



RAPPORT DE STAGE D'APPLICATION DE 2^E ANNÉE

Étude de la topologie des villes à l'aide de kernels sur les graphes

Stagiaire :
Hugo DUPRÉ
Étudiant en 3^e année
ENSAE ParisTech

Maître de stage :
Fabien PFAENDER
Chercheur et coordinateur
ComplexCity Lab

Shanghai, Chine
6 Juin 2016 - 12 Août 2016

Table des matières

Liste des figures et tableaux	2
Remerciements	2
1 Introduction	5
1.1 Contexte général	5
1.1.1 UTSEUS	5
1.1.2 ComplexCity Lab	5
1.1.3 Conditions de mon stage	5
1.2 Sujet du stage : la topologie des villes	5
1.2.1 Point de vue des experts du domaine urbain	5
1.2.2 Point de vue d'un data scientist	6
2 Préliminaires	6
2.1 Rappels sur les graphes	6
2.2 Récupération des données	6
2.3 Statistiques descriptives et échantillonnage	8
2.3.1 Statistiques descriptives	8
2.3.2 Échantillonnage	9
3 Méthodologie	9
3.1 Choix de la méthodologie	9
3.2 Rappels sur les kernels	10
4 Random walk kernel	10
4.1 Définition	10
4.2 Problèmes : tottering et temps de calcul	10
5 Shortest path kernel	11
5.1 Définition du kernel et approche intuitive	11
5.2 Normalisation du kernel	11
5.3 Apprentissage non-supervisé sur les villes françaises	12
5.3.1 Motivation	12
5.3.2 Catégorisation avec l'agglomerative hierarchical clustering	12
5.3.3 Interprétation des clusters	14
5.3.4 Visualisation des clusters : ACP kernelisée	14
5.3.5 Utilisation des informations supplémentaires sur les graphes	15
5.4 Conclusions	15
6 k-Graphlet kernels	16
6.1 Introduction sur les k-graphlet kernels	16
6.2 3,4,5-graphlets kernel	16
6.2.1 Définition	16
6.2.2 Une complexité trop grande	16
6.3 3,4,5-graphlets kernel avec échantillonnage	17
6.3.1 Définition	17
6.3.2 Limites	17
6.4 Alternative : all connected 3,4,5-graphlets kernel	17
6.4.1 Définition	17
6.4.2 Apport personnel : all connected 3,4,5-graphlets kernel modifié	18
6.5 Apprentissage non supervisé sur les villes françaises	19
7 Application à un problème de classification binaire : France vs USA	20
7.1 Motivation	20
7.2 Statistiques descriptives des graphes des villes américaines	20
7.3 Résoudre un problème de classification avec une matrice de Gram	21
7.4 Support Vector Machine (SVM)	21
7.5 Création de D_{train} et D_{test}	22

7.6 Comparaison des kernels	23
8 Conclusion	24
8.1 Réponse à la problématique	24
8.2 Un pas vers une meilleure compréhension des villes?	24
9 Ouverture	25
9.1 Weisfeiler-Lehman kernel	25
9.1.1 Principe	25
9.1.2 Un cadre de travail à exploiter	25
9.2 Autres améliorations possibles	25
A Références	26
B Liens GitHub	27
C Annexes	28
C.1 Variété de dimension 3 issue de l'ACP kernélisée	28
C.2 Graphe de Malakoff (92)	29

Liste des figures et tableaux

Table des figures

1	Graphe pondéré non-orienté	6
2	Procédé de simplification des graphes	7
3	Simplification d'une ville très sinueuse : Saint-Paul de la Réunion.	8
4	Distributions des sommets et des edges	8
5	Dendrogramme du hierarchical clustering sur S , en utilisant le shortest-path kernel normalisé	13
6	Boxplots des tailles des villes dans les 3 catégories.	14
7	Visualisation des clusters à l'aide des axes de l'ACP kernelisée.	15
8	Les 11 différents graphlets d'ordre 4.	16
9	Deux premiers axes de l'ACP kernelisée	19
10	Distributions des sommets et des edges	21
11	Utilisation des kernels pour SVM	22
12	Courbes ROC obtenus avec les différents kernels	24
13	ACP kernelisée : 3 premiers axes.	28
14	Graphe de Malakoff (92) simplifié	29

Liste des tableaux

1	Type des voies en fonction de leur classe	15
2	Résultats de la classification avec les différents kernels	23

Remerciements

Je tiens à remercier les personnes qui m'ont permis, de près ou de loin, de passer un été formidable à Shanghai pour ce stage d'application de deuxième année. Je voudrais remercier spécialement :

- Fabien Pfaender, mon maître de stage, qui m'a bien aidé et s'est beaucoup intéressé à mon travail. Nous nous sommes mutuellement appris beaucoup de choses, donc je le remercie de m'avoir accueilli pour ce stage.
- Bruno Bachimont, pour avoir débloqué mon salaire, ce qui m'a permis de valider ma convention de stage.
- L'équipe pédagogique de l'ENSAE, notamment Pierre Alquier et Xavier Dupré, qui sont deux professeurs toujours disponibles et dont l'aide est très précieuse. Merci aussi à mon référent pédagogique, Guillaume Lecué.
- Les internautes de Stackoverflow.com, qui m'ont débloqué un nombre incalculable de fois, que ce soit sur le plan théorique ou pour les codes.
- Fériel Ben Miled, Alexia Bernard, Alexandre Barbusse et Poetini Lehartel, mes colocataires et collègues, étudiants à l'EIVP.
- Mes parents, Anne-Marie et François.

1 Introduction

1.1 Contexte général

Le stage a été réalisé au sein du laboratoire de recherche ComplexCity, à Shanghai, en République Populaire de Chine. Ce laboratoire est étroitement relié à l'Université de Technologie Sino-Européenne de l'Université de Shanghai (UTSEUS), qu'il convient donc de présenter.

1.1.1 UTSEUS

L'Université de Technologie Sino-Européenne de l'Université de Shanghai (UTSEUS) est une école d'ingénieur créée en 2005 par le réseau des Universités de Technologie (Belfort-Montbéliard, Compiègne, Troyes) françaises et par l'Université de Shanghai. Les locaux de l'UTSEUS se situent sur le campus de Baoshan de l'Université de Shanghai. Cette école est la première coopération universitaire sino-européenne. L'UTSEUS a pour objectif de former plus de mille étudiants chaque année, principalement des étudiants chinois amenés par la suite à réaliser une partie de leur cursus au sein de l'une des trois Universités de Technologie françaises. Elle forme également des étudiants de ces Universités de Technologie, dans le cadre d'un semestre d'échange à Shanghai. Génie informatique, science et génie des matériaux, génie mécanique et automatisation, génie biologique, science des données et machine learning sont les principaux enseignements proposés au sein des différents programmes de l'UTSEUS, toujours accompagnés de cours de langues intensifs.

1.1.2 ComplexCity Lab

En 2012, l'UTSEUS a créé un laboratoire de recherche mondial : il s'agit du laboratoire ComplexCity (ComplexCity Lab). Ce laboratoire a pour mission l'étude et la compréhension de la ville et des systèmes urbains. La mise en adéquation du milieu urbain avec ses habitants et l'amélioration de l'espace urbain deviennent de véritables enjeux. Afin de mieux comprendre les systèmes urbains, anticiper leurs mutations ou encore analyser de potentielles solutions durables, ce qui est crucial dans l'optique de l'amélioration de l'espace urbain, ComplexCity Lab développe une approche multi-disciplinaire. Celle-ci regroupe les compétences analytiques des sciences de l'ingénieur ainsi que l'ouverture et la créativité des sciences humaines et sociales. Elle tente de dépasser les approches classiques destinées à rendre la ville meilleure, notamment la notion de ville intelligente (smart city) qui met en avant le rôle important des nouvelles technologies du numérique et des sciences de l'information pour apporter une réponse aux problématiques urbaines. La capture massive et le traitement de données de sources et de formats très variés (données géographiques, réseaux sociaux, indicateurs de pollution), de même que l'analyse de ces données hétérogènes, spatiales, temporelles afin d'en extraire de la valeur, occupent tous deux une place primordiale. La manipulation des outils de la Data Science, de l'apprentissage et de l'informatique est donc presque toujours une nécessité afin de pouvoir produire des études de qualité. ComplexCity Lab regroupe aujourd'hui des chercheurs français et des chercheurs chinois, autour d'une multitude de projets différents. Une partie de l'équipe travaille à Shanghai dans une start-up space en plein centre-ville, et l'autre partie à Paris.

1.1.3 Conditions de mon stage

Mon stage s'est déroulé au sein des locaux chinois du laboratoire ComplexCity, dans une start-up space à l'angle de XinZha Lu et XiKang Lu, à Jing'An en pleine concession française. Celui-ci a duré dix semaines, du lundi 6 juin 2016 au vendredi 12 août 2016. M. Fabien Pfaender, chercheur et coordinateur au sein du laboratoire, était mon maître de stage. Il a été présent pendant une grande majorité de mon stage, et en plus d'avoir suivi mon travail, il s'y est investi. Une journée typique débutait vers 9h et se terminait à 18h, même si ces horaires pouvaient varier en fonction des besoins. J'avais la possibilité d'utiliser beaucoup d'équipements du laboratoire (ordinateurs, écrans, raspberries et même imprimante 3D).

1.2 Sujet du stage : la topologie des villes

1.2.1 Point de vue des experts du domaine urbain

La topologie des villes est un point central de l'étude des systèmes urbains. La topologie d'une ville consiste en son organisation spatiale. La façon dont on peut se déplacer en ville, par exemple, est intrinsèquement liée à la topologie de la ville. Comment organiser spatialement les villes durables de demain ? La réponse à cette question d'importance majeure passe par la compréhension de l'organisation spatiale des villes actuelles.

Les géographes et les urbanistes ont une méthodologie de travail bien différente d'un data scientist. Lorsqu'ils se lancent dans l'étude d'une ville, ou de la topologie des villes en général, cela mène souvent à une étude qualitative, qui s'appuie sur un nombre assez faible d'exemples. Leurs études sont le reflet de leur connaissances et de leur

expertise sur le domaine urbain, et sont menées dans le but d'infirmer ou de confirmer des hypothèses qui émergent de leur compréhension des villes. C'est une méthodologie scientifique tout-à-fait valable, mais elle est souvent que partiellement appuyée par des analyses de données et autres méthodes statistiques, qui peuvent être incomplètes ou pas assez poussées.

1.2.2 Point de vue d'un data scientist

Aujourd'hui, des méthodes statistiques récentes peuvent tout-à-fait venir aider ces scientifiques à trouver des résultats nouveaux et pertinents, afin d'aider à la compréhension des systèmes complexes que sont les villes. Cette compréhension des systèmes urbains est cruciale à notre époque où l'importance des villes est en constante augmentation, et où il est nécessaire de bien les comprendre pour pouvoir apporter des solutions aux problèmes dont les enjeux sont les plus importants du III^e millénaire : l'écologie et la démographie. En effet la prépondérance des villes n'est plus à établir : les villes couvrent seulement 2% de la surface de la Terre habitée mais comptent 50% de la population mondiale (chiffre qui pourrait atteindre 80% en 2050 selon l'ONU). Elles consomment près de 75% de la consommation d'énergie mondiale et représentent près de 80% des émissions mondiales de CO_2 . Les géographes et les urbanistes sont les scientifiques qui doivent comprendre les villes afin de nous permettre d'aller dans la bonne direction pour que ces villes se développent durablement. Ainsi les data scientists, scientifiques polyvalents et serviables, se doivent d'offrir leur expertise pour aider leurs collègues à construire les villes de demain. Un data scientist seul est inutile, mais il a le potentiel d'être une clé vers de nouvelles portes dans énormément de domaines, ici l'étude des villes.

Il est difficile pour les géographes et urbanistes d'étudier des collections de villes dans leur ensemble, en effet chaque ville possède déjà sa propre complexité, et comparer ces complexités entre elles se révèlent être un challenge de taille. Mais ces experts de la ville sont en veine ! La topologie des villes peut-être représentée par le réseau de ses voies de transport, et ce réseau peut-être modélisé par la fabuleux objet mathématique qu'est le graphe.

Étudier la topologie des villes, c'est donc étudier la topologie d'une collection de graphes. Mais pas n'importe quels graphes, des gros graphes. En effet la difficulté réside dans le fait que non seulement ces graphes ont des milliers, voir des dizaines de milliers de noeuds et arêtes, mais il en existe 36 000 rien que pour la France. C'est bien pour cela que les géographes et urbanistes se restreignent à étudier seulement quelques exemples, mais de manière approfondie. Le but de mon étude est donc le suivant : **"Comment étudier la topologie des villes en s'appuyant sur les graphes qu'elles forment ?"**

2 Préliminaires

2.1 Rappels sur les graphes

Un graphe G est défini par le couple (V, E) , où V est l'ensemble des noeuds (vertices en anglais) et E l'ensemble des arêtes (edges en anglais). Un graphe peut être orienté ou non-orienté. Dans un graphe orienté on précise le sens des arêtes, tandis que dans les graphes non-orientés on précise seulement si les noeuds sont connectés ou non. Les graphes peuvent aussi être pondérés, dans ce cas on associe un poids à chacune des arêtes. Le graphe ci-contre est un exemple de graphe pondéré non-orienté (Figure 1). Nous modéliserons les villes par des graphes non-orientés, où les arêtes représentent les voies tandis que les intersections sont modélisées par certains noeuds. Le reste des noeuds seront utilisés pour former les courbes. En effet, on a besoin de ces noeuds supplémentaires pour modéliser les courbes.

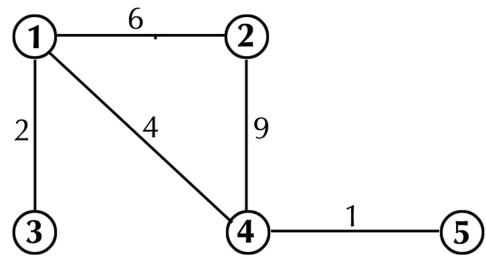


FIGURE 1: Graphe pondéré non-orienté

2.2 Récupération des données

Fabien Pfaender, mon maître de stage, m'a grandement aidé pour la récupération des données, comme prévu. Notre but est d'étudier la topologie des villes, et nous avons choisi d'étudier les villes françaises dans un premier temps. Dans un deuxième temps, une fois les premiers résultats obtenus, nous nous intéresserons aussi aux villes américaines, et nous les comparerons aux villes françaises. Pour ce faire, il faut récupérer les graphes des routes des villes françaises et américaines. Nous avons utilisé OpenStreetMap pour réaliser cette tâche. OpenStreetMap (OSM) est un projet qui a pour but de constituer une base de données géographiques libre du monde (permettant par exemple de créer des cartes sous licence libre), en utilisant le système GPS et d'autres données libres.

On a ici utilisé cet outil pour récupérer la carte des régions, qu'il a ensuite fallu découper en communes pour obtenir les cartes des communes. Ensuite, nous avons réalisé des conversions de format afin de récupérer seulement le graphe des communes. Pour finir, nous avons simplifié les graphes (nous reviendrons sur cette étape en détail). Le format final des graphes est .gexf, un format standard pour les graphes.

Pour résumer, la récupération des données consiste en les 6 étapes suivantes :

1. Obtenir le fichier des limites administratives des villes.
2. Obtenir les sources des régions OpenStreetMap (osm), c'est-à-dire le graphe des routes de France et USA découpés par régions.
3. Extraire des limites administratives des fichiers de contour ".poly".
4. À l'aide d'osmosis, découper les .poly dans le fichier osm de région pour obtenir un fichier pbf ou osm par ville.
5. Transformer le pbf en gexf à l'aide d'un script Python.
6. Simplifier le gexf.

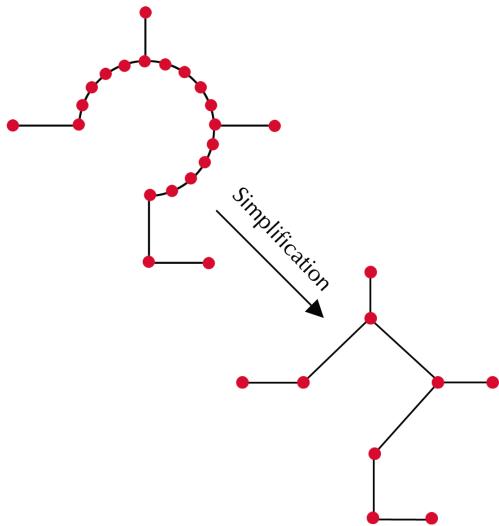


FIGURE 2: Procédé de simplification des graphes.

on peut le voir, la ville comporte énormément de chemins très sinueux, ce qui fait que sans simplification, son graphe possède énormément de noeuds et arêtes : 48 000 noeuds et 50 000 arêtes ! Après simplification, le graphe de la ville ne possède plus que 6 200 noeuds et 7 700 arêtes. On peut facilement voir à l'œil nu que la topologie a été conservée.

En annexe A (Figure 14), on peut aussi voir une représentation du graphe de Malakoff où apparaissent des informations supplémentaires. Sur OpenStreetMap, nous avons pu récupérer ces informations supplémentaires, comme le type des routes : une indicatrice qui vaut 1 si la route est à sens unique ou bien la taille des routes. Les couleurs sur les arêtes du graphe représentent les types des routes (pistes cyclables, autoroutes, nationales...). La couleur sur les noeuds représente le degré du noeud. Nous essayerons ensuite d'utiliser ces données supplémentaires pour affiner notre étude.

Au début, nous n'avions pas simplifié les graphes. Les graphes obtenus comportaient alors beaucoup de noeuds et d'arêtes *superflus*. En effet, la plupart des routes et chemins dans les villes comportent des virages. Il existe aussi beaucoup de chemins sinueux. Toutes ces routes et chemins sont formés par de très nombreux noeuds et arêtes. Cette information est superflue et gênante car elle augmente les temps de calcul sans ajouter aucune information pertinente au graphe. Nous avons donc procédé à la simplification des graphes, selon le procédé exposé en Figure 2. Ce procédé consiste à détecter toutes les chemins ne comportant que des noeuds de degrés 2, puis de supprimer tout les noeuds de ces chemins autre que le premier et le dernier. Bien évidemment, on fait attention aux cas particuliers, par exemple : si deux chemins composés seulement de noeuds de degrés 2 partent du même noeud et arrivent au même noeud, on transforme ces chemins en gardant un seul noeud de degré 2 par chemin.

Cette simplification nous a permis de réduire d'environ 30% la taille des graphes (en termes de nombre de noeuds et d'arêtes), tout en conservant la topologie de ceux-ci. À l'aide du logiciel Gephi, nous pouvons représenter les graphes. L'exemple suivant, Figure 3 montre la commune de Saint-Paul de La Réunion, avant et après simplification. Nous avons choisi cette ville en exemple car comme

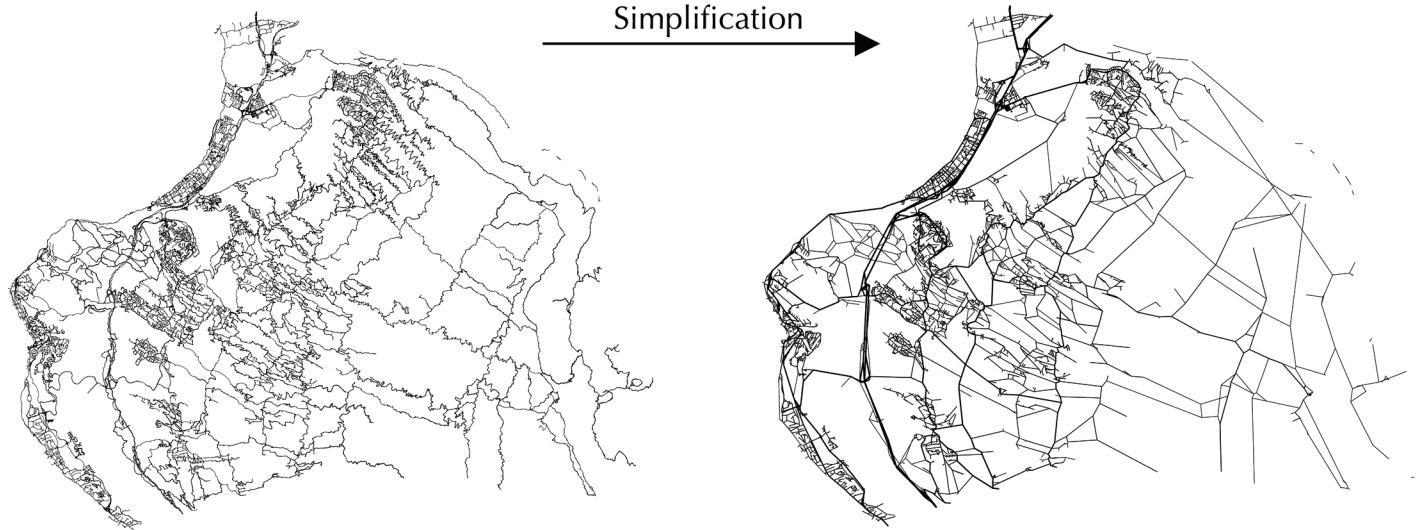


FIGURE 3: Simplification d'une ville très sinueuse : Saint-Paul de la Réunion.

2.3 Statistiques descriptives et échantillonnage

2.3.1 Statistiques descriptives

Il s'agit ici de décrire rapidement les graphes qui font l'objet de notre étude. Cela nous permettra de mieux comprendre et mieux appréhender les graphes des villes françaises. Cette étape est essentielle dans notre étude : comprendre à quel type de graphes nous avons affaire est important pour pouvoir capturer cette information *topologie* contenue dans les graphes.

Comme nous avons pu le voir auparavant, les graphes sont définis par leurs nœuds et leurs arêtes. La première étape naturelle est d'étudier la distribution du nombre de nœuds et d'arêtes dans la collection.

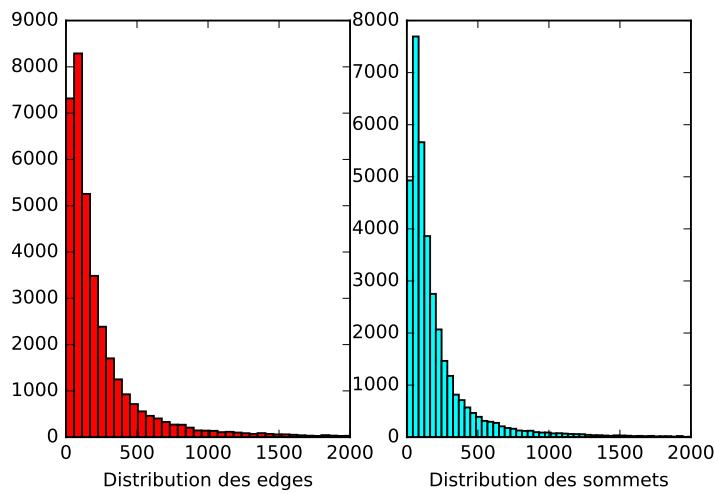


FIGURE 4: Distributions des sommets et des edges

une densité inférieure à 0.1. D'un point de vue mathématique, les graphes des villes sont donc très peu denses, à ne pas confondre avec la densité d'une ville. Cette information nous renseigne alors sur les degrés des nœuds des graphes : ils sont pour la plupart faibles. Le graphe est donc pratiquement *vide*, relativement à n'importe quel graphe possédant le même nombre de nœuds. Cette information sera cruciale par la suite, dans la partie 6 (*k-graphlets kernel*).

On peut confirmer en partie cette affirmation en regardant les degrés les plus communs des graphes. Pour 30% d'entre eux, le degré le plus présent est le degré 1 et le degré 3 pour le reste (70%).

La Figure 4 nous renseigne sur ces distributions. On peut observer alors que la moyenne du nombre de nœuds est 250, et celle du nombre d'arêtes est de 300. Effectivement, les graphes avec lesquels on travaille sont donc de taille modérément *grandes*. On peut par ailleurs remarquer qu'une centaine de graphes ont plus de 1000 nœuds et arêtes. Cela va rendre le travail assez difficile dans la mesure où il faudra surveiller les temps de calcul des algorithmes.

D'autre part, on peut étudier la densité des graphes. La notion de *densité* employée ici se réfère à la densité d'un graphe, c'est-à-dire le rapport entre le nombre d'arêtes divisé par le nombre d'arêtes possibles compte tenu du nombre de sommets du graphe. La totalité des graphes de notre collection possèdent une densité inférieure à 0.2, et plus de 99% de ceux-ci possèdent

2.3.2 Échantillonnage

Maintenant que nous avons décrit la collection de graphes qui va être étudiée, nous pouvons créer un échantillon $S = (G_1, \dots, G_n)$ représentatif de cette collection, sur lequel nous travaillerons. Ceci nous permettra de réduire grandement les temps de calcul, qui sont ici plutôt longs. Pour respecter les caractéristiques de la collection initiale, l'échantillonnage est aléatoire. De plus, il faut choisir assez de graphes pour que les caractéristiques de la collection soient conservées. Finalement, l'échantillon $S = (G_1, \dots, G_n)$ sera composé de 20% des graphes des villes françaises, choisis aléatoirement. Après vérification, les distributions et autres caractéristiques vues dans la partie précédente sont conservées.

3 Méthodologie

3.1 Choix de la méthodologie

Une fois les graphes obtenus, le but est donc de les étudier, et d'essayer de voir si certaines catégories de topologies peuvent être reconnues, ou non.

Au delà d'une étude purement axée sur les résultats interprétables directement, le but majeur de mon travail est de pouvoir proposer un outil qui permet d'étudier la topologie des villes. Cet outil se doit d'être capable de résumer l'information qui est pertinente dans les graphes des villes : la structure de ceux-ci. Il faut donc pouvoir être capable de s'affranchir de l'information qui peut-être considérée comme du bruit : **on ne veut pas être capable de dire que deux villes se ressemblent parce qu'elles sont grandes, on veut pouvoir dire que se ressemblent car elles ont une topologie similaire..** Attention cela ne veut pas dire que la taille des villes ne peut pas être un critère qui puisse influencer la topologie d'une ville (et on peut être amené à penser que cela peut justement être le cas), mais plutôt qu'il ne faut pas que la taille, par exemple, soit l'information résumée dans notre outil, car il serait alors inutile (il suffirait de garder seulement l'information taille). Si l'on arrive à créer un outil qui puisse résumer au mieux l'information *topologie* contenue dans les graphes qui représentent les villes, alors toutes les corrélations avec les hypothétiques variables déterminantes dans la topologie d'une ville seront prises en compte. On gardera bien en tête cette notion de *topologie* des villes qui motive l'étude.

Deux méthodologies se sont offertes à moi pour étudier cette collection de graphes.

1. Trouver m variables pertinentes qui puissent résumer l'information *topologie* dans les villes. Dans ce cas, la base de données à étudier serait une matrice X où l'élément $X_{i,j}$ représente la caractéristique j du graphe (ou de la ville) i .
2. Trouver un kernel K sur les graphes capable de résumer l'information *topologie* dans les villes.

Idéalement, il aurait fallu essayer les deux méthodes, cependant par faute de temps j'ai du choisir une des deux. **Mon choix s'est porté sur la méthode 2..** D'une part car cette méthode, si elle aboutit, répond mieux à la problématique qui vise à trouver un outil capable de résumer l'information "topologie" des villes. D'autre part, cela n'a "jamais" été fait, en tout cas je n'ai pu trouver aucun article qui se rapprochait ne serait-ce qu'un peu de ce que j'ai étudié pendant mon stage. Les kernels sur les graphes sont un sujet *à la mode* et très utile, qui a donné de bons résultats notamment en génétique ([1]), et qui étend le machine learning à d'autres types de données. Un extrait de [2] présente exactement cela, et nous pouvons retrouver ce type de commentaire dans tous les articles que nous avons pu trouver :

Graph-structured data is becoming more and more abundant : examples are social networks, protein or gene regulation networks, chemical pathways and protein structures, or the growing body of research in program flow analysis. To analyze and understand this data, one needs data analysis and machine learning methods that can handle large-scale graph data sets. For instance, a typical problem of learning on graphs arises in chemoinformatics : In this problem one is given a large set of chemical compounds, represented as node- and edge-labeled graphs, that have a certain function (e.g., mutagenicity or toxicity) and another set of molecules that do not have this function. The task then is to accurately predict whether a new, previously unseen molecule will exhibit this function or not. A common assumption made in this problem is that molecules with similar structure have similar functional properties.

Cependant, il existe un trade-off entre temps de calcul et représentation fidèle de la topologie du graphe. Il est difficile de simplifier *suffisamment* le kernel afin qu'il prenne en compte l'information voulue, tout en étant calculable en un temps polynomial. C'est une problématique en plein essor, qui ne date pas de plus de 10 ans. Il est donc académiquement très intéressant de choisir la méthode 2, plus novatrice dans son approche. Cependant cette méthode est aussi plus *risquée* dans le sens où j'ai peu d'articles auxquels me référer pour alimenter mon travail, étant donné que l'utilisation de kernels sur les graphes est très récente. Voyons ensemble quelques rappels sur les kernels, afin de se lancer sereinement dans l'étude des kernels sur les graphes.

3.2 Rappels sur les kernels

Soit χ un ensemble non-vide. Soit $K : \chi \times \chi \rightarrow \mathbb{R}$ une fonction symétrique. Alors K est un *kernel semi-défini positif sur χ* , ou *noyau semi-défini positif sur χ* , si et seulement si, $\forall n \in \mathbb{N}, \forall (x_1, \dots, x_n) \in \chi^n, \forall (c_1, \dots, c_n) \in \mathbb{R}^n :$

$$\sum_{i,j}^n c_i c_j K(x_i, x_j) \geq 0$$

De plus, le Théorème de Moore-Aronszajn assure le résultat suivant :

K est un kernel semi-défini positif sur l'ensemble χ , *si et seulement si* il existe un unique espace de Hilbert \mathcal{H} et une unique fonction ϕ tels que $\forall (x_i, x_j) \in \chi^2 :$

$$K(x_i, x_j) = \langle \phi(x_i) | \phi(x_j) \rangle_{\mathcal{H}}$$

où $\langle . | . \rangle_{\mathcal{H}}$ représente le produit scalaire défini sur \mathcal{H} , de part sa qualité d'espace de Hilbert.

Ce résultat est très important en *machine learning*. On dispose d'algorithmes qui ne reposent que sur les distances, ou les valeurs des produits scalaires, entre les individus, comme k-Nearest Neighbors, Support Vector Machine, Analyse en Composantes Principales, Spectral Clustering, Hierarchical Clustering, et j'en passe. Cela implique que si l'on trouve un *kernel semi-défini positif* valide sur la base de données $D = (X_1, \dots, X_n)$, alors on peut appliquer les algorithmes cités précédemment à la base de données $D^* = (\phi(X_1), \dots, \phi(X_n))$. On peut donc travailler sur les données projetées par ϕ dans l'espace \mathcal{H} qui est souvent de plus grande dimension (voir infinie). Cette projection peut permettre de *démêler* les données, ou de faire en sorte que les algorithmes donnent de meilleurs résultats. Un exemple bien connu est un jeu de données ayant deux classes distribuées selon deux cercles concentriques de rayons différents. Un kernel adapté permet de trouver un hyper-plan non-linéaire évident qui permet de parfaitement classer les données.

Ici, utiliser un kernel K sur les graphes (G_1, \dots, G_n) nous permet de projeter les graphes dans un espace de Hilbert \mathcal{H} grâce à ϕ , dans lequel il est possible d'obtenir les valeurs des produits scalaires entre les projetés des graphes $(\phi(G_1), \dots, \phi(G_n))$. On peut ranger ces valeurs dans la matrice de Gram $\mathcal{G} = (\mathcal{G}_{i,j})_{i,j \in [|1, \dots, n|]^2}$ associée à K . Ainsi $\forall (i, j) \in [|1, \dots, n|]^2 :$

$$\mathcal{G}_{i,j} = K(G_i, G_j) = \langle \phi(G_i) | \phi(G_j) \rangle_{\mathcal{H}} \quad (1)$$

Cette matrice de Gram \mathcal{G} nous permet donc d'utiliser une myriade d'algorithmes sur notre collection de graphes (G_1, \dots, G_n) qui représentent les villes. Certains de ces algorithmes utilisent des distances entre les données comme argument. Comme K est un produit scalaire sur \mathcal{H} , on peut aussi en déduire la matrice des distances $\mathcal{D} = (\mathcal{D}_{i,j})_{i,j \in [|1, \dots, n|]^2}$, où $\forall (i, j) \in [|1, \dots, n|]^2 :$

$$\mathcal{D}_{i,j} = \|\phi(G_i) - \phi(G_j)\|_{\mathcal{H}} = \sqrt{K(G_i, G_i) + K(G_j, G_j) - 2K(G_i, G_j)} \quad (2)$$

Évidemment, le choix du kernel semble crucial. Mon but est de trouver un, ou des, kernels qui prennent en compte l'information *topologie* des graphes (et donc des villes). Les parties suivantes seront donc consacrées à une étude des différents kernels qui existent dans la littérature actuelle. Je chercherais les kernels qui s'adaptent le mieux à ma problématique, et à la collection de graphes à ma disposition.

4 Random walk kernel

4.1 Définition

Le *random walk kernel* fut proposé par Gärtner et Al. () en 2003. C'est historiquement le premier kernel sur les graphes que j'ai pu trouver dans la littérature. Le principe est à priori simple : étant donné deux graphes, le kernel effectue des marches aléatoires dans les deux graphes, puis compte le nombre de marches aléatoires communes. Deux marches aléatoires sont communes si leurs longueurs sont égales et que leurs séquences de labels sont les mêmes (dans le cas de graphes labelés, sur les noeuds ou bien les arêtes). Dans notre cas nous n'avons pas de labels sur les noeuds ou les arêtes donc il s'agit juste de compter les marches aléatoires de même taille.

4.2 Problèmes : tottering et temps de calcul

Nous ne nous intéresserons pas ici à l'aspect mathématique de ce *random walk kernel*. En effet, l'utilisation de ce kernel a été impossible dans notre cas, donc par soucis de concision il semble inutile de présenter complètement le kernel. On admettra donc que le *random walk kernel* a une complexité de $O(n^6)$, n étant le nombre de noeuds. Ce kernel est donc seulement calculable sur des graphes de tailles "raisonnables". Raisonnables est un terme assez

flou, mais il est clair qu'il ne qualifie pas les graphes des villes que nous étudions. Les graphes des villes françaises possèdent en moyenne 1400 nœuds et 1700 arêtes, ce qui rend le calcul bien trop long. J'ai tout de même codé entièrement le kernel sur Python, et essayé de l'appliquer sur l'échantillon $S = (G_1, \dots, G_n)$ représentatif des villes françaises, construite dans la partie Échantillonnage. Le code de ce kernel est disponible dans l'annexe B. Le résultat est celui attendu : le calcul est trop long. Il faut plusieurs minutes pour chacun des graphes, et ceux qui sont trop grands comme Marseille ou Lyon prennent un temps supérieur à une journée.

Le kernel n'est donc pas utilisable dans le cadre de notre étude. Par dessus le marché, le *random walk kernel* est non seulement long à calculer, mais il présente aussi un autre problème, le *tottering*. Les marches aléatoires du kernel s'autorisent à visiter plusieurs fois les mêmes nœuds et/ou arêtes. Cela peut créer le phénomène de *tottering*, où les mêmes marches aléatoires se répètent, et cela crée des similarités artificielles. Par exemple la marche aléatoire peut faire l'aller-retour sur la même arête un grand nombre de fois. Il s'agit d'une raison de plus pour ne pas utiliser le *random walk kernel*.

Heureusement ce kernel fut un des (voir le) premier proposé, et ce il y a 13 ans ! Depuis 2003, la recherche s'est employée à proposer des alternatives au *random walk kernel*. Meilleurs temps de calcul, pas de tottering... Nous allons enfin pouvoir obtenir des premiers résultats.

5 Shortest path kernel

5.1 Définition du kernel et approche intuitive

Le *shortest-path kernel*, proposé par Borgwardt et Al. ([\[3\]](#)), est une alternative connue au *random-walk kernel*. Ce kernel ne présente pas le problème du *tottering*, et il est calculable dans un temps polynomial. Plus précisément, sa complexité est $O(n^4)$. Comparé à la complexité $O(n^6)$ du *random-walk kernel*, on peut maintenant penser que celui-ci sera calculable en un temps raisonnable sur notre collection de graphes, ou au moins sur un échantillon représentatif de cette collection.

Commençons par expliquer le principe du *shortest-path kernel*, et prouvons que celui-ci défini bien un kernel sur les graphes.

Définition (Shortest-path kernel k_{sp}) : Soient (G_1, G_2) deux graphes pondérés non-orientés, alors $k_{sp}(G_1, G_2)$ est défini ainsi :

1. Calcul de $(S(G_1), S(G_2))$, matrices d'adjacence associées aux graphes des plus courts chemins de (G_1, G_2) . (Algorithme de Floyd-Warshall)
2. Calcul de $(\phi(G_1), \phi(G_2))$, où l'élément i de $\phi(G_j)$ est le nombre d'occurrences de plus courts chemins de taille i dans $S(G_j)$.
3. $k_{sp}(G_1, G_2) = \langle \phi(G_1) | \phi(G_2) \rangle_{\mathcal{H}}$

Ici, $\mathcal{H} = \mathbb{R}^m$ où m est la longueur du *plus long* plus court chemin dans (G_1, G_2) .

Propriété : k_{sp} est un kernel semi-défini positif.

Preuve : On peut définir $k_{sp}(\cdot, \cdot)$ par la fonction $\langle \phi(\cdot) | \phi(\cdot) \rangle_{\mathcal{H}}$, qui est un produit scalaire sur \mathcal{H} , espace de Hilbert. Or tout produit scalaire sur un espace de Hilbert est un kernel semi-défini positif. \square

Pour ce qui est d'une approche plus intuitive, le *shortest-path kernel* établira que deux graphes (G_1, G_2) sont similaires (i.e la valeur de $k_{sp}(G_1, G_2)$ sera *grande* relativement aux autres valeurs du kernel) si (G_1, G_2) ont *beaucoup* de plus courts chemins de tailles égales. À l'inverse, si (G_1, G_2) ont peu de plus courts chemins de tailles égales, alors la valeur du produit scalaire, et donc de $k_{sp}(G_1, G_2)$, sera *faible*. Cela provient grossièrement du fait que : $\forall (A = [a_1, \dots, a_n], B = [b_1, \dots, b_n]) \in (\mathbb{N}^n)^2$,

$$\langle A | B \rangle_{\mathcal{H}} = \sum_{i=1}^n (a_i * b_i) \leq \sum_{i=1}^n (a_{(i)} * b_{(i)}) = \langle \tilde{A} | \tilde{B} \rangle_{\mathcal{H}} \quad (3)$$

où \tilde{A} est le vecteur A dont les coordonnées sont rangées dans l'ordre croissant.

5.2 Normalisation du kernel

Il est clair que la répartition des tailles des villes est un problème. En effet certaines similarités peuvent être faussées par la taille des villes. Plus une ville est grande, plus les chances sont grandes pour que cette ville possède *beaucoup* de plus courts chemins d'une certaine longueur. Ainsi à cause de ce problème la valeur de k_{sp} sera artificiellement grande. Une manière de régler ce problème est de *normaliser* le kernel. À partir d'une matrice de Gram K , on peut en déduire la matrice de Gram normalisée K_{norm} de la manière suivante :

$$\forall (i, j) \in [|1,..n|]^2, (K_{norm})_{i,j} = \frac{K_{i,j}}{\sqrt{K_{i,i}K_{j,j}}}$$

Théoriquement, *normaliser* un kernel revient à forcer nos données $(G_1,..,G_n)$ projetées par ϕ dans \mathcal{H} à appartenir à l'hyper-sphère de rayon 1. Ainsi $\forall i \in [|1,..n|]$, $\|\phi(G_i)\|_{\mathcal{H}} = 1$. Un kernel normalisé définit évidemment un kernel semi-défini positif. Ici, normaliser la matrice de Gram nous permet de nous débarrasser d'un effet taille qui pollue l'information apportée par le kernel.

5.3 Apprentissage non-supervisé sur les villes françaises

5.3.1 Motivation

Maintenant que le *shortest-path kernel* est présenté, nous allons l'appliquer à notre échantillon $S = (G_1,..,G_n)$ représentatif des villes françaises, que nous avons construit dans la partie reliée à l'échantillonnage.

D'un point de vue computationnel, le *shortest-path kernel* est bien calculable sur S en un temps raisonnable. La matrice de Gram normalisée \mathcal{G}_{sp} est calculable en approximativement 5400s, soit 1 heure et demie. J'ai codé entièrement ce kernel, et le code est disponible dans mon GitHub. Le lien est donné dans l'annexe B.

Une fois \mathcal{G}_{sp} calculée, nous pouvons enfin étudier les villes, à travers le *shortest-path kernel*. Il s'agit donc de visualiser les données, et tenter de voir si des catégories naturelles se distinguent dans cette base de données S : il s'agit donc d'apprentissage non-supervisé. Nous allons utiliser deux algorithmes qui permettent d'effectuer cette tâche : *agglomerative hierarchical clustering* et *ACP kernélisée*.

5.3.2 Catégorisation avec l'*agglomerative hierarchical clustering*

L'*agglomerative hierarchical clustering* est une technique de clustering largement utilisée. Elle prend en entrée de l'algorithme une matrice de distances. Initialement, chaque individu forme une classe, soit n classes. On cherche à réduire le nombre de classes au fur et à mesure. À chaque étape, on fusionne deux classes, réduisant ainsi le nombre de classes total. Les deux classes choisies pour être fusionnées sont celles qui sont les plus « proches » au sens d'un certain critère (dit d'*agrégation*), en d'autres termes, celles dont la dissimilarité entre elles est minimale. La méthode suppose qu'on dispose d'une mesure de dissimilarité entre les individus ; dans le cas de points situés dans un espace euclidien, on peut utiliser la distance euclidienne comme mesure de dissimilarité. Ici nous utiliserons la distance associée au *shortest-path kernel* normalisé, donnée par l'équation (2). Nous utiliserons la distance de Ward comme indice d'*agrégation*, et nous le justifierons de manière quantitative plus tard. Ce critère vise à maximiser l'*inertie inter-classe* :

$$ward(C_i, C_j) = \frac{n_i * n_j}{n_i + n_j} \|\phi(g_i) - \phi(g_j)\|$$

Où n_i et n_j sont les effectifs des deux classes (C_i, C_j) , g_i et g_j leurs centres de gravité respectifs.

Les résultats de cet algorithme sont recensés dans un dendrogramme, qui est un arbre qui récapitule toutes les fusions entre classes. Une fois que l'on a décidé du nombre de classes à retenir, on peut obtenir les individus de chaque classe en coupant simplement l'arbre au seuil choisi. Pour juger de la qualité du clustering, il nous faut un critère qui puisse s'appuyer seulement sur les distances entre les individus (qui sont ici des graphes). Disons que les données $(G_1,..,G_n)$ soient catégorisées dans m clusters. Pour $i \in [|1,..n|]$, soit $a(i)$ la distance moyenne entre G_i et les autres graphes du même cluster. Intuitivement, plus $a(i)$ est faible, plus G_i est bien catégorisé. On peut maintenant définir la distance d'un graphe à un cluster de graphes C par la moyenne des distances de G_i à tous les graphes de C . Soit $b(i)$ la distance minimale de G_i à un autre des $(m-1)$ clusters. Le cluster ainsi visé est le cluster voisin de G_i . On peut maintenant définir la *silhouette* $s(i)$ de G_i par :

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

Le *score silhouette* s sera défini comme la moyenne de la *silhouette* sur $(G_1,..,G_n)$. Ainsi défini, le *score silhouette* est un nombre appartenant à $[-1, 1]$. Si le clustering est excellent, alors $a(i)$ sera très faible, et $b(i)$ sera relativement grand. En effet le graphe sera proche (au sens de la distance dans \mathcal{H}) des graphes dans son cluster, et relativement loin des graphes de son cluster voisin. Ainsi plus le clustering est bon, plus s sera proche de 1.

La Figure 5 présente le dendrogramme obtenu après avoir appliqué l'*agglomerative hierarchical clustering* sur l'échantillon S .

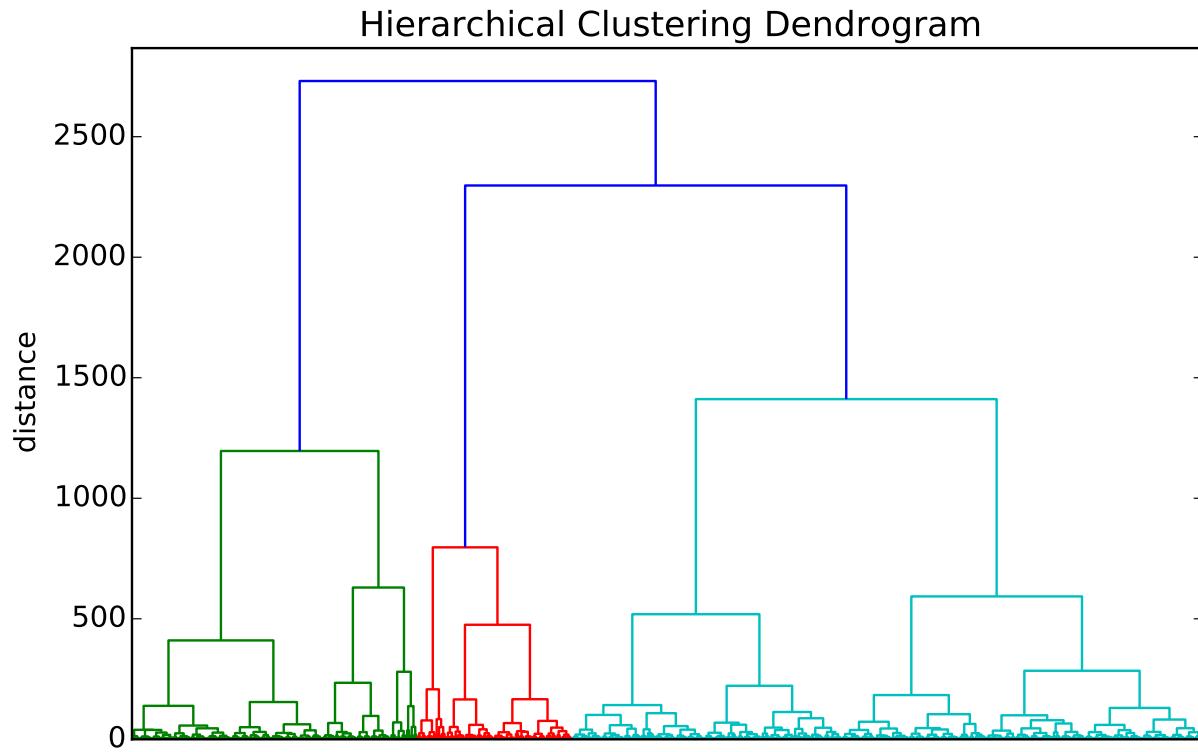


FIGURE 5: Dendrogramme du hierarchical clustering sur S , en utilisant le shortest-path kernel normalisé

Il semblerait, en observant le dendrogramme, que les données se répartissent naturellement en 3 ou 6 catégories. Nous allons donc utiliser le *score silhouette* pour décider du nombre de catégories, et vérifier que la qualité du clustering est acceptable. Pour 3 catégories, le *score silhouette* est de 0.57, tandis qu'il est de 0.52 pour 6 catégories. Ainsi deux conclusions peuvent être tirées de ces résultats. Premièrement le clustering est plutôt bon, on peut avoir confiance en cette catégorisation. Deuxièmement, il semblerait que les graphes de S , et donc par extension les villes françaises, se répartissent en 3 catégories différentes. Essayons d'interpréter ce résultat.

5.3.3 Interprétation des clusters

Après un étude rapide des 3 clusters, il semble évident que ces catégories soient très fortement corrélés à la taille des villes. La taille d'une est définie par la taille du graphe qui la représente. De manière formelle, la taille d'un graphe est $\text{card}(E)$, soit le nombre d'arêtes. Pour mettre en évidence cette corrélation, la Figure 6 ci-contre recense les boxplots des tailles des villes parmi les clusters. Pour chaque boxplot, la ligne rouge correspond à la moyenne des tailles dans le cluster : on obtient 45, 189 et 1112.

Néanmoins, le plus frappant reste la répartition des tailles dans les clusters. Les boxplots nous permettent de voir que les graphes sont très peu à cheval sur les autres clusters voisins, en terme de taille. La séparation entre les catégories semble vraiment liée à la taille : tous les graphes dont la taille est inférieure à un certain seuil (qu'on peut estimer à 100) sont catégorisées dans le cluster des *petites* villes. De même pour les *grandes* villes dont la taille est supérieure à un seuil d'environ 500. Le reste des villes sont catégorisées comme villes de *taille moyenne*.

Ces résultats ne sont pas de bonne augure. Le *shortest-path kernel* n'a pas l'air en mesure de prendre en compte l'information *topologie* contenue dans les graphes des villes françaises. Si cela avait été le cas, alors on pourrait énoncer un résultat comme le suivant : "deux villes de tailles très différentes ne peuvent pas se ressembler topologiquement". Or on sait qu'un tel résultat est faux. De nombreuses villes de tailles raisonnables ont été construites en imitant l'organisation et la topologie de grandes villes. Les villes de devraient pas se distinguer par leurs tailles, de manière aussi restrictive que le clustering suggère. Ces considérations nous amènent à penser que ce kernel ne répond pas correctement à la problématique que l'on s'était posé. Nous vérifierons cela dans la partie 7 (*Application à un problème de classification binaire*), dans laquelle nous utiliserons le kernel pour répondre à un problème de classification.

5.3.4 Visualisation des clusters : ACP kernélisée

L'analyse en composantes principales (ACP) est un méthode statistique qui consiste à transformer des variables corrélées entre elles en nouvelles variables dé-correlées les unes des autres. Ces nouvelles variables sont nommées "composantes principales", ou axes principaux. Lorsqu'on veut compresser un ensemble de N variables aléatoires, les n premiers axes de l'analyse en composantes principales sont un meilleur choix, du point de vue de l'inertie ou de la variance.

Cette méthode statistique fait partie des algorithmes qui peuvent être *kernélisés*. L'*ACP kernélisée* est une extension de l'ACP classique, où l'ACP se fait sur les données projetées $(\phi(G_1), \dots, \phi(G_n))$ dans l'espace de Hilbert \mathcal{H} . Ceci est rendu possible en utilisant seulement les valeurs des produits scalaires dans \mathcal{H} , i.e la matrice de Gram. Les résultats s'interprètent de la même façon que les résultats d'une ACP classique. Nous ne reviendrons pas sur le principe de l'ACP que nous avons largement étudié cette année, pour des soucis de concision.

Les résultats de l'*ACP kernélisée* sont présentés dans la Figure 7. Les trois premiers axes expliquent 95% de l'inertie. L'axe 1 traduit à lui seul 63% de l'inertie, tandis que l'axe 2 compte 25% de l'inertie et l'axe 3, 7%. Ainsi nous aurions pu omettre les graphiques contenant l'axe 3, cependant cela aurait soustrait le lecteur à la beauté de cette projection. Lorsque l'on projette les coordonnées des données dans l'espace euclidien de dimension 3, celles-ci forment une variété de dimension 3 remarquable. L'annexe A (Figure 13) est une représentation 3D de cette variété de dimension 3.

Il semble donc, selon la Figure 7, que l'axe 1 représente la taille des villes. Les 3 clusters sont en effet très bien séparés par cet axe. Plus une ville est grande, plus sa coordonnée sur l'axe 1 sera positivement grande. Cette interprétation nous permet de comprendre pourquoi l'*agglomerative hierarchical clustering* a formé ces 3 catégories.

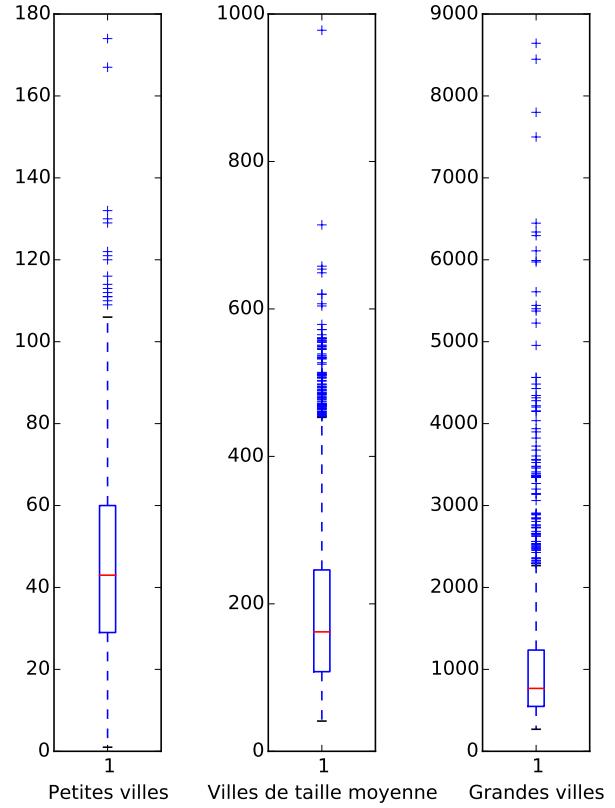


FIGURE 6: Boxplots des tailles des villes dans les 3 catégories.

Cependant, l'interprétation de l'axe 2 est plus floue, il semblerait que ce soit aussi relié à la taille des villes, mais de façon moins claire.

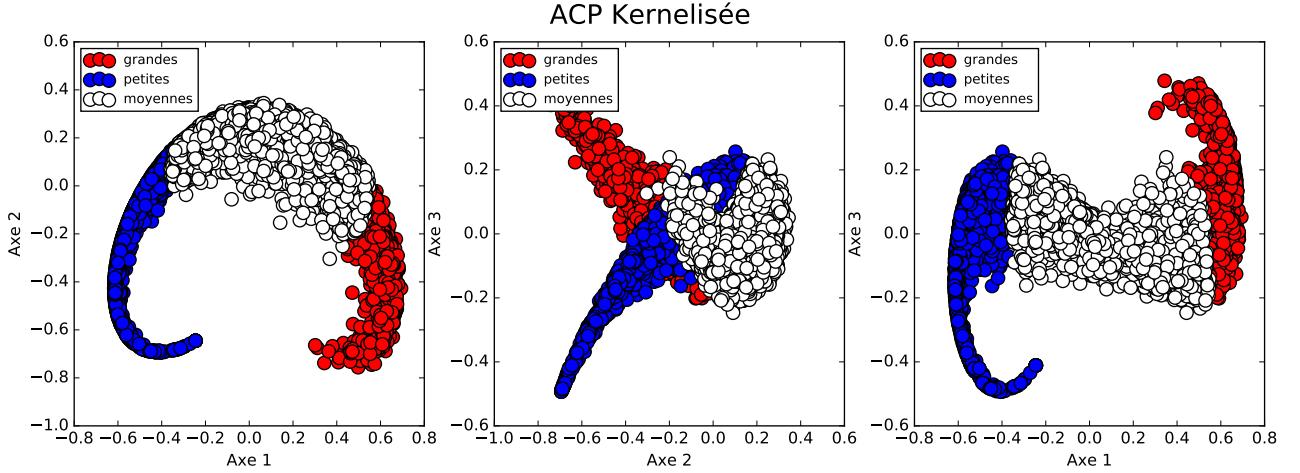


FIGURE 7: Visualisation des clusters à l'aide des axes de l'ACP kernelisée

5.3.5 Utilisation des informations supplémentaires sur les graphes

Rappelons-nous que nous possédons des informations supplémentaires sur les routes des villes françaises. Notamment le type des routes : pistes cyclables, nationales, autoroutes, chemins... Nous avons essayé de prendre en compte ces informations, afin de voir si les résultats étaient différents. Ceci s'applique spécialement bien avec le *shortest-path kernel*, car nous pouvons modifier les arêtes des graphes en les labelant.

Ainsi, nous avons créé des labels, allant de 1 à 8, pour les arêtes des graphes des villes. Le graphe est donc maintenant pondéré, sur les arêtes. Le *shortest-path kernel* trouve les chemins les plus courts, et les trouve maintenant en prenant en compte ce poids. Prenons un exemple : soient deux nœuds A et B reliés par une arête dont le poids est 6. Alors, aller du sommet A à un sommet B coûte maintenant 6, et non plus 1 comme avant. Cela modifie donc les plus courts chemins, à priori. Intuitivement, nous avons placé un poids plus fort pour les chemins piétons et cyclables, et moins fort pour les autoroutes et autres voies rapides. Le tableau I renseigne sur ces 8 types de voies.

TABLE 1: Type des voies en fonction de leur classe

Classe de la voie	1	2	3	4	5	6	7	8
Type de voie	Autoroute	Nationale	Départementale	Rue	Voie privée	Piste cyclable	Voie piétonne	Chemin

Les résultats obtenus sont quasiment les mêmes que précédemment. Il s'avère que cette information concernant les types de routes n'est pas assez importante pour changer les résultats de l'ACP kernélisée et du hierarchical clustering. Il était quand même important d'en être sûr que l'ajout d'informations ne change rien : ces informations auraient pu être pertinentes. On peut maintenant dire sereinement qu'elles ne le sont pas pour le *shortest-path kernel*.

5.4 Conclusions

Le *shortest-path kernel* semble donc étroitement lié à la taille des villes d'après le clustering et la visualisation que l'on avions donné. À priori, il semble donc ne pas prendre en compte complètement l'information *topologie*, comme nous le souhaitions. Nous vérifierons cela dans la partie Application, à travers un problème de classification binaire.

6 k-Graphlet kernels

6.1 Introduction sur les k-graphlet kernels

Étant donné que le *shortest-path kernel* semble ne pas répondre de manière satisfaisante à notre problématique, nous allons nous tourner vers une autre catégorie de kernels sur les graphes : les graphlet kernels. Nous nous baserons sur le travail exceptionnel de Shervashidze et Al. (4), duquel toute la partie suivante est tirée. Ces kernels se basent sur la distribution des graphlets dans un graphe. Soit G un graphe d'ordre n . Un graphlet g sur G est un **sous-graphe de G** d'ordre $k \leq n$. Ici on se limitera, comme dans l'article 4, à $k \in \{3, 4, 5\}$. Il existe 4 types de graphes d'ordre 3, 11 types de graphes d'ordre 4 et 34 types pour l'ordre 5. Par exemple, la Figure 8 présente les 11 différents graphlets d'ordre 4. Selon Shervashidze et Al., la distribution des 3,4,5-graphlets d'un graphe serait alors similaire à une statistique exhaustive de celui-ci, surtout si le graphe est *grand*. Voilà pourquoi il serait intéressant de baser un kernel sur cette distribution.

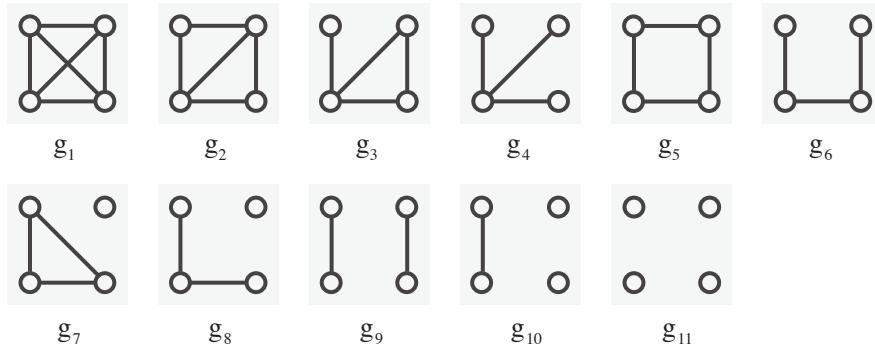


FIGURE 8: Les 11 différents graphlets d'ordre 4.

6.2 3,4,5-graphlets kernel

6.2.1 Définition

Voyons ensemble le principe du *3,4,5-graphlet kernel*, basé sur les distributions des 3,4,5-graphlets.

Définition (3,4,5-graphlet kernel k_g): Soient (G_1, G_2) deux graphes non-orientés, alors $k_g(G_1, G_2)$ est défini ainsi :

1. Calcul de $(\psi_3(G_1), \psi_3(G_2), \psi_4(G_1), \psi_4(G_2), \psi_5(G_1), \psi_5(G_2))$, où $\psi_k(G_i)$ correspond au vecteur d'occurrence des graphlets d'ordre k dans le graphe G_i
2. Soit $n(G_i)$ le nombre de 3,4,5-graphlets total dans G_i . On définit le *3,4,5-spectrum* $\phi(G_i)$ du graphe G_i par $\phi(G_i) = \frac{1}{n(G_i)} * (\psi_3(G_i), \psi_4(G_i), \psi_5(G_i))$
3. $k_g(G_1, G_2) = \langle \phi(G_i) | \phi(G_j) \rangle_{\mathcal{H}}$

Tout comme le *shortest-path kernel*, on peut déduire de sa définition que le *3,4,5-graphlet kernel* est bien un kernel semi-défini positif, en tant que produit scalaire dans un espace de Hilbert.

Cette fois-ci, l'espace de Hilbert \mathcal{H} associé au kernel est donc un espace de dimension $4 + 11 + 34 = 49$. On projette donc nos graphes, objets complexes difficiles à décrire, dans un espace de Hilbert de dimension 49 afin de pouvoir les comparer dans cet espace \mathcal{H} .

Intuitivement, deux graphes qui ont des distributions de 3,4,5-graphlets similaires auront une forte similarité, et inversement une faible similarité si leurs distributions sont très différentes.

6.2.2 Une complexité trop grande

Malheureusement, le calcul de ce kernel devient vite impossible quand les graphes sont trop *grands*. Un graphe d'ordre n possède $\binom{n}{k}$ graphlets d'ordre k . La conséquence est que la complexité du calcul est $O(n^k)$, ce qui n'est pas acceptable pour la taille des graphes de notre collection.

6.3 3,4,5-graphlets kernel avec échantillonnage

6.3.1 Définition

Il existe une solution à ce problème : l'échantillonnage. On peut approcher la distribution des 3,4,5-graphlets d'un graphe en prenant un échantillon assez grand de chacun des k-graphlets. Pour s'assurer que l'échantillon est significativement assez grand, une borne du nombre d'échantillons à tirer est donnée par Shervashidze et Al. (4), que nous avons utilisé.

Soit D une distribution de probabilité sur un ensemble fini $A = \{1, \dots, a\}$. Soient $X = (X_1, \dots, X_m)$ des variables indépendantes suivant D . Soit \hat{D}^m la distribution empirique de D telle que $\forall i \in A$:

$$\hat{D}^m(i) = \frac{1}{m} \sum_{j=1}^m \delta(X_j = i)$$

Alors, selon [4] et nous accepterons ce résultat, on peut construire une borne m qui suffit à assurer que la distribution empirique est assez proche de la *vraie* distribution. $\forall \epsilon > 0 \ \forall \delta > 0$:

$$m = \lfloor \frac{2 \log(2) * a + \log(\frac{1}{\delta})}{\epsilon^2} \rfloor$$

Ce nombre m suffit à assurer que $P \left\{ \|D - \hat{D}^m\|_1 \geq \epsilon \right\} \leq \delta$, ce qui permet de régler m en fonction de la précision que l'on souhaite.

Une simple application montre qu'environ 5 000 échantillons de k-graphlets par ordre suffit à approcher la *vraie* distribution des 3,4,5-graphlets de manière satisfaisante.

6.3.2 Limites

Le calcul se fait maintenant bien à l'aide de cet échantillonnage. Mais les résultats sont décevants : il y a bien trop de sous graphes vides dans les distributions des 3,4,5-graphlets des graphes de notre échantillon représentatif S . Par exemple, la moyenne des distributions des 3,4-graphlets sur l'échantillon S représentatif de notre collection de graphes est le vecteur de taille 15 (4 graphlets d'ordre 3 et 11 graphlets d'ordre 4) suivant :

$$[46\%, 4\%, < 1\%, < 1\%, 42\%, 7\%, 0\%, < 1\%, < 1\%, 0\%, < 1\%, < 1\%, < 1\%, < 1\%, < 1\%]$$

On voit très bien que 2 des 15 graphlets sont très présents. Ils correspondent aux graphlets vides d'ordre 3 et 4. À eux seuls, ils concentrent plus de 88% des graphlets échantillonés ! Et il faut savoir que les deux graphlets suivants les plus représentés sont des graphlets quasi-vides, qui ne comportent qu'une arête. C'est une conséquence directe de la faible densité de nos graphes, que l'on avait remarqué dans la partie *Statistiques descriptives*. Comme 99% des graphes possèdent une densité inférieure à 0.1, alors il est logique que si l'on tire 3, 4, ou 5 noeuds au hasard et qu'on forme le sous-graphe associé, les chances pour que ce sous-graphe soit vide ou qu'il n'ai qu'une arête sont immenses. C'est un problème car ce qui nous intéresse, ce sont justement les motifs contenus dans les graphes de notre collection. Ce sont ces mêmes motifs qui forment la topologie de la ville associée au graphe. Nous voulons pouvoir être capable de voir les différences entre ces motifs, ou les différences d'occurrence des motifs, et ici cette information est parasitée par ces sous-graphes vides.

6.4 Alternative : all connected 3,4,5-graphlets kernel

6.4.1 Définition

Heureusement, Shervashidze et Al. proposent justement, en fin d'article, une alternative pour les graphes dont le degré maximum d est relativement *faible*. En effet en règle générale les graphes qui présentent de l'intérêt (réseaux sociaux, génétique, etc...) sont comme les graphes des villes que nous avons : les degrés des noeuds sont généralement faibles. C'est pourquoi on peut adapter le kernel à ces graphes aux propriétés spécifiques. On peut donc définir le **all connected 3,4,5-graphlet kernel** k_{acg} . Le principe de ce kernel est le même que le *3,4,5-graphlet kernel*, mais on ne va compter que les graphlets qui sont connexes. On rappelle qu'un graphe est connexe si et seulement il existe un chemin entre toutes les paires de points du graphe. Intuitivement, il s'agit de graphes qui sont composés d'un seul *bloc*. Il en existe 2 pour les graphes d'ordre 3, 6 pour les graphes d'ordre 4 et 21 pour les graphes d'ordre 5. Par exemple, pour les graphes d'ordre 4, il s'agit des 6 premiers graphes de la Figure 8.

Ce nouveau kernel présente deux avantages majeurs par rapport à la version du *3,4,5-graphlet kernel avec échantillonnage*.

- Soit d le degré maximum d'un graphe G et $k \in \{3, 4, 5\}$. Alors il est possible de compter tout les k -graphlets connexes de G , **sans avoir à échantillonner**, en un temps de $O(nd^{k-1})$.

Preuve : Pour trouver ces k -graphlets connexes, il faut d'abord trouver tout les chemins de longueur $k - 1$ dans le graphe G . L'algorithme DFS (Deep First Search) peut trouver tout les chemins de longueur $k - 1$ partant d'un noeud v en $O(d^{k-1})$. Ainsi, on a besoin de $O(nd^{k-1})$ pour trouver tout les chemins de longueur $k - 1$ de G . Ensuite, il faut regarder les différents cas selon k .

- Pour $k = 3$: tout les 3-graphlets connexes possèdent un chemin de longueur 2, ainsi la tâche est finie dès lors qu'on a trouvé tout les chemins de longueur 2, ce qui se fait bien en $O(nd^2)$.
- Pour $k = 4$: il n'existe qu'un seul 4-graphlet connexe qui ne contiennent pas de chemins de longueur 3, c'est la 3-star (la graphe 4 dans la Figure 8). Pour trouver ces 3-stars, on regarde noeud par noeud, parmi les noeuds de degrés supérieurs ou égaux à 3, si une 3-star est formée. Cela coûte au maximum $O(d^3)$ par noeud, et donc bien $O(nd^3)$ en tout. Cela ajouté à la complexité de calcul pour les chemins de longueur 3, on retrouve bien $O(nd^3)$.
- Pour $k = 5$: Il existe 3 différents 5-graphlets connexes qui ne comportent pas de chemins de longueur 4. Pour les trouver, il suffit de trouver les 3-stars du graphe G , puis de vérifier si elles induisent le 5-graphlet connexe recherché, une opération qui coûte $O(d)$ par graphlet. On retombe bien donc sur $O(nd^4)$ au total.

□

- Le problème des k -graphlets vides est terminé. Les k -graphlets connexes représentent vraiment la **topologie du graphe**.

En effet, on peut maintenant penser que nous avons réussi à balayer toute l'information nuisible contenue dans les graphes. Nous ne comptons maintenant plus que les motifs *intéressants*, ceux qui font que certaines parties de la ville sont très denses, et d'autres plus parcimonieuses. **Ce kernel permet de mieux se concentrer sur ce qui caractérise la topologie d'une ville : la nature des motifs.** Centres villes quadrillés, rond-points, intersections et autres sont maintenant bien identifiés au premier plan, et ne sont plus perdus dans l'ombre des sous-graphes vides, comme pour le *3,4,5-graphlet kernel avec échantillonnage*.

En résumé, la projection des graphes se fait maintenant dans un espace de Hilbert \mathcal{H} de dimension $2 + 6 + 21 = 29$. Les projections des graphes (G_1, \dots, G_n) sont les vecteurs $(\phi_{acg}(G_1), \dots, \phi_{acg}(G_n))$ où $\phi_{acg}(G_i)$ représente la distribution des 3,4,5-graphlets connexes de G_i .

Ici, lorsqu'on applique le kernel à notre échantillon représentatif $S = (G_1, \dots, G_n)$ et qu'on calcule la matrice de Gram associée \mathcal{G}_{acg} , il n'est pas nécessaire de normaliser cette matrice. En effet comme les $(\phi_{acg}(G_i))_{i \in [|1 \dots n|]}$ représentent des distributions, qui sont donc des vecteurs qui somment à 1, alors il n'y a pas d'effet taille nuisible, comme il pouvait y avoir avec le *shortest-path kernel*.

6.4.2 Apport personnel : all connected 3,4,5-graphlets kernel modifié

Je me suis intéressé de plus près à la projection ϕ_{acg} . Observons la moyenne des distributions des 3,4,5-graphlets connexes sur S , qui est le vecteur suivant :

$$[< 1\%, 19\%, < 1\%, < 1\%, 2\%, 6\%, < 1\%, 27\%, 0\%, 0\%, 0\%, < 1\%, < 1\%, \\ < 1\%, < 1\%, < 1\%, < 1\%, 3\%, 39\%, < 1\%, 0\%, < 1\%, 2\%, 2\%, < 1\%, 0\%, < 1\%]$$

On voit, encore, que certains k -graphlets connexes sont très présents. Une fois de plus, ce sont les k -graphlets connexes les plus vides, ce qui est toujours une conséquence de la faible densité de nos graphes, et du fait que nos graphes représentent des villes. Les 3 graphlets connexes les plus présents sont les chemins de longueur 2, 3 et 4, ce à quoi on pourrait s'attendre dans une ville. La conséquence de cette remarque est que lorsque les produits scalaires sont calculés pour former la matrice de Gram \mathcal{G}_{acg} , les valeurs des kernels risquent d'être très proches les unes des autres. Les valeurs de la matrice peuvent finir par être trop proches les unes des autres, ce qui a pour conséquence dramatique de fausser les résultats de certains algorithmes. Nous verrons cela par la suite, dans la partie *Application*, où certains algorithmes comme *SVM* nécessite que la valeur du kernel associé à deux graphes topologiquement similaires soient assez différentes de la valeur du kernel associé à deux graphes topologiquement différents. On peut par exemple citer la librairie Scikit-Learn (5), qui nous renseigne sur ce genre de problématique dans leur page Web ([cliquez ici](#)) sur le *preprocessing* de données :

For instance, many elements used in the objective function of a learning algorithm (such as the RBF kernel of Support Vector Machines or the l_1 and l_2 regularizers of linear models) assume that all features are centered around zero and have variance in the same order. If a feature has a variance that is orders of magnitude larger than others, it might dominate the objective function and make the estimator unable to learn from other features correctly as expected.

Une manière de contourner ce problème est de changer d'échelle. Les distributions données par ϕ_{acg} ne sont pas à la bonne échelle, donc mon idée est de les centrer et les réduire. Nous allons considérer que chaque coordonnée des $\phi_{acg}(G_i)$ est une variable aléatoire, puis nous allons la centrer et la réduire.

Je vais donc transformer la *feature map* ϕ_{acg} en $\phi_{acg\star}$. La matrice de Gram $\mathcal{G}_{acg\star}$ associée au kernel $k_{acg\star}$ et à la collection de graphes (G_1, \dots, G_n) se calcule ainsi :

1. Calcul de $\Phi = (\phi_{acg}(G_1), \dots, \phi_{acg}(G_n))$, où $\forall i \in [|1,..n|] \phi_{acg}(G_i) = (X_i^1, \dots, X_i^{29})$.

X_i^j est la variable aléatoire donnant la distribution du j ème k-graphlet associée au graphe i . On va donc centrer et réduire ces variables. Cela demande d'estimer la moyenne et la variance de chaque variable : on utilise la moyenne et la variance empirique grâce aux données contenues dans Φ . On notera (\bar{X}_i^j) les variables centrées réduites.

2. Calcul de $\Phi^* = (\phi_{acg\star}(G_1), \dots, \phi_{acg\star}(G_n))$, où $\forall i \in [|1,..n|] \phi_{acg\star}(G_i) = (\bar{X}_i^1, \dots, \bar{X}_i^{29})$

3. Enfin on calcule la matrice de Gram : $\mathcal{G}_{acg\star} = \left(\langle \phi_{acg\star}(G_i) | \phi_{acg\star}(G_j) \rangle_{\mathcal{H}} \right)_{i,j \in [|1,..n|]^2} = \Phi^* (\Phi^*)^t$

Grâce au kernel $k_{acg\star}$, on peut s'attendre à des résultats satisfaisants. Le changement d'échelle équivaut à un changement d'unité, et il n'a pas d'incidence sur les profils de variation. Ainsi, maintenant les variations de chacune des variables X_i^j est comparable aux autres, et donc naturellement la valeur du kernel variera de manière appropriée.

J'ai codé entièrement **chacun** de ces k-graphlets, ce qui ne fut pas une mince affaire. Cela m'a pris une partie non négligeable de mon temps durant mon stage : il faut savoir qu'il n'existe aucune librairie qui donne directement les kernels sur les graphes (sur python ou R). J'ai donc codé le *3,4,5-graphlets kernel* et sa version avec échantillonnage, puis le *all connected 3,4,5-graphlets kernel*. Les codes sont disponibles via les liens GitHub, dans l'annexe B.

6.5 Apprentissage non supervisé sur les villes françaises

Comme pour le *shortest-path kernel*, nous allons tenter de faire une catégorisation, et de visualiser les données à travers le *all connected 3,4,5-graphlets kernel*. Nous utiliserons exactement la même méthodologie et algorithmes que pour le *shortest-path kernel*. On peut donc directement passer aux résultats de cet apprentissage non-supervisé.

Malheureusement, ces résultats ne sont pas directement interprétables. Je n'ai pas su interpréter la catégorisation issue de l'*agglomerative hierarchical clustering*. Les classes étaient très déséquilibrées et il semblait que cette catégorisation n'avait aucun sens. Pour ce qui est de l'ACP kernélisée, les deux premiers axes représentent 90% de l'inertie, et la Figure 9 en donne une représentation graphique.

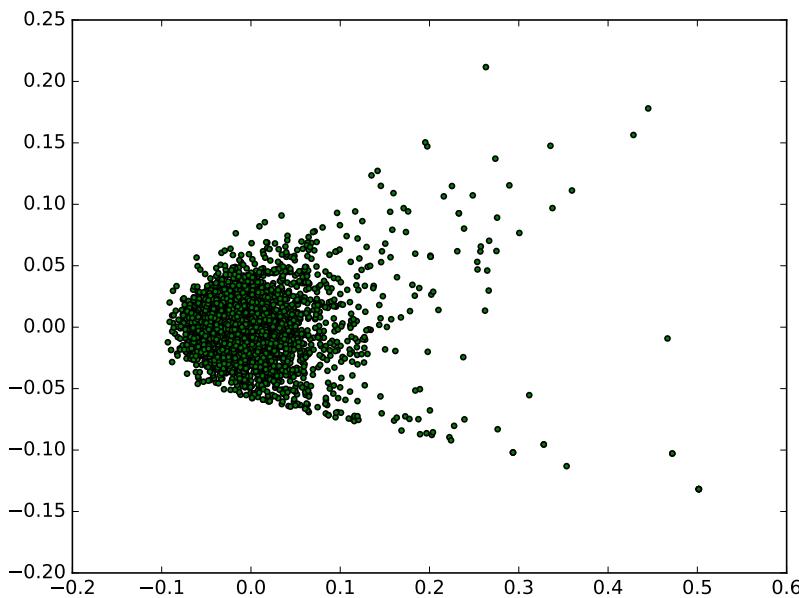


FIGURE 9: Deux premiers axes de l'ACP kernélisée

Aucune catégorie n'est visible. Il ne semble pas exister de **structure sous-jacente** parmi les villes françaises, lorsqu'on prend en compte leur topologie (à priori c'est notre intuition).

Nous sommes donc face à une impasse. Notre intuition nous dit que les deux *all connected 3,4,5-graphlets kernels* (modifié ou non) prennent en compte l'information *topologie*, cependant nous sommes incapables de le prouver avec ces résultats. C'est un des problèmes avec l'apprentissage non-supervisé : si l'on a pas une idée claire de ce que l'on veut observer (ou que l'on est pas en mesure d'interpréter des résultats), alors les algorithmes et leurs résultats peuvent s'avérer inutiles. Il faut donc changer de problème pour prouver que le *all connected 3,4,5-graphlets kernel* est bien le kernel que nous cherchions. Nous allons créer un problème d'apprentissage supervisé.

7 Application à un problème de classification binaire : France vs USA

7.1 Motivation

Notre but était de créer un outil capable de comparer et d'étudier la topologie des villes. Est-ce que l'information *topologie* est enfin prise en compte ? Les méthodes statistiques que nous avons utilisé jusqu'ici nous permettent de faire des conclusions surtout qualitatives et quasi subjectives. Dans cette partie, nous allons proposer une application pour les kernels présentés ci-dessus, et nous pourrons interpréter les résultats directement de manière quantitative. Ces résultats, libres de toute subjectivité, nous permettront de conclure sur la qualité ou non des kernels que nous avons étudié et proposé.

De la même manière que précédemment, nous avons récupéré les graphes des villes américaines. On pourra se référer à la partie *Récupération des données* pour se rappeler du procédé. Ainsi nous disposons maintenant des 36 000 graphes des villes françaises et des 30 000 graphes des ville américaines.

Nous allons utiliser des intuitions de géographes afin de créer un problème de classification binaire. La France et les États-Unis ont des histoires très différentes. La France, pays du vieux continent qu'est l'Europe, possède des villes dont la topologie est très spéciale. Selon les géographes, dans la plupart des villes se situerait un centre-ville dense avec de nombreuses rues sinuées, qui créent un réseau dense et désorganisé. Ce plan d'organisation est nommé, par les urbanistes, **plan radio-concentrique**. Au contraire, les États-Unis sont un pays très jeune du point de vue historique, et ses villes ne possèdent en général pas le même plan d'organisation. Les urbanistes disent que les villes américaines ont été construites selon un **plan hippodamien**. Les villes sont quadrillées : les rues sont rectilignes et se croisent en angle droit, créant des îlots de forme carrée ou rectangulaire.

Cependant, toutes les villes n'ont pas été construites aux même moment, donc on ne peut pas conclure de manière manichéenne sur la topologie des villes de ces deux pays. Par exemple, certaines villes françaises ont été détruites pendant les guerres mondiales, et ont été reconstruites selon des plans qui se rapprochent d'un plan hippodamien. De plus, selon les contraintes de relief ou de littoral, ou autres, les villes ont souvent une topologie unique. Mais les géographes et urbanistes s'accordent pour dire qu'il existe une tendance de topologie différente entre les villes américaines et les villes françaises. **Est-ce que nos kernels sont capables de détecter cette différence ?** Ils devraient en être capable s'ils répondent à notre problématique.

Nous avons donc des villes catégorisées 1 (françaises) et d'autres catégorisées 0 (américaines).

Il existe des algorithmes d'apprentissage, tel *SVM*, qui ne prennent en entrée qu'une simple matrice de Gram basée sur un kernel et les catégories des individus associés, et qui forment un prédicteur, ou classifieur, dont la tâche est de prédire la catégorie de villes.

Ainsi, nous allons créer une base de données d'apprentissage, D_{train} , et une base de test D_{test} . Nous prendrons un échantillon représentatif des villes de chaque pays pour former la base d'apprentissage. Ensuite, nous allons entraîner un algorithme basé sur les kernels sur la base d'apprentissage, et lui faire prédire les catégories des villes de la base de test. Enfin, nous pourrons quantitativement interpréter les résultats, en terme de précision pure, et via l'étude de courbes ROC.

Cette application va nous permettre de vérifier si les kernels que nous avons présenté répondent à notre problématique, et prennent bien en compte l'information *topologie* des villes.

7.2 Statistiques descriptives des graphes des villes américaines

Pour commencer, nous allons rapidement décrire la collection des graphes des villes américaines, comme nous l'avions fait pour les villes françaises dans la partie *Statistiques descriptives*. Nous allons décrire les graphes selon les mêmes critères : étude de la distribution des tailles et des ordres, densité et degrés les plus communs. Cette étude va nous permettre de voir s'il existe des différences fondamentales entre les graphes des deux pays. Si des différences évidentes existent entre les graphes, alors notre problème d'application n'aura pas beaucoup d'intérêt, car le problème de classification serait rendu trivial.

Dans la Figure 10, nous pouvons voir que la distribution des tailles et des ordres de graphes des villes américaines est sensiblement la même que celle des graphes des villes françaises (Figure 4). On ne peut donc pas discriminer les villes des deux pays par leurs tailles ou leurs ordres.

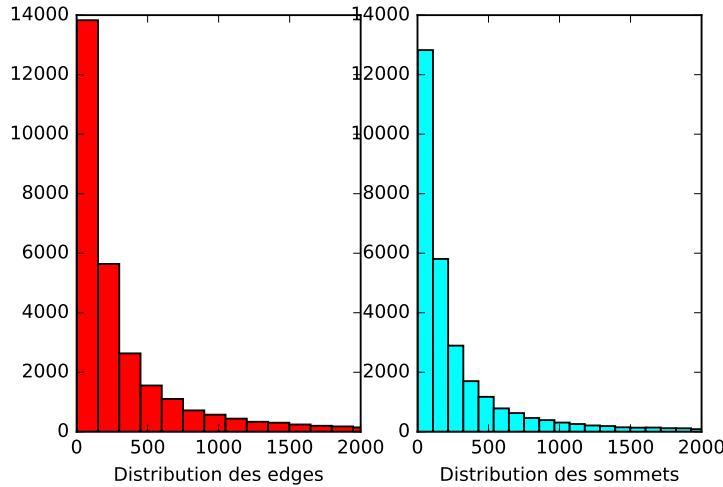


FIGURE 10: Distributions des sommets et des edges

Nous avons aussi la même situation qu'avec les graphes français concernant la densité des graphes américains : les graphes sont très peu denses. Comme avec la France, 99% des graphes possèdent un densité inférieure à 0.1. Enfin, les degrés les plus communs sont encore une fois à 30% le degré 1 et à 70% le degré 3.

Ainsi, à priori, ce problème de classification n'est pas évident. Les deux collections ont l'air très similaires. Il n'y a pas de caractéristiques évidentes qui ont l'air de nous permettre de classifier les graphes aisément. Pourtant notre intuition donnée par les géographes nous dit qu'il devrait quand même exister une différence de topologie entre ces deux collections de graphes. Voyons-voir si les kernels que nous avons présenté nous permettent de capturer cette différence.

7.3 Résoudre un problème de classification avec une matrice de Gram

Pour faire de la classification d'objets "non standards", c'est-à-dire qui ne sont pas des vecteurs, il faut trouver une mesure de distance entre ces objets, qui est ici le kernel sur les graphes (on rappelle que le kernel est un produit scalaire donc on peut en déduire une distance). À partir de cette distance, on déduit la matrice de Gram. On peut utiliser plusieurs algorithmes de classification pour cette matrice de Gram, comme *k-plus proches voisins "KNN"* (qui utilise la distance) ou *Support Vector Machine "SVM"* (qui utilise directement le kernel). À posteriori, SVM donne de bien meilleurs résultats, donc nous nous concentrerons sur cet algorithme par la suite. Nous ne comparerons pas les algorithmes : le but poursuivi ici est de comparer les kernels sur ce problème de classification afin de trouver le meilleur pour notre problème.

7.4 Support Vector Machine (SVM)

SVM est un algorithme d'apprentissage supervisé basé sur la création d'un hyperplan qui définit un seuil de décision pour la prédiction. Pour un problème de catégorisation binaire, l'hyperplan coupe l'espace dans lequel vivent les données en 2, formant le demi-espace associé à la catégorie 1 et le demi-espace associé à la catégorie 0. Ainsi, la prédiction de la catégorie d'une nouvelle donnée sera la catégorie du demi-espace dans laquelle la donnée vit. Pour trouver l'hyperplan optimal, l'idée est de maximiser la *marge*. La marge est la distance entre la frontière de séparation et les échantillons les plus proches, nommés *support vectors*. Le problème est donc de trouver cette frontière séparatrice optimale, à partir d'un ensemble d'apprentissage, en l'occurrence la base d'apprentissage D_{train} . Ceci est fait en formulant le problème comme un problème d'optimisation quadratique.

La deuxième idée clef avec SVM est l'utilisation des kernels. Comme on pourrait s'y attendre, certains problèmes de classification peuvent-être mieux résolus par des hyperplans complexes, courbes ou tordus (pas des hyperplans affines). Pour résoudre ce genre de problème, on utilise les kernels. La projection dans l'espace de Hilbert \mathcal{H} induite par ϕ permet souvent de séparer les catégories bien plus simplement. Cela est dû au fait qu'il est souvent plus simple de séparer des données lorsqu'elles sont dans un espace de dimension plus grande. De plus, comme le nombre de kernels possible est infini, on peut supposer qu'il existe des kernels bien choisis qui permettent une telle séparation. L'exemple en Figure 11 montre une représentation graphique de ce procédé (Source : <https://www.dtreg.com/solution/view/20>).

Separation may be easier in higher dimensions

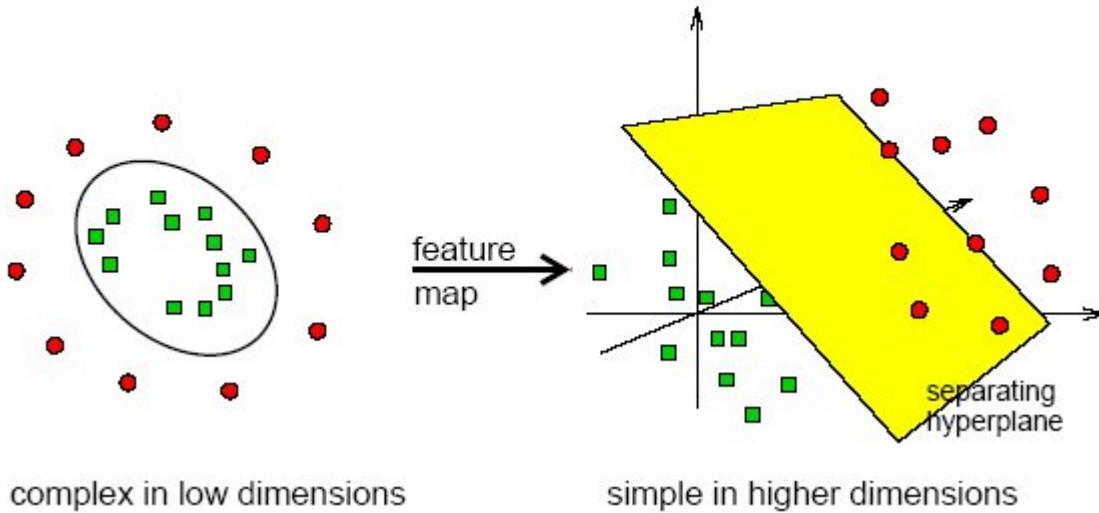


FIGURE 11: Utilisation des kernels pour SVM

La théorie autour de SVM est plus complexe que cela, et pour expliquer cet algorithme complètement il faudrait rentrer dans des détails longs et techniques : étude du problème d'optimisation, complexité de Rademacher et théorie de Vapnik-Cervonenkis. Pour des soucis de concision et parce que ce rapport n'est pas censé être un cours, nous ne rentrerons pas dans ces détails.

Nous allons utiliser SVM pour le problème de classification des nationalités des villes. Ici nous utilisons pas SVM comme nous l'avons présenté précédemment. Les graphes vivent dans un espace de dimension très grande : si on appelle n l'ordre maximal des graphes de nos deux collections, alors les graphes sont dans l'espace des graphes d'ordre inférieur ou égal à n . Il est très difficile de travailler dans cet espace, donc on se projette (via ϕ) dans un espace plus simple (\mathcal{H}) mais censé résumer l'information topologie contenue dans les graphes. Même si les intentions ne sont pas les mêmes, SVM marche exactement de la même façon, cherchant l'hyperplan affine optimal dans le feature space.

Un des avantages de SVM est qu'il est possible, pour chaque prédiction, de donner la probabilité de prédiction de chaque classe. C'est ainsi que se fait justement la prédiction : si la probabilité est inférieure à 0.5 alors la prédiction sera la catégorie 0, et inversement si la probabilité est supérieure à 0.5. Ceci nous permet de construire des courbes ROC (Receiver Operating Characteristic) très utiles pour comparer différents classifiants.

L'algorithme SVM comporte deux parties : la partie d'apprentissage où le prédicteur est créé, puis la phase de prédiction. La première étape nécessite la matrice de Gram X_{train} calculée sur la base D_{train} et le vecteur y_{train} des labels de chaque ville (1 pour la France, 0 pour les USA). En résolvant le problème d'optimisation, SVM crée donc l'hyperplan prédicteur. Ensuite, pour prédire la nationalité d'une ville, il faut alors calculer toutes les valeurs du kernel entre cette ville et les villes de D_{train} . Cela permet à SVM de savoir de quel côté de l'hyperplan prédicteur est la nouvelle ville. La prédiction est donnée sous forme de 0 ou 1. Pour prédire les catégories de la base de test D_{test} , il faut donc la matrice de Gram de prédiction X_{test} (nous la nommerons ainsi) des villes de la base D_{test} . Enfin, on peut utiliser les *vrais* labels des villes de D_{test} , le vecteur y_{test} , pour calculer la précision (le taux de bons classés).

7.5 Création de D_{train} et D_{test}

Pour créer D_{train} et D_{test} , nous avons recours à l'échantillonnage, pour des questions de temps de calcul. Pour se rendre compte des échelles, calculer la matrice de Gram pour le *all connected 3,4,5-graphlets* (modifié ou non) sur 7000 graphes prend environ 8h. Je n'ai clairement pas pu me permettre de travailler sur un plus grand nombre de graphes, car par dessus le marché la taille de la matrice de Gram augmente en n^2 avec le nombre n de graphes et cela devient vite difficilement contrôlable.

Afin de procéder à la classification, il faut choisir intelligemment D_{train} et D_{test} . Nous devons nous assurer que les résultats que l'on va obtenir peuvent se généraliser à tous les graphes des villes américaines et françaises, et qu'ils ne sont pas juste dûs à de la *chance*. Nous avons donc choisi de prendre un échantillon S_{fr} de 10% des graphes français,

et un échantillon S_{am} de 10% graphes américains créés par échantillonnage aléatoire. En juxtaposant les deux, nous obtenons D_{train} . Pour D_{test} , nous avons retenu 200 graphes par pays, aléatoirement choisis dans D_{train} . Pour des résultats plus précis, nous allons aussi faire de la cross-validation sur D_{train} . Nous allons recommencer plusieurs fois la classification en changeant D_{test} à chaque fois. Cela va permettre notamment d'estimer des moyennes et variances des précisions obtenus, ce qui est un critère intéressant pour comparer les kernels.

7.6 Comparaison des kernels

Nous allons comparer les 3 kernels qui semblent les plus intéressants : *shortest-path kernel*, *all connected 3,4,5-graphlets kernel* et *all connected 3,4,5-graphlets kernel modifié*.

Pour chacun des kernels, nous allons faire le même procédé, composé de 5 itérations. À chaque itération, on change D_{test} en l'échantillonnant aléatoirement dans D_{train} . Voici le programme de l'itération i :

1. Calcul de la matrice de Gram d'apprentissage X_{train} , et de la matrice de Gram de prédiction X_{test} .
2. Création du vecteur des labels d'apprentissage y_{train} et de test y_{test}
3. Méthode fit de SVM (issu de Scikit-Learn [5]) sur X_{train} et y_{train}
4. Prédiction sur X_{test} pour obtenir y_{pred}
5. Calcul de la précision à l'itération i τ_i :

$$\tau_i = \frac{1}{n} \sum_{i=1}^n \mathbf{1}_{(y_{pred,i} = y_{test,i})}$$

Ainsi, avec les $(\tau_i)_{i \in [|1,..5|]}$, on peut calculer pour chaque kernel une précision moyenne, et la variance de la précision. Le tableau 2 référence justement ces résultats.

TABLE 2: Résultats de la classification avec les différents kernels

Type de kernel	Précision moyenne	Variance de la précision
Shortest-path	66%	0.031
All connected 3,4,5-graphlets	78%	0.026
All connected 3,4,5-graphlets modifié	83%	0.027

Les conséquences de ces résultats sont les suivantes. D'une part, nous sommes rassurés : les kernels que nous avons présenté prennent bien en compte l'information *topologie* ! Les 3 kernels ont des précisions dont la moyenne et la variance montrent qu'elles significativement supérieures à 50%. Cela montre que les kernels arrivent à déceler des différences entre les graphes américains et les graphes français. Ces différences sont forcément dues aux différences de topologie entre les graphes de ces deux pays, compte tenu des définitions des kernels, et de notre analyse en terme de statistiques descriptives. **Il existe quelque chose de fondamentalement différent (autre que la taille, l'ordre, la densité ou une autre statistique évidente) entre la topologie des graphes français et celle des graphes américains.** Ce résultat est fort, et permet de confirmer l'intuition des experts sur la différence de topologie entre les deux collections de villes.

D'autre part, les kernels n'ont pas la même performance. Il semblerait que le *shortest-path kernel* soit bien moins utile pour identifier les différences et similarités entre les graphes. Ce résultat ne nous étonne que peu. Dans la partie qui étudiait ce kernel, nous avions remarqué que les résultats du kernel étaient très corrélés à la taille et l'ordre des graphes. Comme la taille et l'ordre des graphes américains ne se distinguent pas clairement de ceux des graphes français, ce kernel ne permet pas de classifier avec une grande précision. Ainsi nos doutes se confirment : cette corrélation empêche le kernel de capturer uniquement l'information *topologie* qui nous intéresse. Par contre, pour les deux *all connected 3,4,5-graphlets kernel*, les résultats sont impressionnantes. Les variances des précisions sont faibles, et les moyennes des précisions montrent que la classification est très satisfaisante. On peut voir que la modification s'avère utile sur ce problème, augmentant d'environ 5% la précision en moyenne. Nous pouvons enfin énoncer sereinement le résultat que l'on cherchait à montrer : **le *all connected 3,4,5-graphlets kernel* est un outil adapté à l'étude de la topologie des villes.**

Pour perfectionner notre analyse, nous allons aussi construire une courbe ROC pour un classifieur issu de chaque kernel. Cela va nous permettre de comparer de manière plus complète les classificateurs issus des kernels. La Figure 12 présente les 3 courbes ROC superposées sur le même graphique, ainsi que l'aire sous chacune des 3 courbes.

Nos résultats précédents sont confirmés par cette Figure. L'aire sous la courbe est meilleure pour les 2 *all connected 3,4,5-graphlets kernel*, avec respectivement 0.87 et 0.84 pour le modifié et le non-modifié. Ces nombres, proches de 0.90, montrent bien que la classification est excellente. L'allure des deux courbes est bien celle que l'on cherche aussi. Pour le *shortest-path kernel*, les résultats sont un peu moins bons, certes, mais l'aire sous la courbe

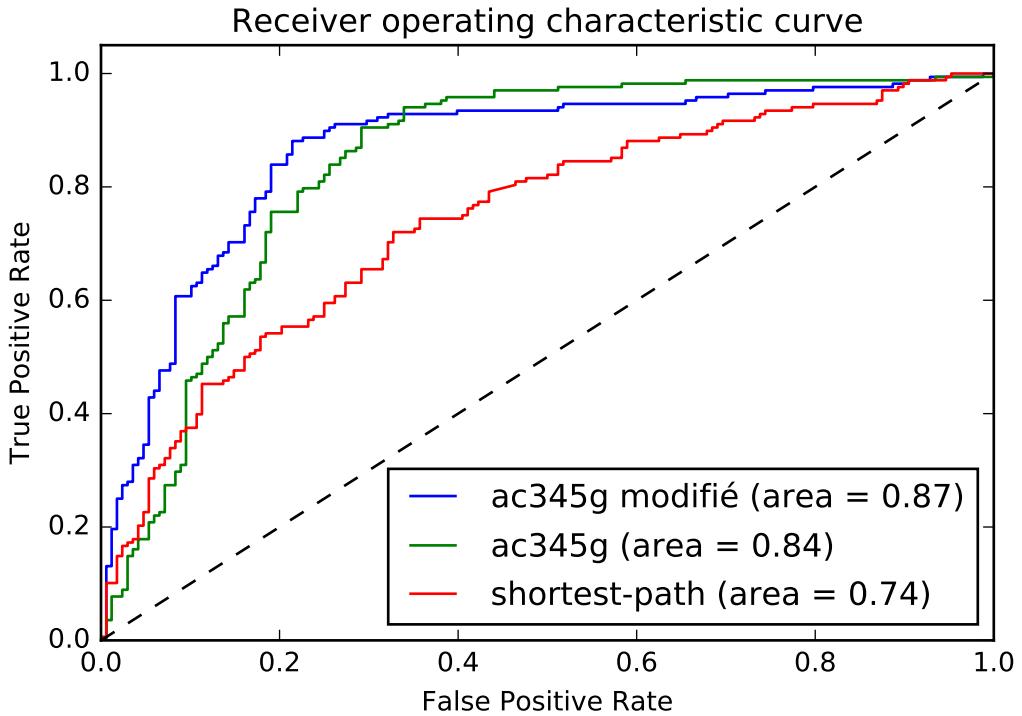


FIGURE 12: Courbes ROC obtenus avec les différents kernels

(0.74) et l'allure de la courbe montrent quand même que la classification est assez bonne. Même si ce kernel ne prend pas uniquement en compte l'information *topologie*, il semble bien qu'il soit tout de même corrélé à cette information.

8 Conclusion

8.1 Réponse à la problématique

Finalement, nous pouvons dire que nous avons répondu à notre problématique, qui était : "**Comment étudier la topologie des villes en s'appuyant sur les graphes qu'elles forment ?**"

Nous avons proposé une modélisation de la topologie des villes via les graphes. De plus, nous avons apporté un cadre de travail pour la comparaison des topologies des villes : les kernels sur les graphes. Nous avons surmonté les difficultés techniques et informatiques liées à l'exploitation de données complexes comme les graphes (temps de calcul, mémoire). Nous avons énoncé le résultats suivant : les kernels que nous avons proposé, et notamment le *all connected 3,4,5-graphlets kernel*, sont capables de résumer l'information *topologie* contenue dans les graphes des villes. En s'appuyant sur un cadre mathématique et statistique rigoureux, nous avons pu prouver ce résultat, **qualitativement et quantitativement**.

8.2 Un pas vers une meilleure compréhension des villes ?

Les résultats prouvés dans ce rapport ne sont pas des découvertes pour les experts du milieu urbain, comme les géographes et les urbanistes. La différence de topologie entre les villes américaines et les villes françaises est un fait connu, qui s'explique par l'Histoire. Ce n'est pas ce qui compte dans mon travail. L'important est d'avoir apporté un cadre de travail susceptible d'aider des experts du milieu urbain à mieux comprendre la topologie des villes. D'une manière ou d'une autre, cet outil pourrait permettre d'aider à répondre à des problématiques liées à la topologie des villes. **Les kernels proposés dans ce rapport constituent une nouvelle manière d'exploiter les données des villes, en l'occurrence leurs graphes.** J'ai l'intime conviction que c'est le but d'un chercheur : essayer de créer/découvrir quelque chose qui puisse servir à quelque chose/quelqu'un.

9 Ouverture

9.1 Weisfeiler-Lehman kernel

Dans ce rapport, je n'ai pas inclus toute une partie de mon travail, qui portait sur un autre type de kernel : les *Weisfeiler-Lehman kernels*. Mon travail s'appuyait sur un article très récent (2011) de Shervashidze et Al. ([2]).

9.1.1 Principe

Shervashidze et Al. ont créé une nouvelle famille de kernels sur les graphes. Cette famille est basée sur le test d'isomorphisme de Weisfeiler-Lehman. Ce test permet de comparer deux graphes $(G_{i,0}, G_{j,0})$, afin de pouvoir tester s'ils sont isomorphes ou non, c'est-à-dire identiques à un changement de nom de noeuds près. Pour ce faire, à chaque itération on transforme les deux graphes et on les compare. Ainsi cela crée deux séquences de graphes $((G_{i,1}, G_{i,h}), (G_{j,1}, G_{j,h}))$. Le *framework* proposé est donc le suivant : lorsqu'on a un kernel k (le kernel de base), on va l'appliquer à $G_{i,l}$ et $G_{j,l}$, $\forall l \in [|0..n|]$, puis la valeur du kernel k à la sauce Weisfeiler-Lehman (qu'on notera ${}^W k$) est simplement la somme des valeurs des kernels. Autrement dit, pour h itérations :

$${}^W k(G_{i,0}, G_{j,0}, h) = \sum_{l=0}^h k(G_{i,l}, G_{j,l})$$

Tout réside donc dans le test d'isomorphisme de Weisfeiler-Lehman. Celui-ci est assez complexe et long à expliquer : il s'agit de modifier, à chaque itération, les labels des noeuds afin de converger vers des labels identiques. Cela demande donc aux kernels de base qu'ils s'appliquent aux graphes qui ont des labels sur les noeuds, ce qui est un peu spécifique et s'applique pas directement à notre problème. Cependant même pour un graphe simple, on peut toujours prendre les degrés des noeuds du graphe comme label. Il faut aussi garder en tête que ce test ne marche pas tout le temps : certaines paires de graphes ne peuvent pas être reconnues comme isomorphiques alors qu'elles le sont. En effet le problème d'isomorphie des graphes est NP-complet, il n'existe pas d'algorithme qui puisse le résoudre en un temps polynomial. Malgré cela, ce test s'avère tout de même très bon, et permet de créer ce cadre de travail pour les kernels, qui donne de très bons résultats ([2]).

J'ai codé entièrement ce kernel sur Python, et je l'ai appliqué dans forme la plus simple : en prenant les degrés des noeuds comme labels. On peut trouver les liens GitHub qui donnent les codes de ce kernel dans l'annexe B.

9.1.2 Un cadre de travail à exploiter

Malheureusement, je n'ai pas eu assez de temps pour exploiter correctement ce cadre de travail. J'ai obtenu quelques résultats intéressants avec le kernel que j'ai implémenté sur Python, notamment des bonnes précisions sur le problème d'application en utilisant les coordonnées des 3 premiers axes de l'ACP kernélisée comme données et Random Forest comme algorithme. Mais je considère que ma recherche n'était pas assez approfondi pour qu'il fasse partie du corps de mon rapport. Ainsi, cela laisse une voie ouverte pour continuer cette étude.

9.2 Autres améliorations possibles

L'amélioration naturelle à faire à mon travail est d'utiliser la première méthodologie (cf partie *Choix de la méthodologie*) pour étudier la topologie des villes. Il est bien possible que les résultats soient meilleurs, en tout cas il est nécessaire de le faire afin de donner un avis complet sur la topologie des graphes des villes.

D'autre part, il existe probablement des kernels qui peuvent encore mieux s'adapter à notre problème. Dans la littérature actuelle, je n'en ai pas trouvé mais je pense qu'une recherche plus exhaustive et surtout des contacts parmi les chercheurs du milieu pourrait permettre de découvrir de nouveaux kernels. Autrement, il est aussi possible de créer un kernel. Cependant, ce travail est plus proche d'une thèse que d'un stage d'application, mais il permettrait de vraiment faire avancer l'état de l'art, et aurait sûrement des applications dans beaucoup d'autres domaines. La chercheuse française N. Shervashidze semble être une pointure du domaine, il peut par exemple être intéressant de la contacter. Les kernels sur les graphes reste un sujet d'actualité récente, pour travailler dessus il faut donc se retrousser les manches : vous ne trouverez pas une myriade d'articles (scientifiques ou de blog quasi-scientifiques) qui vous donneront des méthodes toutes faites pour les utiliser ou les choisir, contrairement à la plupart des applications en machine learning/deep learning. Ainsi je pense que les améliorations à apporter à mon travail, et globalement au sujet des kernels sur les graphes, sont réservées à la recherche.

A Références

Références

- [1] T. Gärtner, P. Flach, and S. Wrobel, “On graph kernels : Hardness results and efficient alternatives,” in *Learning Theory and Kernel Machines*, pp. 129–143, Springer, 2003.
- [2] N. Shervashidze, P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, and K. M. Borgwardt, “Weisfeiler-lehman graph kernels,” *Journal of Machine Learning Research*, vol. 12, no. Sep, pp. 2539–2561, 2011.
- [3] K. M. Borgwardt and H.-P. Kriegel, “Shortest-path kernels on graphs,” 2005.
- [4] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. M. Borgwardt, “Efficient graphlet kernels for large graph comparison.,” in *AISTATS*, vol. 5, pp. 488–495, 2009.
- [5] 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*, vol. 12, pp. 2825–2830, 2011.

B Liens GitHub

Tout les codes des 4 types de kernels utilisés dans le stage :

- Random-walk kernel : https://github.com/Caselles/graph_kernels/blob/master/random_walk_kernel.py
- Shortest-path kernel : https://github.com/Caselles/graph_kernels/blob/master/shortest_path_kernel.py
- Tout les k-graphlets kernels cités : https://github.com/Caselles/graph_kernels/blob/master/graphlet_kernels.py
- Weisfeiler-Lehman subtree kernel : https://github.com/Caselles/graph_kernels/blob/master/WL_subtree_kernel.py

C Annexes

C.1 Variété de dimension 3 issue de l'ACP kernélisée

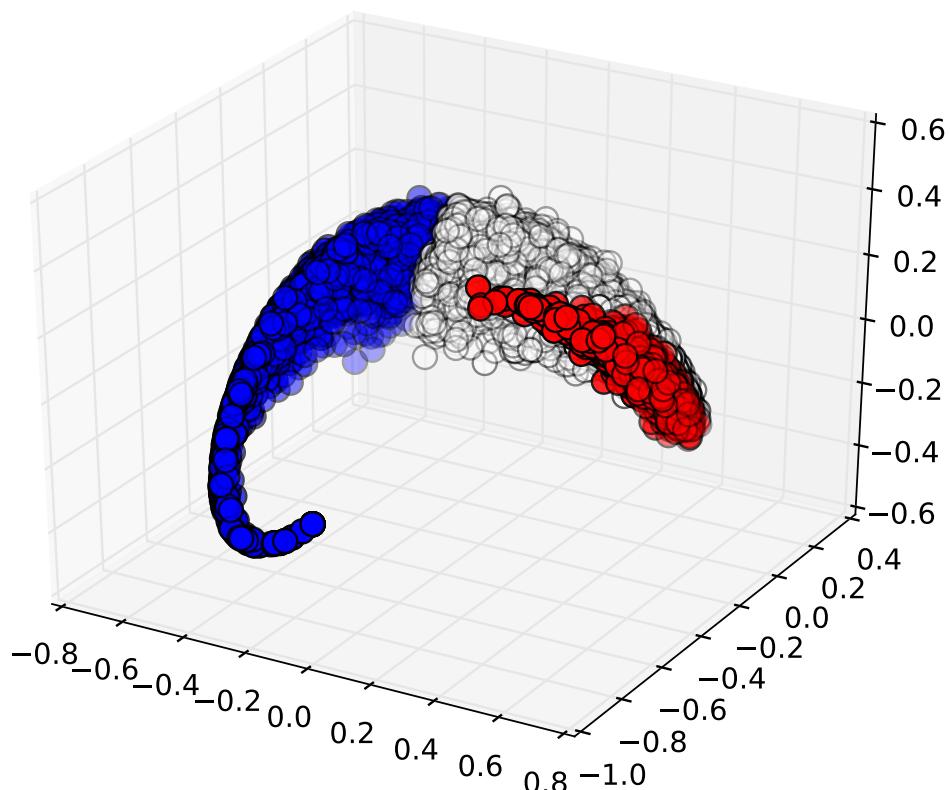


FIGURE 13: ACP kernélisée : 3 premiers axes.

C.2 Graphe de Malakoff (92)



FIGURE 14: Graphe de Malakoff (92) simplifié



SYNTHESIS NOTE OF 2nd YEAR INTERNSHIP

A quantitative study of cities topology using kernels on graphs

Stagiaire :

Hugo DUPRÉ
3rd year student
ENSAE ParisTech

Maître de stage :

Fabien PFAENDER
Researcher and coordinator
ComplexCity Lab

Shanghai, China
June 6, 2016 - August 12, 2016

This summer, I worked as a research intern in Shanghai, in a sino-french research laboratory called ComplexCity Lab. This laboratory focuses its researches on the study of cities, considered as complex systems.

With the help of my supervisor, Fabien Pfaender, I studied cities topology, from a data driven point of view. It is relevant to study this topic from my data scientist point of view, since it can allow geographers and urbanists to better understand cities and their spatial organization, using a different point of view. As a data scientist, I should be able to help experts from other domains by offering my expertise in applied mathematics and statistics.

I tried to model cities by the graphs of their streets and routes, and we collected the data using OpenStreetMap, an open source software that gives maps for the whole world. We manage to get the graphs of every french and american city in the right format. I had to simplify those graphs, because they were a lot of unnecessary information in them. Figure 1 shows a simplification example of the graph of Saint-Paul de La Réunion.

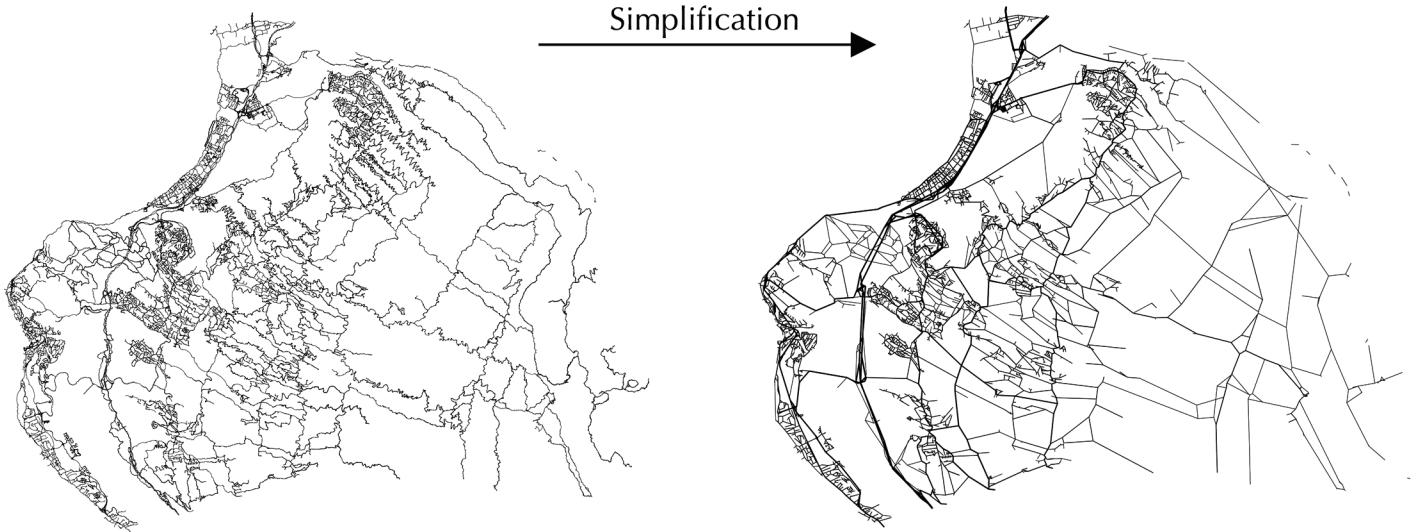


FIGURE 1: Simplification d'une ville très sinuose : Saint-Paul de la Réunion.

I was given the task of comparing and analyzing these graphs. My internship lasted 2 and a half months, and I was able to deliver a framework for comparing the topology of graphs representing cities. I spent a lot of time trying to find functions that would allow to compare the topology of these graphs representing real cities. These functions are called kernels, and I found a bunch of them that seemed well suited for my problem. Then, using several algorithms and throughout an application problem, I collected some results that were supposed to prove that the functions I had chosen were in fact the right functions to compare graphs' topology of cities. The application was a learning problem : predicting the country (USA or France) of a graph using examples of others graphs of the two countries. I implemented these algorithms on Python, using different kernel functions I found or created. I managed to get a 83% accuracy rate on this application problem, using the best kernel I had, which allowed me to prove my point.

In table 1 one can find the results obtained with 3 different kernels.

TABLE 1: Résultats de la classification avec les différents kernels

Type de kernel	Précision moyenne	Variance de la précision
Shortest-path	66%	0.031
All connected 3,4,5-graphlets	78%	0.026
All connected 3,4,5-graphlets modifié	83%	0.027

My work can be, and should be, continued since there must be some other kernel functions out there that would allow to better understand cities topology, which is essentially the goal of my work.



NOTE DE SYNTHÈSE DE STAGE D'APPLICATION DE 2^E ANNÉE

Étude de la topologie des villes à l'aide de kernels sur les graphes

Stagiaire :

Hugo DUPRÉ
Étudiant en 3^e année
ENSAE ParisTech

Maître de stage :

Fabien PFAENDER
Chercheur et coordinateur
ComplexCity Lab

Shanghai, Chine
6 Juin 2016 - 12 Août 2016

Cet été, j'ai effectué un stage de recherche dans le laboratoire sino-français ComplexCity Lab. Ce laboratoire étudie les villes, notamment en les modélisant comme des systèmes complexes.

Avec l'aide de mon maître de stage, Fabien Pfaender, j'ai étudié la topologie des villes, en utilisant des données réelles. Il est intéressant d'étudier la topologie des villes de ce point de vue, car cela permet aux géographes et urbanistes de mieux comprendre les villes et leur organisation spatiale. En tant que data scientist, ma mission est d'aider les experts d'autres domaines à l'aide de mon expertise en mathématiques appliquées et en statistique.

J'ai modélisé les villes par les graphes de leurs rues et routes, et Nous avons pu récupérer les données à l'aide d'OpenStreetMap, un logiciel open-source qui propose des cartes pour le monde entier. Nous avons réussi à obtenir les graphes de toutes les villes françaises et américaines, dans le bon format. Il a fallu simplifier ces graphes, car beaucoup d'entre eux comportaient de l'information non-nécessaire. La figure 1 montre un exemple de simplification d'une modélisation par un graphe de la ville de Saint-Paul de La Réunion.

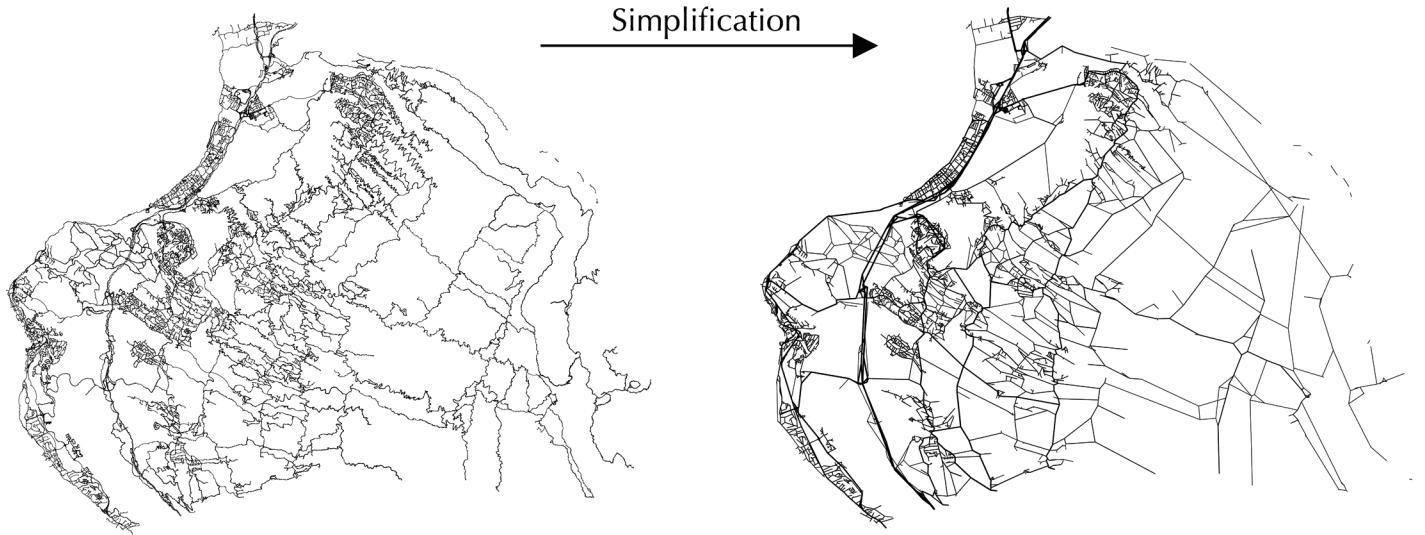


FIGURE 1: Simplification d'une ville très sinueuse : Saint-Paul de la Réunion.

La mission qu'on m'a donc proposé était de comparer et analyser ces graphes. Mon stage a duré 2 mois et demi, et j'ai pu proposer un cadre de travail pour comparer la topologie des graphes qui représentent les villes. J'ai passé beaucoup de temps à chercher des fonctions qui permettent de comparer la topologie de ces graphes qui modélisent les villes. Ces fonctions sont appelées kernels, et j'en ai trouvé plusieurs qui s'appliquaient bien à ma problématique. Ensuite, en utilisant de nombreux algorithmes et à travers un problème d'application, j'ai obtenu des résultats qui étaient censé prouver que les fonctions que j'avais choisi étaient en effet adaptées à la comparaison des topologies des villes. Le problème d'application consistait en de l'apprentissage supervisé : prédire le pays (Etats-Unis ou France) de graphes en apprenant avec d'autres exemples de graphes des deux pays. J'ai implémenté ces algorithmes sur Python, en utilisant différents kernels que j'avais trouvé ou créé. J'ai réussi à obtenir une précision de prédiction de 83% sur ce problème d'application en utilisant mon meilleur kernel, ce qui m'a permis de prouver ce que je voulais démontrer. La table 1 donne les résultats obtenus avec 3 différents kernels que j'ai utilisé.

TABLE 1: Résultats de la classification avec les différents kernels

Type de kernel	Précision moyenne	Variance de la précision
Shortest-path	66%	0.031
All connected 3,4,5-graphlets	78%	0.026
All connected 3,4,5-graphlets modifié	83%	0.027

Mon travail peut, et doit, être poursuivi puisqu'il doit exister d'autres kernels qui pourraient permettre de mieux comprendre la topologie des villes, ce qui est mon but final.