvelle variable ou *features* est $V_m(d) = TF(m,d) \times IDF(m)$.

Comme pour les transformations des variables quantitatives (centrage, réduction), les mêmes transformations, c'est-à-dire la même fonction de hashage et le même ensemble de pondérations (IDF), sont calculés, appliqués sur l'échantillon d'apprentissage puis appliqués sur celui de test.

Il s'agit finalement d'évaluer distinctement ces trois étapes au regard des technologies disponibles. De façon schématique, la première (nettoyage) est très facilement parallélisable et peut se décomposer en séquences de traitements ou d'étapes fonctionnelles *Map* tout à fait adaptées à une architecture distribuée et donc des données très massives. En revanche, les étapes suivantes (vectorisation, modélisation) nécessitent des comparaisons plus fouillées.

3.5.3 *Cdiscount*

Il s'agit d'une version simplifiée du concours proposé par *Cdiscount* et paru sur le site `datascience.net`. Les données d'apprentissage sont accessibles sur demande auprès de *Cdiscount* dans le forum de ce site et les solutions gagnantes ont été présentées aux journées de Statistique de Montpellier par [Goutorbe et al., 2016]. Comme les solutions de l'échantillon test du concours ne sont pas et ne seront pas rendues publiques, un échantillon test est donc extrait pour l'usage de cet exemple. L'objectif est de prévoir la catégorie d'un produit à partir de son descriptif. Seule la catégorie principale (1er niveau de 47 classes) est modélisée au lieu des trois niveaux demandés dans le concours (5789 classes). L'objectif n'est pas de faire mieux que les solutions gagnantes basées sur des *pyramides* complexes de régressions logistiques programmées en Python mais de comparer les performances des méthodes et technologies en fonction de la taille de la base d'apprentissage ainsi que d'illustrer, sur un exemple réel, le pré-traitement de données textuelles. La stratégie de sous ou sur-échantillonnage des catégories très déséquilibrées qui permet d'améliorer la prévision n'a pas été mise en œuvre.

Les données se présentent sous la forme d'un fichier texte de 3.5 Go. Il comporte quinze millions de lignes, un produit par ligne contenant sa catégorie et le descriptif. La nature brute de ces données, leur volume, permet de considérer toute la chaîne de traitement et pas seulement la partie apprentissage.

R, peu adapté à une fouille de textes d'un tel volume, n'a pas été testé, seules sont comparées deux séquences d'analyses identiques réalisées avec Python *Scikit-learn* et *Spark Mllib*.

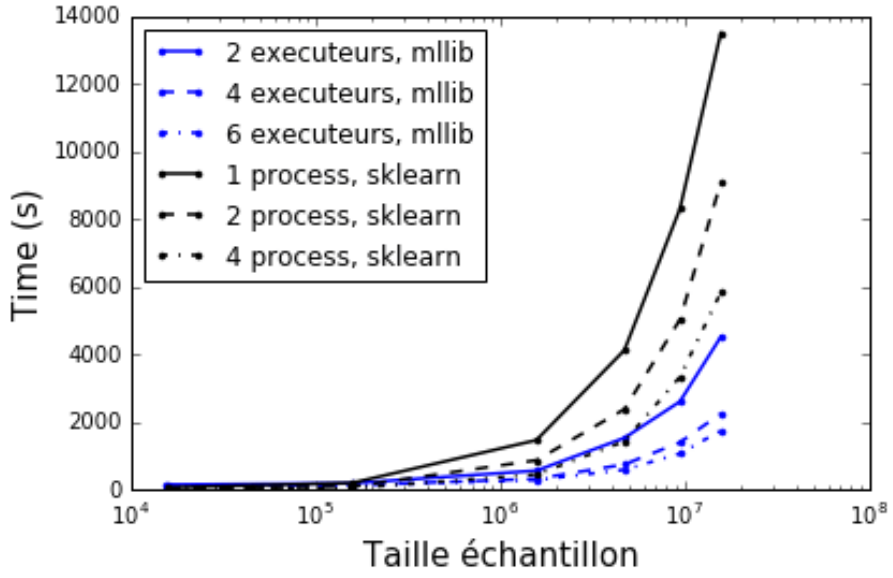


FIGURE 3.8 – *Cdiscount : nettoyage des données. Évolution du temps de nettoyage pour plusieurs versions du cluster Spark avec 2, 4 ou 6 exécuteurs et avec Scikit-learnet en Python (Scikit-learn) avec 1, 2, 4 processeurs.*

Nettoyage

La première étape opère une série de traitements élémentaires (suppression des caractères mal codés et de ponctuation, transformation des majuscules en minuscules) puis certains adaptés à l'objectif : suppression des mots vides (*stop words*) et *racinisation* ou *stemming* des mots.

Ces traitements ont été exécutés avec différentes configurations du *cluster* (2, 4, 6 exécuteurs de 7Go) ainsi qu'en contrôlant le nombre de cœurs (1,2 ou 4) actifs d'un Macbook Pro (2,2 GHz) avec 16Go de RAM. L'échelle des abscisses (figure 3.8) est logarithmique mais le temps d'exécution croît approximativement linéairement avec le volume des données à nettoyer. Les résultats de *s*calabilité sont ceux attendus en fonction du nombre de cœurs ou du nombre d'exécuteurs et nettement en faveur de l'architecture distribuée. Intuitivement, une architecture physiquement plutôt que virtuellement distribuée devrait encore conduire à de meilleurs résultats sur de très gros volumes avec une parallélisation plus efficace des opérations de lecture.

En résumé, une architecture distribuée avec *Spark* est plus efficace pour l'étape de nettoyage qu'une architecture intégrée.

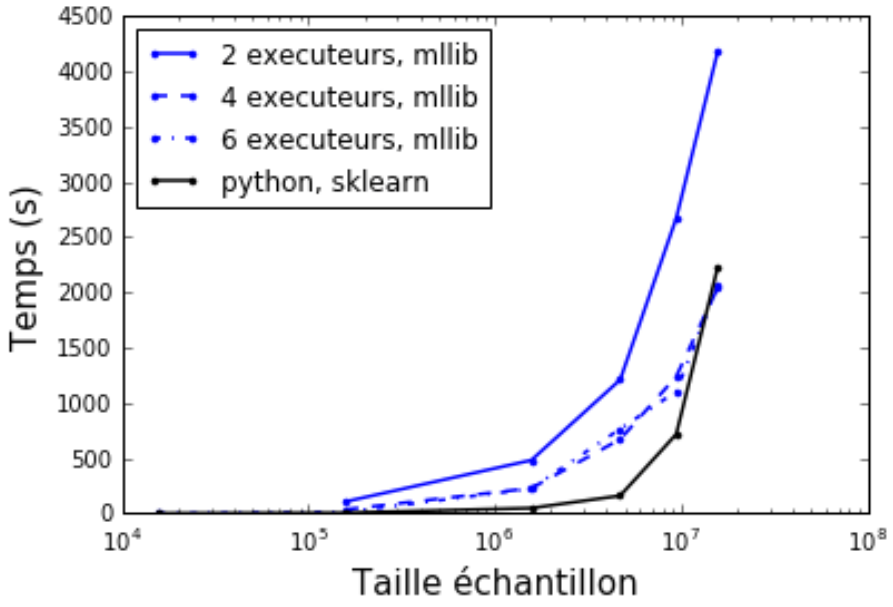


FIGURE 3.9 – *Cdiscount : vectorisation. Évolution du temps de la vectorisation (hashage et TF-IDF) pour plusieurs versions du cluster Spark avec 2, 4 ou 6 exécuteurs et en Python (Scikit-learn) avec* `n_hash` *= 60 000 variables.*

Vectorisation

Les étapes de hashage puis de calcul des fréquences (TF) et des pondérations (IDF) sont nécessairement associées avec la gestion d'un dictionnaire commun dont le cardinal est la valeur du paramètre `n_hash`. Le résultat de cette étape est une matrice de fréquences relatives comportant autant de lignes que de textes et `n_hash` colonnes. Les documentations ne sont pas très explicites sur les implémentations de ces traitements mais, de toute façon, plusieurs étapes (au moins 3 ?) *Mapreduce* sont nécessaires avec *Spark MLlib*. Des valeurs plus grandes (100k, 500k, 1M) de `n_hash` ont été testées mais comme

- la précision (erreur de classement) obtenue avec *Scikit-learn* n'est pas meilleure au delà de `n_hash =60 000` variables,
- *Spark Mllib* provoque une erreur mémoire à partir de `n_hash =100 000` variables dans l'étape suivante d'apprentissage,

seuls les résultats pour des valeurs plus faibles car utiles de `n_hash` (10k, 35k, 60k) sont considérés. D'autre part, les graphiques ne sont pas superposés par souci de lisibilité mais la valeur du paramètre `n_hash` n'influe pas très sensiblement sur le temps d'exécution.

La figure 3.9 montre des temps d'exécution en faveur d'une architecture intégrée (Macbook pro, 16Go), à moins sans doute de traiter des données très volumineuses car dans

ce cas, la gestion de la mémoire virtuelles(*swapping*) pénalise l'exécution. Mais, il suffirait alors de renforcer la RAM au niveau, par exemple, de celle globale du *cluster* pour les temps concurrentiels.

En résumé, une architecture intégrée, en renforçant éventuellement la mémoire, est préférable à une architecture distribuée pour l'étape de *vectorisation* de documents.

Apprentissage

Plusieurs méthodes d'apprentissage (arbres, *random forest*) ont été testées sur ces données pour finalement choisir une ou plutôt un ensemble de 47 régressions logistiques. En effet, par défaut, *Scikit-learn*, comme *MLlib*, estiment une version multimodale de la régression logistique en considérant des modèles prévoyant l'occurrence d'une classe contre celle des autres. Il est important de noter que, pour le problème initial, l'objectif est de prévoir le troisième niveau de catégorie qui comporte 5789 classes. Les solutions gagnantes, estimées avec des valeurs de `n_hash` = 500k, produisent donc un ensemble de modèles de régressions logistiques nécessitant 20Go de mémoire rien que pour l'archivage des paramètres. Bien sûr la pénalisation Lasso engendre des solutions creuses mais il n'est pas sûr que l'espace mémoire des paramètres mis à 0 soit bien économisé et il ne l'est de toute façon pas an début d'exécution.

Par ailleurs, les deux algorithmes d'optimisation (*LBGFS et Liblinear*) proposés pour estimer une régression logistique ont été comparés sans mettre en évidence de différence ; les résultats ne sont pas présentés.

la figure 3.10 compare les temps d'exécution pour l'estimation des modèles en fonction de la taille de l'échantillon. Le comportement est assez identique à celui de l'étape (vectorisation) précédente. L'architecture intégrée s'avère plus performante à condition de disposer de suffisamment de mémoire.

Les figures 3.11 et 3.12 représentent les évolutions du taux d'erreur et du temps d'exécution de l'estimation des modèles en fonction de la taille de l'échantillon d'apprentissage et ce pour plusieurs valeurs du paramètre `n_hash`. La croissance du temps est plutôt linéaire avec la taille de l'échantillon tandis que ce temps est assez insensible au nombre de variables (`n_hash`) pour *Scikit-learn* contrairement à *MLlib*.

Le taux d'erreur décroit avec la taille de l'échantillon d'apprentissage mais ne semble pas encore stabilisée avec 15M de produits. Il décroît également avec le nombre de variables mais se stabilise au delà de 60 000. Sans explication claire, les régressions logistiques estimées par *MLlib* stagnent à des taux d'erreur supérieurs à celles estimées par *Scikit-learn* même en limitant ces dernières au même nombre de variables. Enfin ajouter plus de variables (75k) pour *Spark MLlib* provoquent des erreurs d'exécution pour défaut de mémoire, contrairement à Python *Scikit-learn* qui accepte des valeurs de `n_hash` nettement plus grandes (1M) en précisant que cette fois (*swapping*) les temps d'exécution deviennent nettement plus importants.

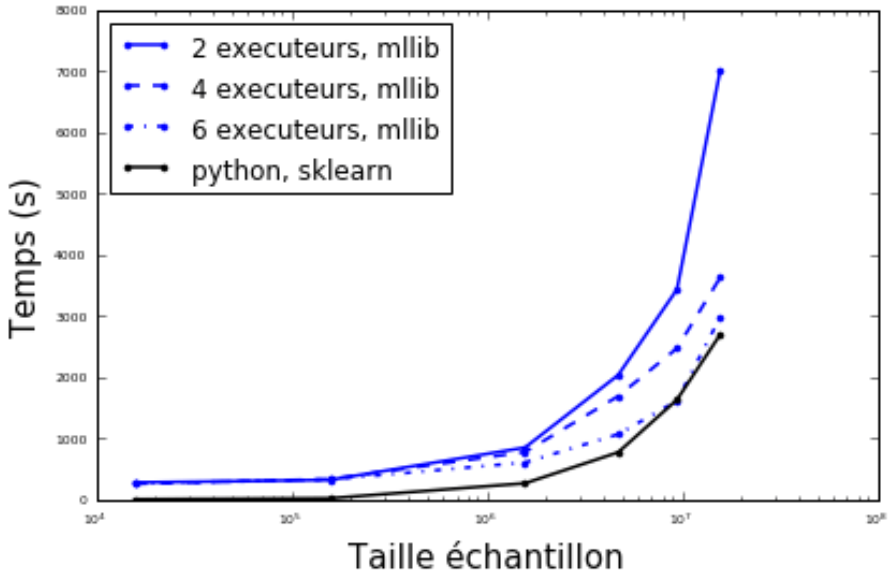


FIGURE 3.10 – *Cdiscount : apprentissage. Évolution du temps d'apprentissage pour plusieurs versions du cluster Spark avec 2, 4 ou 6 exécuteurs et en Python (Scikit-learn) avec* `n_hash` *= 60 000 variables.*

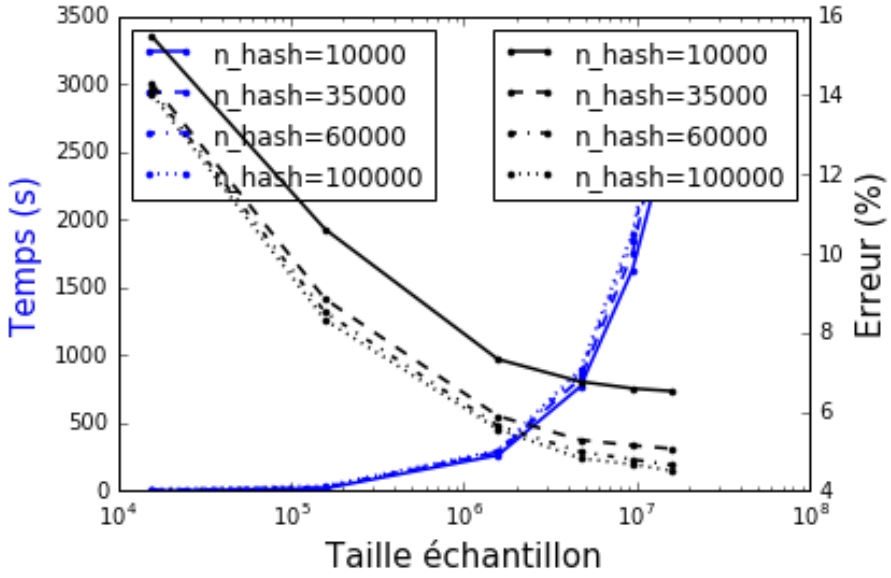


FIGURE 3.11 – *Cdiscount. Évolution du temps d'exécution de l'apprentissage avec Scikit-learn et du taux d'erreur sur l'échantillon test en fonction de la taille de l'échantillon et du nombre* `n_hash` *de variables ou mots du dictionnaire.*

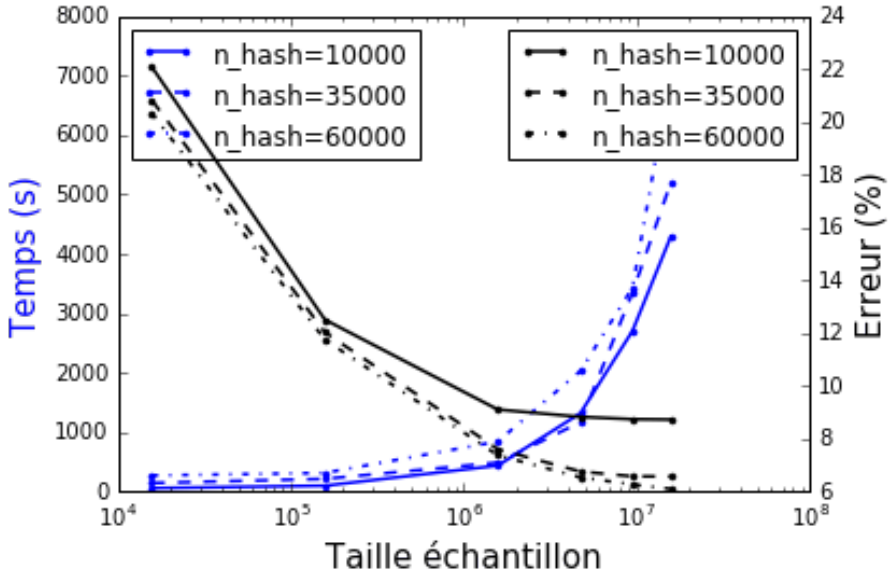


FIGURE 3.12 – *Cdiscount. Évolution du temps d'exécution de l'apprentissage avec Spark MLlib et du taux d'erreur sur l'échantillon test en fonction de la taille de l'échantillon et du nombre* `n_hash` *de variables ou mots du dictionnaire.*

En résumé, pour cette étape de modélisation par un ensemble de régressions logistiques, l'architecture intégrée se montre plus efficace en temps et en précision que l'architecture distribuée qui nécessite, globalement, des ressources mémoires importantes pour éviter des erreurs à l'exécution.

3.6 Conclusion

Finalement la comparaison s'établit principalement entre deux types d'architecture de parallélisation pour l'analyse de données massives ; celle *intégrée* du calcul à haute performance avec un serveur opérant de nombreux processeurs et beaucoup de mémoire (RAM) ou celle des données et calculs *distribuées* sur des nœuds physiquement distincts. Pour ajouter à la confusion, les architectures "distribuées" testées le furent virtuellement avec la définition de *nœuds virtuels* au sein d'une architecture intégrée. C'est d'expérience très souvent le cas et ce qui est alors recherchée est plus la capacité de *Spark* à gérer des données hétérogènes, qu'une réelle parallélisation des entrées / sorties de données excessivement massives.

La mise en œuvre de solutions *Hadoop Spark* ajoute une couche de complexité supplémentaire. Cet investissement n'est à tenter que mûrement réfléchi et justifié. Les comparaisons réalisées sur ces trois cas d'usage amènent quelques réflexions dont la validité

reste limitée dans le temps avec le niveau de maturité de ces technologies et même avec leur simple existence. Ainsi *Spark* dont une diffusion stable n'est disponible que depuis 2014 est en plein développement ; la version 2 est annoncée pour septembre 2016 de même que la version stable 0.18 de *Scikit-learn*.

La pression académique de publication conduit à la production de méthodes ou variantes de méthodes dont les performances ne sont montrées que sur des exemples sélectionnés. Parallèlement, la pression commerciale engendre des battages médiatiques mettant exagérément en valeur les avantages de nouvelles technologies. Même limitées et sans doute assez naïves, les quelques expérimentations conduites ici permettent de relativiser certains jugements.

- Pour la gestion (*munging*) de volumes et/ou de flux de données importants, complexes, hétérogènes, la parallélisation efficace de traitements élémentaires, les fonctionnalités déjà opérationnelles de *SparkSQL*, *streaming, pipeline* sont des atouts à prendre en compte.

- En revanche, le recours à *MLlib* ou *SparkML* pour la phase d'apprentissage ou de modélisation (TF-IDF incluse) n'est pas, en l'état du développement de ces librairies, une priorité. Les librairies de Python (*Scikit-learn*), R, éventuellement celles en développement de Julia, répondent aux besoins en utilisant une machine avec suffisamment de mémoire et de processeurs. La distribution des algorithmes (*MapReduce*) sur plusieurs machines soulève plus de problèmes, notamment de gestion de la mémoire, qu'elle n'en résout pour des algorithmes complexes.

- Il ne faut pas perdre de vue que des *big data* peuvent engendrer des *big* modèles de régression, de forêts aléatoires... sans même aller jusqu'à la complexité des solutions gagnantes de concours *Kaggle*. Il ne suffit plus de gérer les données, il faut aussi mémoriser ces modèles lors de leur estimation puis en exploitation. Une architecture intégrée de calcul haute performance (HPC) semble plus adaptée à l'utilisation de modèles complexes qu'une architecture distribuée.

- Néanmoins, utiliser *SparkML, MLlib* en *modélisation* si la mise en œuvre d'un algorithme distribué reste simple, sans risque de dépassement mémoire ; c'est le cas des problèmes de recommandation par complétion de matrice. L'algorithme de factorisation non négative (NMF) par moindre carrés alternés (ALS) *MLlib* se montre simple et efficace pour gérer des grandes matrices creuses et produire des matrices de facteurs latents de faible rang donc de volume accessible.

- Ne pas perdre de vue que les comparaisons de cet article se limitent aux méthodes implémentées dans les libraires librement accessibles les plus en vogue. Il y a un fossé avec des développements académiques récents ([Genuer et al., 2015]) pas ou pas encore implémentés dans ces librairies. Le temps et une forme de sélection naturelle fera le tri des approches les plus efficaces afin de combler ce fossé.

- Entre R et Python, le choix est assez facile et dépend de l'objectif, de la complexité des flots de traitement. Schématiquement, R propose beaucoup plus d'outils et de méthodes permettant de comprendre, interpréter, donner du sens aux modèles. En revanche Python se montre plus efficace dans les phases de traitements et exploitations, pour la prévision brute. Les deux environnements sont, en l'état actuel, plutôt complémentaires.

En définitive, les conclusions de ces expérimentations ne présentent rien d'extraordinaire ; elles sont fidèles à une forme de bon sens et correspondent aux pratiques industrielles présentées dans différents forums et conférences (Cdiscount, Critéo, Deepky, Tinyclues...) ; il est néanmoins utile de le vérifier concrètement, indépendamment de la pression commerciale des éditeurs de logiciels ou de celle, de publication, des chercheurs académiques. Une architecture distribuée (*Hadoop Spark*) est adaptée à la gestion de gros volumes et flux de données en ligne, alors que la modélisation ou l'apprentissage peut se faire hors ligne sur des sous-ensembles des données avec une architecture parallèle et une mémoire importante intégrée. Les besoins en calcul priment alors sur les entrées / sorties. D'autres approches d'apprentissage en ligne, notamment par des algorithmes de gradient stochastique, soulèvent d'autres questions d'indépendance du flux de données.

Il est nécessaire de rappeler que la qualité de la *Science des Données* dépend directement de leur fiabilité (*garbage in garbage out*), de leur représentativité. L'exemple des données MNIST montre bien qu'au delà d'un seuil, la taille des données n'améliore pas la précision. En revanche, la *construction d'une base d'apprentissage* en insistant éventuellement sur les zones de l'espace plus difficiles à prédire, tout en allégeant celles plus faciles, reste un défi majeur à relever dans chaque cas étudié. Enfin ce travail n'aborde pas les algorithmes d'apprentissage profond nécessitant des moyens de calcul et bases d'apprentissage d'une autre nature pour faire le choix, par exemple en analyse d'images, entre construction de variables caractéristiques (*features*) discriminantes puis apprentissage classique et analyse brutes des images ou signaux par *deep learning*.

Conclusion

This thesis deals with the application of machine learning methods in two particular situations.

First we tackled the problem of applying prediction methods to road traffic data. The main difficulty we encountered during this undertaking, was in dealing with the particular structure of the data. Indeed, the observed locations of a vehicle trip are all of the same dimension. They are then correctly fitted to be used as input to a machine learning algorithm. However these locations can be interpreted only if we consider their time aspect, *i.e.*, a succession of locations from the same vehicle which form a trajectory. Trajectory objects are of different sizes and can not be used directly for machine learning algorithms. We overcame this problem by defining a model enabling us to give the data a new structure in order to capture user behaviour and to develop prediction methods based on this structure.

The model we developed followed a two step procedure. It first models the main paths taken by users by applying hierarchical clustering on a new distance between trajectories (Symmetrized Segment-Path Distance). Then, it models main traffic flow patterns within each cluster of trajectories by a mixture of 2d-Gaussian distributions. This yields a data driven grid of locations thereby enabling the structuring of the trajectory as a succession of states.

The advantages of this model are numerous. The characteristics of the Gaussian distribution enable quick evaluation of new trajectory locations belonging to different states of the model. Hence it enables quick conversion of a trajectory to its new state-structure and thus real-time evaluation. The new state representation also enables quick extraction of characteristics and features of the trajectory in order to compare a trajectory to others and to develop predictive applications. We also developed this model in order for it to be entirely data driven. It does not require any additional parameters to function properly. Therefore, it can be applied on different datasets, with different characteristics.

We have elaborated three methods based on this model which enable the prediction of the final destination, the arrival time, and the next location of a vehicle during its trip. For each of theses methods, we provide the methodology and the experimental results on two datasets with different characteristics, with special attention paid to the properties of the model to enable easy extraction of the desired features required for prediction. We

also emphasised that these applications were developed in order to perform predictions for any trajectory at any given completion time. We paid special attention to this aspect in order to avoid over-fitting on a dataset taken at a given timestamp.

We also tackled the problem of choosing the right technology when applying machine learning algorithms on massive volumes of data. For that, we compare three different use cases concerning different datasets and applied different machine learning algorithms to each. The tests performed on these three datasets lead to the same conclusions. The choice of using the *Spark* cluster computing framework for machine learning applications should be undertaken with caution. The machine learning library, *MLlib* and *SparkML*, are currently underdeveloped though some functionalities are fully operational, especially for data management and munging. The python library, *Scikit-learn*, and some of the R libraries are preferable. Their implementations lead to better results in terms of the quality of prediction and sometimes in terms of the time required for learning. In the case of our method, the preprocessing step of converting the set of locations to their trajectory forms, as well as the computation of the distance between trajectories for example, can take advantage of the *Spark* framework, because it implies simple calculations and that are easily parallelizable. However, for the following steps, and especially for predictive application, the use of the *Spark* framework is not recommended according to the conclusions drawn from the Chapter 3. Moreover, we conclude that continuously augmenting the amount of data in the training dataset does not indefinitely increase the quality of precision. In the three use cases we studied, we observed a threshold beyond which adding more data not only improved very slightly the quality of the precision but also increased considerably the computation time required for the learning step.

The amount of data readily available today yields more information and enables the development of new applications and solutions, but the correct study and analysis of the data is a necessary precursor step before applying predictive methods. We have demonstrated in the case of road traffic, that we can achieve good results with a reasonable amount of data, as long as the data are well structured.

Bibliographie

Helmut Alt and Michael Godau. Computing the fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5(01n02) : 75–91, 1995.

Gennady Andrienko, Natalia Andrienko, and Stefan Wrobel. Visual analytics tools for analysis of movement data. *ACM SIGKDD Explorations Newsletter*, 9(2) :38–46, 2007.

Mihael Ankerst, Markus M Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics : ordering points to identify the clustering structure. In *ACM Sigmod Record*, volume 28(2), pages 49–60. ACM, 1999.

Daniel Ashbrook and Thad Starner. Using gps to learn significant locations and predict movement across multiple users. *Personal and Ubiquitous Computing*, 7(5) :275–286, 2003.

Leonard E Baum and Ted Petrie. Statistical inference for probabilistic functions of finite state markov chains. *The annals of mathematical statistics*, 37(6) :1554–1563, 1966.

Donald J Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *KDD workshop*, volume 10(16), pages 359–370. Seattle, WA, 1994.

Michael W. Berry, Murray Browne, Amy N. Langville, V. Paul Pauca, and Robert J. Plemmons. Algorithms and applications for approximate nonnegative matrix factorization. *Computational Statistics & Data Analysis*, 52(1) :155 – 173, 2007.

P. Besse and B. Laurent. De statisticien à data scientist – développements pédagogiques à l'insa de toulouse. *Statistique et Enseignement*, 7(1) :75–93, 2016a.

P. C. Besse, B. Guillouet, J. M. Loubes, and F. Royer. Review and perspective for distance-based clustering of vehicle trajectories. *IEEE Transactions on Intelligent Transportation Systems*, PP(99) :1–12, 2016. ISSN 1524-9050. doi : 10.1109/TITS.2016.2547641.

Philippe Besse and Béatrice Laurent. De statisticien à data scientist–développements pédagogiques à l'insa de toulouse. *Statistique et Enseignement*, 7(1) :75–93, 2016b.

Loubes J.-M. Besse P., Guillouet B. Apprentissage sur données massives, trois cas d'usage avec r, pyhton, spark. In *Apprentissage Statistique et Données Massives*. Technip, 2016.

G. Biau, A. Ficher, B. Guedj, and J.-D. Malley. Cobra : A nonlinear aggregation strategy. *Journal of Multivariate Analysis*, 146 :18–28, 2016.

P. Bickel and H. Chernoff. Asymptotic distribution of the likelihood ratio statistic in a prototypical non regular problem. In J.K. Ghosh, editor, *Statistics and probability*, pages 83–96. Wiley Eastern Limited, 1993.

Tolga Bozkaya, Nasser Yazdani, and Meral Özsoyoğlu. Matching and indexing sequences of different lengths. In *Proceedings of the sixth international conference on Information and knowledge management*, pages 128–135. ACM, 1997.

L. Breiman. Random forests. *Machine Learning*, 45 :5–32, 2001.

Joan Bruna and S. Mallat. Invariant scattering convolution networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8) :1872–1886, 2013. ISSN 0162-8828.

E.J. Candes and T. Tao. The power of convex relaxation : Near-optimal matrix completion. *Information Theory, IEEE Transactions on*, 56(5) :2053–2080, 2010.

Lei Chen and Raymond Ng. On the marriage of lp-norms and edit distance. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 792–803. VLDB Endowment, 2004.

Lei Chen, M Tamer Özsu, and Vincent Oria. Robust and fast similarity search for moving object trajectories. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 491–502. ACM, 2005.

Patrick Pakyan Choi and Martial Hebert. Learning and predicting moving object trajectory : a piecewise trajectory segment approach. *Robotics Institute*, page 337, 2006.

Darya Chudova, Scott Gaffney, Eric Mjolsness, and Padhraic Smyth. Translation-invariant mixture models for curve clustering. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 79–88. ACM, 2003.

Alexandre de Brébisson, Étienne Simon, Alex Auvolat, Pascal Vincent, and Yoshua Bengio. Artificial neural networks applied to taxi destination prediction. *arXiv preprint arXiv :1508.00021*, 2015.

Michel Marie Deza and Elena Deza. *Encyclopedia of distances*. Springer, 2009.

David Donoho. John w. tukey 100th birthday celebration at princeton university. In *John W. Tukey 100th Birthday Celebration at Princeton University*, 2015.

J.-J. Droesbeke, J. Fine, and G. Saporta. *Méthodes bayesiennes en statistique*. Paris, Technip, 2002.

Thomas Eiter and Heikki Mannila. Computing discrete fréchet distance. Technical report, Citeseer, 1994.

Mohamed Khalil El Mahrsi and Fabrice Rossi. Graph-based approaches to clustering network-constrained trajectory data. In *NFMCP*, pages 124–137. Springer, 2012.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96(34), pages 226–231, 1996.

E. Forgy. Cluster analysis of multivariate data : Efficiency versus interpretability of classification. *Biometrics*, 21(3) :768–769, 1965.

M Maurice Fréchet. Sur quelques points du calcul fonctionnel. *Rendiconti del Circolo Matematico di Palermo (1884-1940)*, 22(1) :1–72, 1906.

Brendan J Frey and Delbert Dueck. Clustering by passing messages between data points. *science*, 315(5814) :972–976, 2007.

Jerome Friedman. Data mining and statistics : What is the connection ? In *Proceedings of the 29th Symposium on the Interface Between Computer Science and Statistics*, 1997.

Scott Gaffney and Padhraic Smyth. Trajectory clustering with mixtures of regression models. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 63–72. ACM, 1999.

Scott John Gaffney. *Probabilistic curve-aligned clustering and prediction with regression mixture models*. PhD thesis, University of California, Irvine, 2004.

Sébastien Gambs, Marc-Olivier Killijian, and Miguel Núñez del Prado Cortez. Next place prediction using mobility markov chains. In *Proceedings of the First Workshop on Measurement, Privacy, and Mobility*, page 3. ACM, 2012.

Maxime Gariel, Ashok N Srivastava, and Eric Feron. Trajectory clustering and an application to airspace monitoring. *Intelligent Transportation Systems, IEEE Transactions on*, 12(4) :1511–1524, 2011.

Renaud Gaujoux and Cathal Seoighe. A flexible r package for nonnegative matrix factorization. *BMC Bioinformatics*, 11(1) :367, 2010.

R. Genuer, J.-M. Poggi, C. Tuleau-Malot, and N. Villa-Vialaneix. Random Forests for Big Data. *ArXiv e-prints*, 2015.

Fosca Giannotti and Dino Pedreschi. *Mobility, data mining and privacy : Geographic knowledge discovery*. Springer Science & Business Media, 2008.

B. Goutorbe, Y. Jiao, M. Cornec, C. Grauer, and Jakubowicz J. A large e-commerce data set released to benchmark categorization methods. In *Journées de Statistique de Montpellier*. Société Française de Statistique, 2016.

Binh Han, Ling Liu, and Edward Omiecinski. Neat : Road network aware trajectory clustering. In *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*, pages 142–151. IEEE, 2012.

Binh Han, Ling Liu, and Edward Omiecinski. Road-network aware trajectory clustering : Integrating locality, flow, and density. *Mobile Computing, IEEE Transactions on*, 14 (2) :416–429, 2015.

Trevor Hastie, Rahul Mazumder, Jason D. Lee, and Reza Zadeh. Matrix completion and low-rank svd via fast alternating least squares. *J. Mach. Learn. Res.*, 16(1) :3367–3402, January 2015. ISSN 1532-4435.

Felix Hausdorff. Grundz uge der mengenlehre, 1914.

Weiming Hu, Xuejuan Xiao, Zhouyu Fu, Dan Xie, Tieniu Tan, and Steve Maybank. A system for learning statistical motion patterns. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(9) :1450–1464, 2006.

Jung-Rae Hwang, Hye-Young Kang, and Ki-Joune Li. *Spatio-temporal similarity analysis between trajectories on road networks*. Springer, 2005a.

San-Yih Hwang, Ying-Han Liu, Jeng-Kuen Chiu, and Ee-Peng Lim. Mining mobile group patterns : A trajectory-based approach. In *Advances in knowledge discovery and data mining*, pages 713–718. Springer, 2005b.

Chen Jin, Ruoqian Liu, Zhengzhang Chen, William Hendrix, Ankit Agrawal, and Alok Choudhary. A scalable hierarchical clustering algorithm using spark. In *Big Data Computing Service and Applications (BigDataService), 2015 IEEE First International Conference on*, pages 418–426. IEEE, 2015.

A Jonathan, S Sclaroff, G Kollios, and V Pavlovic. Discovering clusters in motion time-series data. In *Computer Vision and Pattern Recognition (CVPR)*, 2003.

kaggle. data set ecml/pkdd 15 : Taxi trajectory prediction (1). Downloaded from https ://www.kaggle.com/c/pkdd-15-predict-taxi-service-trajectory-i/data, April 2015.

Panos Kalnis, Nikos Mamoulis, and Spiridon Bakiras. On discovering moving clusters in spatio-temporal data. In *Advances in spatial and temporal databases*, pages 364–381. Springer, 2005.

Leonard Kaufman and Peter Rousseeuw. *Clustering by means of medoids*. North-Holland, 1987.

Ahmed Kharrat, Iulian Sandu Popa, Karine Zeitouni, and Sami Faiz. Clustering algorithm for network constraint trajectories. In *Headway in Spatial Data Handling*, pages 631–647. Springer, 2008.

Byoungjip Kim, Jin-Young Ha, SangJeong Lee, Seungwoo Kang, Youngki Lee, Yunseok Rhee, Lama Nachman, and Junehwa Song. Adnext : a visit-pattern-aware mobile advertising system for urban commercial complexes. In *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications*, pages 7–12. ACM, 2011.

Jiwon Kim and Hani S Mahmassani. Spatial and temporal characterization of travel patterns in a traffic network using vehicle trajectories. *Transportation Research Part C : Emerging Technologies*, 59 :375–390, 2015.

A. Kleiner, A. Talwalkar, P. Sarkar, and M. Jordan. A scalable bootstrap for massive data. *Journal of the Royal Statistical Society : Series B (Statistical Methodology)*, 76 (4) :795–816, 2014.

Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems, 2009.

John Krumm and Eric Horvitz. Predestination : Inferring destinations from partial trajectories. In *UbiComp 2006 : Ubiquitous Computing*, pages 243–260. Springer, 2006.

Eckhard Kruse, Ralf Gutsche, and Friedrich M Wahl. Acquisition of statistical motion patterns in dynamic environments and their application to mobile robot motion planning. In *Intelligent Robots and Systems, 1997. IROS'97., Proceedings of the 1997 IEEE/RSJ International Conference on*, volume 2, pages 712–717. IEEE, 1997.

Max Kuhn. Building predictive models in r using the caret package. *Journal of Statistical Software*, 28(5), 2008.

Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

Chen-Yu Lee, Patrick W. Gallagher, and Zhuowen Tu. Generalizing pooling functions in convolutional neural networks : Mixed, gated, and tree, 2016.

D. Lee and S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 1999.

Jae-Gil Lee, Jiawei Han, and Kyu-Young Whang. Trajectory clustering : a partition-and-group framework. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 593–604. ACM, 2007.

Seok-Lyong Lee, Seok-Ju Chun, Deok-Hwan Kim, Ju-Hong Lee, and Chin-Wan Chung. Similarity search for multidimensional data sequences. In *Data Engineering, 2000. Proceedings. 16th International Conference on*, pages 599–608. IEEE, 2000.

Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *R News*, 2(3) :18–22, 2002.

M. Lichman. UCI machine learning repository, 2013.

Bin Lin and Jianwen Su. Shapes based trajectory queries for moving objects. In *Proceedings of the 13th annual ACM international workshop on Geographic information systems*, pages 21–30. ACM, 2005.

Rake& Agrawal King-lp Lin and Harpreet S Sawhney Kyuseok Shim. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In *Proceeding of the 21th International Conference on Very Large Data Bases*, pages 490–501. Citeseer, 1995.

James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1(14), pages 281–297. Oakland, CA, USA., 1967.

R. Mazumder, T. Hastie, and R. Tibshirani. Spectral regularization algorithms for learning large incomplete matrices. *Journal of Machine Learning Research*, 11 :2287–2322, 2010.

Anna Monreale, Fabio Pinelli, Roberto Trasarti, and Fosca Giannotti. Wherenext : a location predictor on trajectory pattern mining. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 637–646. ACM, 2009.

Klaus-Robert Müller, Sebastian Mika, Gunnar Rätsch, Koji Tsuda, and Bernhard Schölkopf. An introduction to kernel-based learning algorithms. *IEEE TRANSACTIONS ON NEURAL NETWORKS*, 12(2) :181–201, 2001.

Mirco Nanni and Dino Pedreschi. Time-focused clustering of trajectories of moving objects. *Journal of Intelligent Information Systems*, 27(3) :267–289, 2006.

Raymond T Ng and Jiawei Han. Clarans : A method for clustering objects for spatial data mining. *Knowledge and Data Engineering, IEEE Transactions on*, 14(5) :1003–1016, 2002.

Biswanath Panda, Joshua Herbach, Sugato Basu, and Roberto Bayardo. Planet : Massively parallel learning of tree ensembles with mapreduce. In *Proceedings of the 35th International Conference on Very Large Data Bases (VLDB-2009)*, 2009.

Donald J Patterson, Lin Liao, Dieter Fox, and Henry Kautz. Inferring high-level behavior from low-level sensors. In *UbiComp 2003 : Ubiquitous Computing*, pages 73–89. Springer, 2003.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn : Machine learning in Python. *Journal of Machine Learning Research*, 12 :2825–2830, 2011.

N. Pentreath. *Machine Learning with Spark*. Packt puplishing, 2015.

C-S Perng, Huifang Wang, Sylvia R Zhang, and D Stott Parker. Landmarks : a new model for similarity-based pattern querying in time series databases. In *Data Engineering, 2000. Proceedings. 16th International Conference on*, pages 33–42. IEEE, 2000.

Michal Piorkowski, Natasa Sarafijanovic-Djukic, and Matthias Grossglauser. CRAWDAD data set epfl/mobility (v. 2009-02-24). Downloaded from http ://crawdad.org/epfl/mobility/, February 2009.

R Core Team. *R : A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2016a.

R Core Team. *R : A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2016b.

Salvatore Rinzivillo, Dino Pedreschi, Mirco Nanni, Fosca Giannotti, Natalia Andrienko, and Gennady Andrienko. Visually driven analysis of movement data by progressive clustering. *Information Visualization*, 7(3-4) :225–239, 2008.

Gook-Pil Roh and Seung-won Hwang. Nncluster : An efficient clustering algorithm for road network trajectories. In *Database Systems for Advanced Applications*, pages 47–61. Springer, 2010.

Peter J Rousseeuw. *Finding groups in data : An introduction to cluster analysis*. John Wiley & Sons, 2009.

G. Saporta. The utilization of multiple measurements in problems of biological classification. *Journal of the Royal Statistical Society, B*, 10 :159–203, 1789.

Choon-Bo Shim and Jae-Woo Chang. Similar sub-trajectory retrieval for moving objects in spatio-temporal databases. In *Advances in Databases and Information Systems*, pages 308–322. Springer, 2003.

Patrice Simard, Yann Le Cun, John Denker, and Bernard Victorri. Transformation invariance in pattern recognition - tangent distance and tangent propagation. In *Lecture Notes in Computer Science*, pages 239–274. Springer, 1998.

Reid Simmons, Brett Browning, Yilu Zhang, and Varsha Sadekar. Learning to predict driver route and destination intent. In *Intelligent Transportation Systems Conference, 2006. ITSC'06. IEEE*, pages 127–132. IEEE, 2006.

Nathan Srebro, Jason D. M. Rennie, and Tommi S. Jaakola. Maximum-margin matrix factorization. In *Advances in Neural Information Processing Systems 17*, pages 1329–1336. MIT Press, 2005.

Anuj Srivastava, Eric Klassen, Shantanu H Joshi, and Ian H Jermyn. Shape analysis of elastic curves in euclidean spaces. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(7) :1415–1428, 2011a.

Anuj Srivastava, Wei Wu, Sebastian Kurtek, Eric Klassen, and JS Marron. Registration of functional data using fisher-rao metric. *arXiv preprint arXiv :1103.3817*, 2011b.

C. H. Tang, A. C. Huang, M. F. Tsai, and W. J. Wang. An efficient distributed hierarchical-clustering algorithm for large scale data. In *Computer Symposium (ICS), 2010 International*, pages 869–874, Dec 2010. doi : 10.1109/COMPSYM.2010.5685388.

Eleftherios Tiakas, AN Papadopoulos, Alexandros Nanopoulos, Yannis Manolopoulos, Dragan Stojanovic, and Slobodanka Djordjevic-Kajan. Searching for similar trajectories in spatial networks. *Journal of Systems and Software*, 82(5) :772–788, 2009.

Dalia Tiesyte and Christian S Jensen. Similarity-based prediction of travel times for vehicles traveling on known routes. In *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, page 14. ACM, 2008.

Štěpán Urban, Michal Jakob, and Michal Pěchouček. Probabilistic modeling of mobile agents' trajectories. In *Agents and Data Mining Interaction*, pages 59–70. Springer, 2010.

M. J. van der Laan, E. C. Polley, and A. E. Hubbard. Super learner. *Statistical Applications in Genetics and Molecular Biology*, 6 :1, 2007.

Guido Van Rossum and Fred L Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

Dizan Vasquez and Thierry Fraichard. Motion prediction for moving objects : a statistical approach. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 4, pages 3931–3936. IEEE, 2004.

Michail Vlachos, George Kollios, and Dimitrios Gunopulos. Discovering similar multidimensional trajectories. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 673–684. IEEE, 2002.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann L. Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In Sanjoy Dasgupta and David Mcallester, editors, *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, volume 28, pages 1058–1066. JMLR Workshop and Conference Proceedings, May 2013.

Shen Wang and Haimonti Dutta. Parable : a parallel random-partition based hierarchical clustering algorithm for the mapreduce framework. In *6th Annual Machine Learning Symposium at the New York Academy of Science (NYAS)*, 2011.

Yida Wang, Ee-Peng Lim, and San-Yih Hwang. On mining group patterns of mobile users. In *Database and Expert Systems Applications*, pages 287–296. Springer, 2003.

Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *In International Conference on Artificial Intelligence*, 2009.

Jürgen Wiest, Matthias Höffken, Ulrich Kresel, and Klaus Dietmayer. Probabilistic trajectory prediction with gaussian mixture models. In *Intelligent Vehicles Symposium (IV), 2012 IEEE*, pages 141–146. IEEE, 2012.

Martin Wright and Andreas Ziegler. ranger : A fast implementation of random forests for high dimensional data in c++ and r. *Journal of Statistical Software*, 2016. to appear.

Huey-Ru Wu, Mi-Yen Yeh, and Ming-Syan Chen. Profiling moving objects by dividing and clustering trajectories spatiotemporally. *Knowledge and Data Engineering, IEEE Transactions on*, 25(11) :2615–2628, 2013.

Andy Yuan Xue, Rui Zhang, Yu Zheng, Xing Xie, Jin Huang, and Zhenghua Xu. Destination prediction by sub-trajectory synthesis and privacy protection against such prediction. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 254–265. IEEE, 2013.

M. Zaharia, M. Chowdhury, D. Das, A. Dave, J. Ma, M. McCauly, S. Franklin, M. J.and Shenker, and I. Stoica. Resilient distributed datasets : A fault-tolerant abstraction for in-memory cluster computing. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 15–28. USENIX, 2012.

Brian D Ziebart, Andrew L Maas, Anind K Dey, and J Andrew Bagnell. Navigate like a cabbie : Probabilistic reasoning from observed context-aware behavior. In *Proceedings of the 10th international conference on Ubiquitous computing*, pages 322–331. ACM, 2008.

TITLE : Machine Learning : Application to Road Traffic as Structured Data and to Big Data

---

Abstract

This thesis focuses on machine learning techniques for application to big data. We first consider trajectories defined as sequences of geolocalized data. A hierarchical clustering is then applied on a new distance between trajectories (Symmetrized Segment-Path Distance) producing groups of trajectories which are then modeled with Gaussian mixture in order to describe individual movements. This modeling can be used in a generic way in order to resolve the following problems for road traffic : final destination, trip time or next location predictions. These examples show that our model can be applied to different traffic environments and that, once learned, can be applied to trajectories whose spatial and temporal characteristics are different. We also produce comparisons between different technologies which enable the application of machine learning methods on massive volumes of data.

KEYWORDS : Machine Learning, Clustering, Big Data, Road Traffic, Trajectory.

AUTEUR : Brendan GUILLOUET

TITRE : Apprentissage statistique : application au trafic routier à partir de données structurées et aux données massives.

DIRECTEUR DE THESE : Philippe BESSE & Jean-Michel LOUBES

LIEU ET DATE DE SOUTENANCE : Salle de conférence MIP ,Bâtiment 1R3, 1er étage, Université Paul Sabatier, 18 Novembre 2016.

Résumé

Cette thèse s'intéresse à l'apprentissage pour données massives. On considère en premier lieu, des trajectoires définies par des séquences de géolocalisations. Une nouvelle mesure de distance entre trajectoires (Symmetrized Segment-Path Distance) permet d'identifier par classification hiérarchique des groupes de trajectoires, modélisés ensuite par des mélanges gaussiens décrivant les déplacements par zones. Cette modélisation est utilisée de façon générique pour résoudre plusieurs types de problèmes liés aux trafic routier : prévision de la destination finale d'une trajectoire, temps d'arrivée à destination, prochaine zone de localisation. Les exemples analysés montrent que le modèle proposé s'applique à des environnements routiers différents et, qu'une fois appris, il s'applique à des trajectoires aux propriétés spatiales et temporelles différentes. En deuxième lieu, les environnements technologiques d'apprentissage pour données massives sont comparés sur des cas d'usage industriels.

MOTS-CLES : Apprentissage, Classification non supervisée, Données Massives Trafic Routier, Trajectoire.

DISCIPLINE ADMINISTRATIVE : MITT : Domaine Mathématiques : Mathématiques appliquées

---

INTITULE ET ADRESSE DU LABORATOIRE : Université Toulouse 3 Paul Sabatier. Institut de Mathématiques de Toulouse (UMR 5219) 18 Route de Narbonne, 31400 Toulouse