MASTER EN SCIENCES MATHÉMATIQUES

Analyses et prédictions de déplacements de population à New York et Tokyo basé sur les propriétés des réseaux

Delmotte, Charles

Award date:
2018

Awarding institution:
Universite de Namur

UNIVERSITE DE NAMUR

Faculté des Sciences

ANALYSES ET PREDICTIONS DE DEPLACEMENTS DE POPULATION A NEW YORK ET TOKYO BASE SUR LES PROPRIETES DES RESEAUX

Mémoire présenté pour l'obtention

du grade académique de master en didactique des mathématiques

Charles DELMOTTE

Août 2018

Université de Namur

Bloc de master en Sciences Mathématiques

Analyses et prédictions de déplacements de population à New York et Tokyo basé sur les propriétés des réseaux

Promoteur : R.Lambiotte

Charles DELMOTTE

Année académique 2017-2018

Remerciement

Je remercie mes parents, mon frère et ma compagne pour le soutien et l'aide apporté durant ce mémoire et M. Lambiotte de m'avoir aiguillé.

Résumé

Ce travail étudie les mouvements de population en se basant sur les données Foursquare de Tokyo et de New York.

Afin d'arriver à ce résultat, nous avons étudié les propriétés des réseaux locaux et avons ensuite sélectionné des algorithmes se servant notamment de ces propriétés.

Ensuite nous avons étudié le comportement des utilisateurs afin d'étudier nos données et vérifier la correspondance de nos données avec les propriétés.

Enfin après avoir adapté notre graphe, nous avons utilisé des algorithmes utilisant l'indicateur d'AdamicAdaR, le coefficient de Jaccard, l'index d'allocation de ressources , le score d'attachement préférentiel et le spectre du graphe Laplacien normalisé afin de prédire les arêtes de notre graphe correspondant à un lien entre deux zones délimitées par leurs longitude et latitude.

Table des matières

Introduction

La société actuelle se développe de plus en plus autour de l'informatique et d'internet. La 3G puis 4G se développent et de plus en plus d'utilisateurs font leur apparition. Avec ces technologies, de plus en plus de gens participent à divers programmes en vue de critiquer ou d'informer les autres sur leur environnement. Grâce à cela certaines applications collectent des données sur des lieux, des villes et des pays. Gràce à ce développement l'application Foursquare s'est développée dans certaines grandes villes et a connu un certains succès dans certaines d'entre elles.

D'un autre côté la recherche s'est intéressée de plus en plus au Web 2.0, essayant de tracer des graphes avec les amis et les relations entre les personnes. On établit d'ailleurs un lien entre ces graphes et la localisation et on essaye de trouver des méthodes afin de prédire quels seront les futurs amis ou de compléter les graphes incomplets à l'aide d'algorithmes de prédiction d'arête.[6],[23]

Avec ce dévelopement informatique et cette recherche sur les réseaux sociaux, il m'a donc semblé intéressant de vouloir prédire les mouvements de population à l'aide des données recueillies à l'aide de cette application en utilisant des algorithmes utilisés principalement pour les réseaux sociaux à l'heure actuelle en les adaptant afin qu'ils fonctionnent pour les réseaux locaux.

Nous allons donc au travers de ce travail utiliser des algorithmes afin de prédire les mouvements de population dans différentes villes telles que New York ou Tokyo.

Dans un premier temps, nous allons étudier les caractéristiques des réseaux locaux. Nous allons donc analyser la provenance de nos données, définir ce que c'est un réseau local et voir comment il évolue en théorie.
Ensuite nous allons étudier les propriétés de ces réseaux telles que la distribution des degrés des noeuds, la fermeture triadique, la propriété small-world et l'assortativité.
Nous allons nous intéresser à cinq méthodes de prédiction d'arêtes, trois méthodes basées sur la fermeture triadique, une sur la mesure de centralité et une sur Laplacien.
De plus afin d'évaluer ces méthodes, nous allons définir trois métriques : la ROC, la AUC et

la précision moyenne.

Après ce cadre théorique nous allons analyser nos données.
Dans un premier temps nous allons anlayser nos données afin de voir le type d'endroits fréquentés par les utilisateurs et ayant le plus de trafic entrant et sortant, nous allons ensuite analyser la distribution des endroits en fonction de leur type, le nombre d'entrées par endroit et les activités des utilisateurs.
Nous allons ensuite dans la suite de ce chapitre à vérifier certaines des propriétés citées dans le cadre théorique ; nous allons donc nous intéresser à la distribution des noeuds en fonction de leur degré,des arêtes en fonction de leur poids, s'intéresser à la probabilité de découvrir un nouvel endroit, à la persistance des arêtes ainsi qu'à celle des noeuds.
Enfin nous allons remodeler nos données afin que nos algorithmes de prédiction nous fournissent de meilleurs résultats.

Enfin pour terminer nous allons appliquer nos cinq méthodes de prédictions d'arêtes à nos données remodelées afin de pouvoir analyser si nos algorithmes sont en mesure de prédire les mouvements de population.
Les cinq méthodes que nous utiliserons sont : le coefficient de Jaccard, l'index d'allocation de ressources, l'indicateur AdamicAdar, le score d'attachement préférentiel et la prédiction d'arête via le spectre du graphe Laplacien normal à l'aide du machine learning.

1 Théorie sur les réseaux locaux

Avant de nous lancer dans l'analyse de nos données et dans la prédiction de nos arêtes, nous allons définir ce qu'est un réseau local et ce qu'est l'application Foursquare comme nos données en sont la source.

1.1 Caractérisation de l'application Foursquare et définitions, développement et dynamique des réseaux locaux urbains

Nous définirons dans cette partie ce qu'est un réseau local urbain et analyserons l'accroissement et la dynamique du réseau local.

1.1.1 Caractérisation de l'application Foursquare

Foursquare, comme nous le présente le site officiel [7], est une société de technologie qui, à l'aide des données de localisation, aide les utilisateurs à découvrir des nouveaux endroits et ce grâce aux recommandations précédentes d'autres utilisateurs.
Cette application contient plus de 65 millions d'endroits et plus de 50 millions d'utilisateurs nous permettant ainsi d'avoir un échantillon assez large en utilisant seulement les utilisateurs de la ville de New York.
Enfin, en ce qui concerne son utilisation, comme nous l'apprend [25], l'utilisateur effectue un "check-in" dans un lieu donné. Cela revient à recommander un restaurant, une exposition ou même une entreprise. Cette application rappelle d'une certaine manière l'utilisation de TripAdvisor. Le deuxième but du "check-in" consiste à devenir ce qu'on appelle le maire d'un lieu. Cela signifie que lorsque vous "checkez" dans un lieu précis, vous pouvez devenir le maire de ce lieu, bien que l'algorithme permettant de le devenir reste mystérieux : plus on se localise dans un endroit précis, plus on a de chance de devenir le maire de ce lieu. Le deuxième but est donc d'amener les gens à utiliser cette application à l'aide de ce mini-jeu.

1.1.2 Définition des réseaux locaux urbains

Un réseau local urbain est un réseau dans lequel les noeuds sont les endroits les plus populaires au sein d'une ville. Nous définirons un réseau local urbain de la même manière que l'article [21] nous le définit. Pour chaque ville, nous considérerons une période de temps finie t et construirons un graphe G^t comprenant un ensemble de noeuds V^t et un ensemble d'arêtes E^t. Une arête est formée de deux noeuds sur lesquels un utilisateur de l'application Foursquare a transité directement de l'un à l'autre sur une période de temps t. Lorsqu'un utilisateur transite directement d'un noeud à un autre sur une période de temps t, cela veut dire que cet utilisateur a utilisé l'application Foursquare dans un premier lieu, et que son utilisation suivante de l'application a eu lieu dans un autre lieu. Nous augmenterons le poids d'une arête proportionnellement au nombre d'utilisateurs utilisant cette même arête.

1.1.3 Modèle d'accroissement des réseaux et dynamique temporelle

La densification du réseau est primordiale dans le travail qui va suivre et il est essentiel de comprendre son accroissement. On a pu remarquer que l'ajout des noeuds et des arêtes n'est pas constant et varie en fonction du temps. Il existe une formule empirique pour caractériser cette procédure de densification du réseau, qui montre que le nombre d'arêtes augmente superlinéairement avec le nombre de noeuds.[16]
Cela signifie que si on prend n(t) comme le nombre de noeuds observés au temps t et e(t) comme le nombre d'arêtes à ce même temps t, on a

$$e(t) \propto n(t)^{\alpha}$$

où α peut valoir différentes valeurs. Par exemple pour α valant 1, on a un nombre de degrés stable au fur et à mesure du temps, par contre si $\alpha > 1$, le nombre de degrés va augmenter au fur et à mesure du temps. Cependant, dans beaucoup de réseaux du monde, il semblerait que $\alpha > 1$.

Dans la figure 1, on peut voir le nombre d'arêtes en fonction du nombre de noeuds de la ville tiré de l'article [21]. Nous sommes partis d'un temps t_0 choisi au hasard et avons ensuite observé l'évolution du nombre de noeuds et d'arêtes dans une centaine de réseaux locaux urbains. Nous pouvons ainsi voir que le nombre d'arêtes augmente superlinéairement avec le

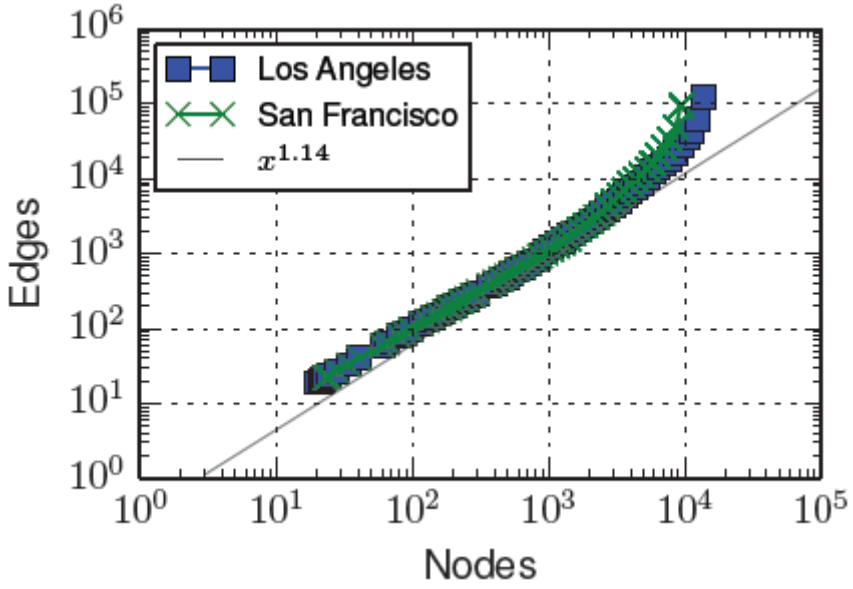

FIGURE 1 – Nombre d'arêtes en fonction du nombre de noeuds de Los Angeles et de San Francisco via les données des utilisateurs de l'application Foursquare. Source : [21]

nombre de noeuds et ensuite, nous avons mesuré α par un calcul d'optimisation du moindre carré et avons ainsi obtenu qu'il valait 1.14 avec un écart-type de ± 0.06. Dans le futur, nous comparerons cette valeur avec le graphique tiré de nos données. Il en va de même pour les graphiques qui vont suivre.

Cependant, à partir d'un certain moment, il devient difficile de découvrir le nombre d'endroits via l'application Foursquare. En d'autres mots, la probabilité de découvrir un nouvel endroit avec l'application Foursquare est peu élevée. En effet, bien qu'il existera toujours de nouveaux endroits à visiter dans une ville, le nombre d'endroits encore à trouver augmentera toujours de façon significativement plus lente que le taux de découverte de nouveaux endroits par les utilisateurs de Foursquare. La figure 2 nous montre cela et bien que de nouvelles arêtes continuent toujours à apparaître de façon superlinéaire comme nous le montre la figure 1, les nouveaux noeuds eux apparaissent de moins en moins souvent à cause de ce que l'on appellera un effet de dimension finie.

Considérant que $\alpha > 1$, on sait que le nombre de degrés va augmenter ainsi que le nombre d'arêtes et de noeuds. Il serait intéressant de savoir si les arêtes et les noeuds persistent au fur et à mesure du temps. Or, on sait que la probabilité d'avoir une nouvelle arête s'obtient comme suit

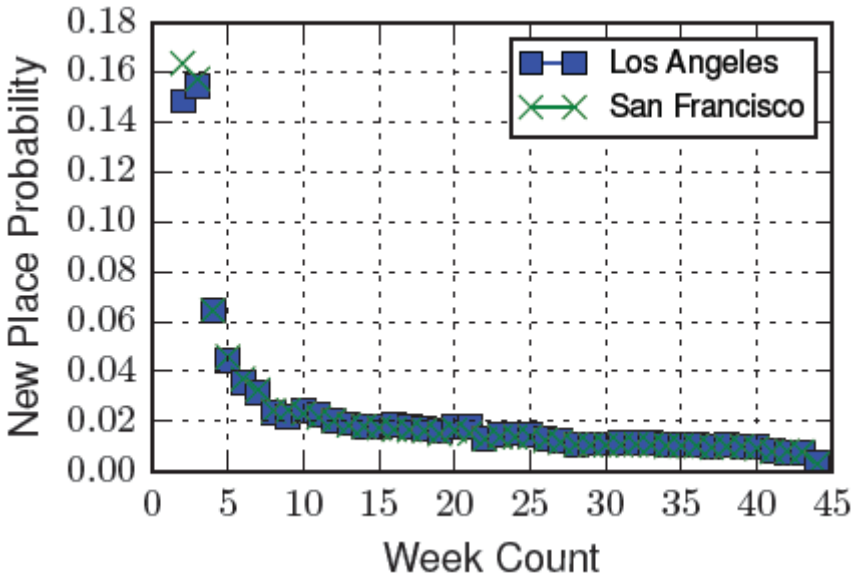

FIGURE 2 – Probabilité de découvrir un nouvel endroit sur le total d'endroits trouvés en fonction du nombre de semaines pour Los Angeles et San Francisco et ce à partir de la deuxième semaine. Source : [21]

$$P_e = \frac{|E^{t+1} - E^t|}{|E^{t+1}|}$$

où E^t est l'ensemble des arêtes au temps t et E^{t+1} est l'ensemble des arêtes au temps suivant. La figure 3 nous montre que cette probabilité est valable aux 7 points sur les graphiques qui correspondent à l'intersection des périodes de temps. On notera que la probabilité sur ces points vaut environ 70-75 % et a un petit écart-type. Or on sait aussi que la probabilité qu'une arête apparue en t réapparaisse en t+1, t+2, t+3... t+n se calcule comme suit :

$$P_{e,n} = \frac{|E^t \cap E^{t+1} \cap E^{t+2} \cap E^{t+3} ... \cap E^{t+n}|}{|E^t|}.$$

Cela correspond à ce que l'on appellera la longévité d'une arête. Comme nous le montre la figure 4, cette probabilité se situe aux alentours de 30 % et diminue vers les 20 % dans les temps suivants. On peut donc voir que 20 % des arêtes perdurent au travers du temps et correspondent vraisemblablement aux arêtes ayant un poids élevé.

La probabilité qu'une arête réapparaisse et reste dans le réseau est directement reliée au

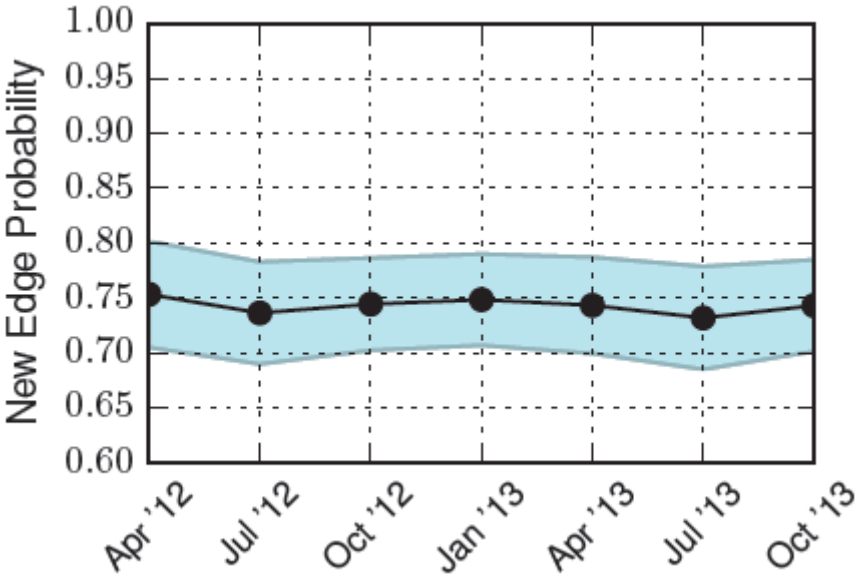

FIGURE 3 – Probabilité d'apparition d'une nouvelle arête. La zone bleue correpond à l'écart-type à travers une centaine de villes. Cette probabilité est mesurée en comparant des périodes de temps différentes, celles-ci correpondent à l'espace(période de temps) entre deux points successifs d'abscisse . Source : [21]

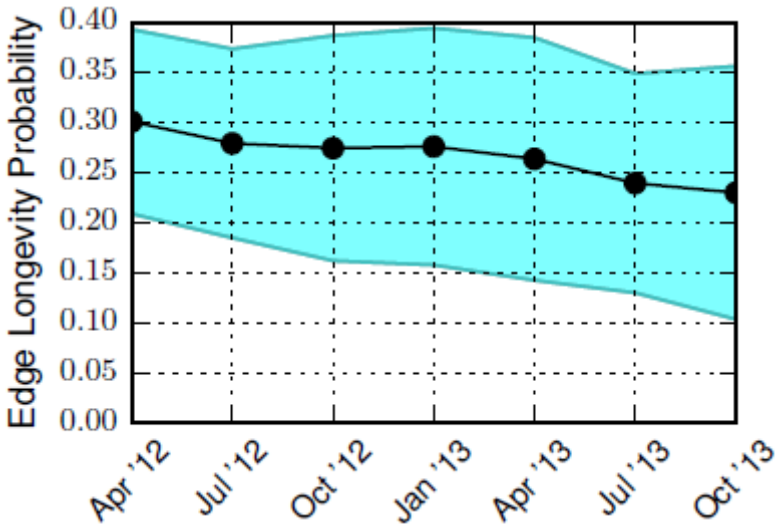

FIGURE 4 – Probabilité qu'une arête déjà présente dans les temps précédents réapparaisse dans le temps suivant. Nous appelerons cela la longévité d'une arête. Source : [21]

poids de cette arête. En effet, il est logique de voir que plus une arête est utilisée, plus elle a de chances de réapparaître. Définissons donc la fonction w(e) comme étant le poids relatif à une arête e de l'ensemble des arêtes E. On a alors la probabilité qu'une arête persiste comme étant

$$P_e(w) = \frac{|\{e \in E^t \cap E^{t+1} : w(e) \geq w\}|}{|\{e \in E^t : w(e) > w\}|}$$

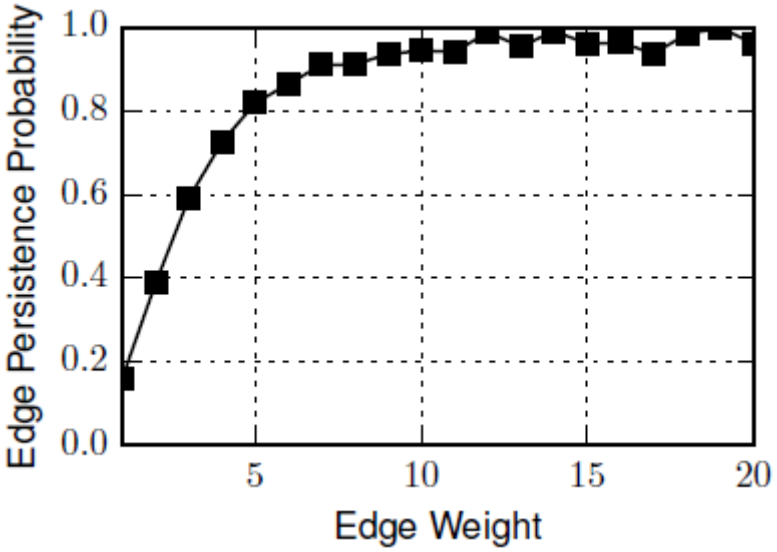

FIGURE 5 – Probabilité qu'une arête déjà présente dans les temps précédents réapparaisse dans le temps suivant en fonction du poids de cet arête. Source : [21]

On peut voir dans la figure 5 que la longévité d'une arête est directement liée à son poids. Plus l'arête a donc de poids, plus elle est susceptible d'apparaitre lors des temps suivants. On peut voir dans cette figure que le comportement du graphique est logarithmique et donc lorsque la valeur de poids est égale à 2, la probabilité qu'il persiste est proche de O.4, mais il faut seulement un poids de 5 pour obtenir le double de cette probabilité. Cependant on peut voir qu'à partir d'un poids de 12, on obtient des fluctuations dans cette courbe ; cela peut s'expliquer par exemple par des travaux sur un axe rendant le trajet de l'un à l'autre plus difficile ou de larges mouvements de foules qui vont ainsi perturber le trafic et donc ainsi affecter la réapparition d'une arête.

Comme nous l'avons fait pour les arêtes, nous allons maintenant voir comment un noeud persiste au fur et à mesure du temps. Nous allons donc définir la probabilité d'avoir un nouveau noeud, comme on l'a fait pour les arêtes. Et donc on obtient

$$P_u = \frac{|V^{t+1} - V^t|}{|V^{t+1}|}$$

où V^t est l'ensemble des noeuds sur une période t. De plus, on peut voir que cette proba-

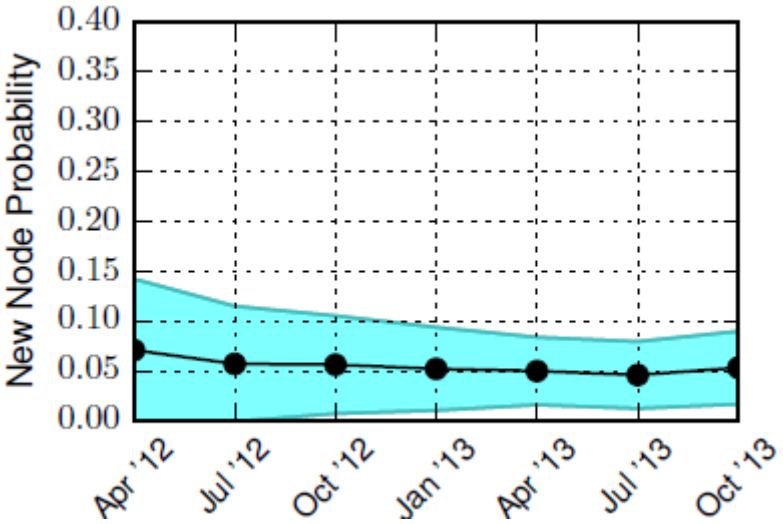

FIGURE 6 – Probabilité d'obtenir un nouveau noeud lors de la période de temps suivante. Source : [21]

bilité est de l'ordre de 5 % comme nous le montre la figure 6. Ce graphique suggère donc que la base d'une ville se réduit à un nombre très restreint de noeuds et que la création de nouveaux noeuds vient des habitudes des différents personnes.

Nous allons maintenant parler de la longévité des noeuds que nous appellerons ici la persistance des noeuds. La probabilité de persistance(longévité) des noeuds se définit d'une manière très semblable à celle des arêtes et se traduit donc comme suit

$$P_{v,n} = \frac{|V^t \cap V^{t+1} \cap ... \cap V^{t+n}|}{|V^t|}$$

Comme nous le montre la figure 7, les noeuds qui persistent ne subsistent en général pas sur une petite période de temps, mais sur plusieurs de celles-ci. En effet, on peut voir que même après un an et demi, il reste encore 90 % des noeuds qui étaient présents à la base.

1.1.4 Conclusion sur la dynamique des réseaux locaux

Si l'on considère tous ces résultats, on peut voir que les réseaux locaux sont des réseaux dans lequels les arêtes changent et évoluent énormément, parfois elles n'existent même que pour de courtes périodes. Si l'on compare cela à des réseaux sociaux, on notera tout de suite

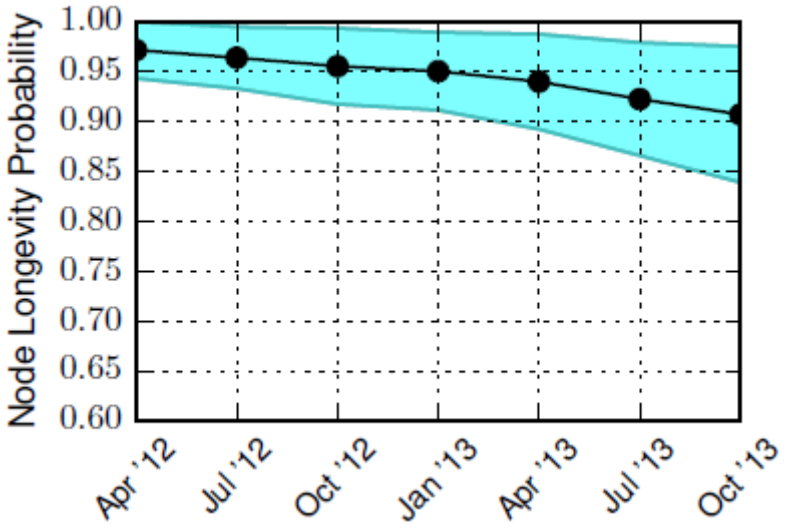

FIGURE 7 – Probabilité de persistance d'un noeud lors de la période de temps suivante. Source : [21]

une différence, car dans ce cas les arêtes (qui correspondent à des liens d'amitié) ont tendance à persister pendant de longues périodes de temps. Il semblerait donc intéressant de fournir un modèle de prédiction afin de voir comment évolue cet ensemble d'arêtes et ainsi prédire la création ou la perte de certaines arêtes de ce type de réseau. Pour cela, il va donc falloir mieux comprendre comment fonctionnent les réseaux locaux afin de créer de meilleurs services de localisation capables de prévoir et de réagir à ces changements. Il semble donc intéressant d'étudier leurs propriétés topologiques afin de fournir des modèles améliorés grâce aux propriétés que l'on en tirera.

1.2 Propriétés des réseaux locaux

Maintenant que nous avons étudié ce réseau, il faut définir son modèle de densification. Il semblerait intéressant de voir s'il ressemble à certains autres afin de pouvoir en tirer des algorithmes similaires. Nous allons donc principalement essayer de découvrir les propriétés de ce réseau et voir si elles possèdent un lien quelconque avec les réseaux sociaux. Nous allons établir ces propriétés afin de nous en servir lors de la construction de nos modèles, car elles permettront notamment d'obtenir de meilleurs modèles ou de les simplifier.

1.2.1 Distribution des degrés de type "heavy-tailed"

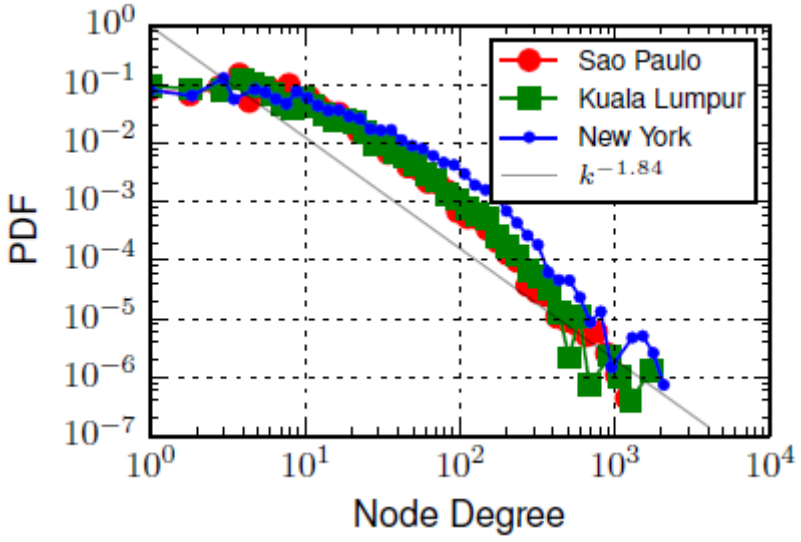

FIGURE 8 – Distribution de la densité de probabilité des degrés des noeuds sur une période de temps pour 3 villes et le comportement linéaire qui leur est associé. Source : [21]

Une des premières propriétés que nous allons aborder est la "heavy-tailed degree distribution". Cette propriété est présente dans de nombreux réseaux associés à notre monde, y compris les réseaux sociaux, elle se caractérise par le fait qu'il existe de nombreux noeuds ayant un degré très faible et que peu de noeuds ont un degré très grand. Elle s'oppose à une loi normale qui comprend peu de noeuds ayant un fort ou un faible degré, mais beaucoup étant de degré moyen (courbe gaussienne). En général, ils suivent une loi de la forme

$$P(k) \propto k^{-\beta}$$

La figure 8 nous montre que les réseaux locaux ont eux aussi une "heavy-tailed degree distribution". Les auteurs, après avoir fait le test du maximum de vraisemblances, ont trouvé, comme valeur de β, 1.84 avec une déviation de 0.09 et ce sur base de cent villes. Les auteurs nous montrent aussi avec la figure 9 que la distribution du poids des arêtes suit davantage cette loi de distribution pour un exposant de -2.65 ± 0.05. En effet, elle nous montre qu'il y a de nombreuses arêtes ayant un poids très faible, mais il existe certaines arêtes parcourues par des centaines d'usagers.

Afin de mieux comprendre le pourquoi de cette propriété, il semble intéressant de savoir à quoi correspondent ces noeuds et ces arêtes dans le monde réel. Pour cela, on peut voir dans

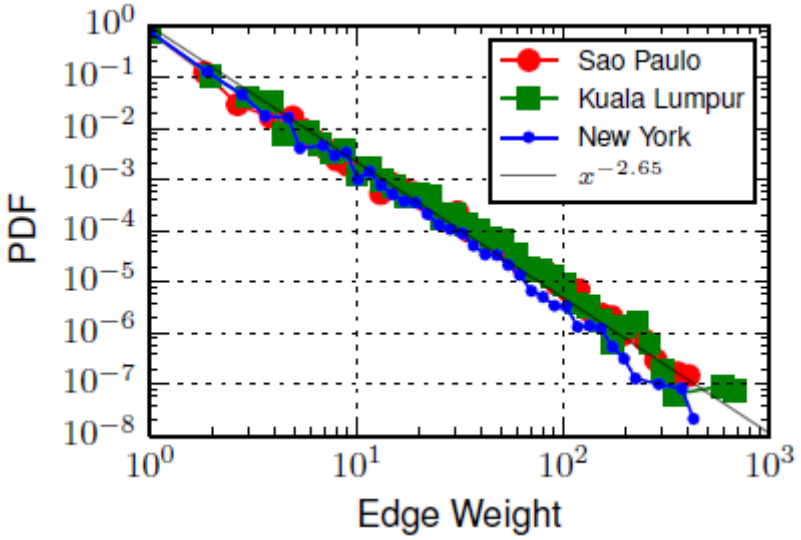

FIGURE 9 – Distribution de la densité de probabilité du poids des arêtes sur une période de temps pour 3 villes et le comportement linéaire qui leur est associé. Source : [21]

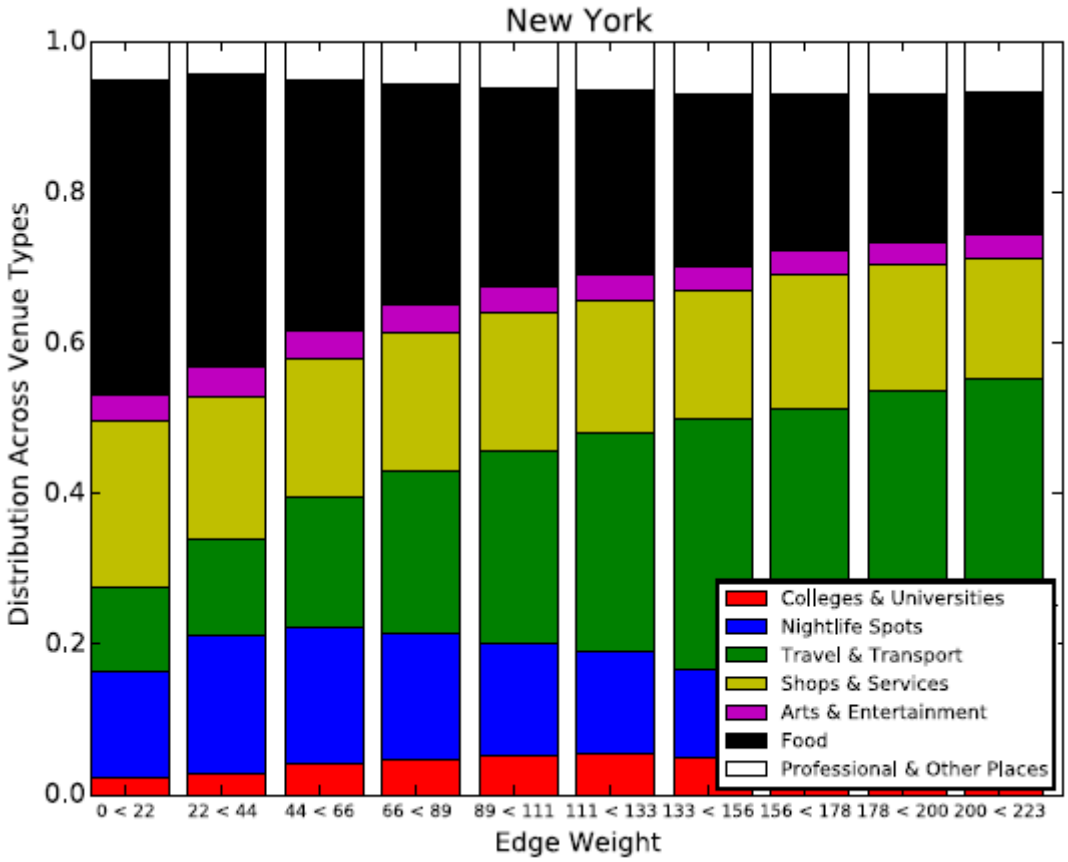

FIGURE 10 – Distribution par catégorie en fonction du poids des noeuds. Source : [21]

la figure 10 la distribution de probabilité par catégorie en fonction du poids des arêtes. Dans un premier temps, on peut voir que la nourriture domine sans conteste pour des arêtes de petit poids. Mais avec le temps, elle est progressivement remplacée par le transport lorsque

l'on prend des arêtes avec de plus grands poids. Il est évident que l'on peut appliquer cela à la réalité par le fait qu'il existe dans les villes de nombreux endroits où manger, mais ceux-ci disposent généralement de peu de places. De plus, dans beaucoup de restaurants, faire la cuisine requiert un certain temps. Et donc le temps nécessaire à la préparation et le peu de places font que dans ces lieux, nous obtenons peu d'entrées et de sorties. De ces graphiques, on peut donc tirer un point clé de notre civilisation actuelle urbaine : car elle se caractérise par des liens de transports ayant un fort poids et par de nombreux endroits où se nourrir.

1.2.2 Fermeture triadique

La fermeture triadique est un des mécanismes centraux dans les réseaux sociaux, c'est le fait que si un lien entre A et B est fort et qu'un lien entre B et C l'est aussi, il y a une grande probabilité qu'un lien faible ou fort naisse entre A et C[3] . D'où l'expression "Les amis de mes amis sont mes amis".
La fermeture triadique est en général mesurée par le coefficient moyen de clustering C

$$C = \frac{1}{|V|} \sum_{u \in V} c_u$$

où c_u est la fraction du nombre de triangles fermés par rapport au nombre de noeuds connectés à u et est appelée "le coefficient local de clustering". Les auteurs ont obtenu un coefficient moyen de clustering de 0.20 avec une déviation standard de 0.06 et ce à travers leurs cent villes étudiées. Ils ont comparé ces résultats à une disposition aléatoire des noeuds et des arêtes afin de voir s'il y avait une grande différence par rapport à ces résultats et ils ont obtenu un coefficient de clustering $C_0 = 0.07 \pm 0.03$ pour une disposition aléatoire, ce qui montre une différence notable entre une disposition aléatoire et la réalité. On peut donc en déduire qu'il y a une disposition triadique.

1.2.3 Propriété small-world

Cependant dans les réseaux locaux, la raison pour laquelle la fermeture triadique serait présente ne peut pas être la même que pour les réseaux sociaux. Mais elle peut probablement

s'expliquer par le fait que les réseaux locaux sont disposés dans l'espace, et donc il peut être intéressant de calculer la distance entre deux noeuds. Les auteurs vont calculer cette distance en termes de sauts pour atteindre ce noeud et vont obtenir une valeur moyenne de plus court chemin de $\overline{d} = 3.35 \pm 0.52$ avec une valeur moyenne de diamètre de $\overline{D} = 6.35 \pm 1.46$, ils vont ensuite la comparer à une version aléatoire comme dans la sous-section précédente et vont obtenir $\overline{d} = 3.33 \pm 0.46$ et $\overline{D} = 5.93 \pm 1.19$. Le fait qu'il existe des plus courts chemins reliant deux noeuds et le fait d'avoir un haut coefficient de clustering impliquent que les réseaux locaux sont considérés comme des small-world. Cette propriété est très importante, car elle aura beaucoup d'implications dans le système urbain, telles que la propagation des maladies ou encore la circulation des rumeurs.

1.2.4 Assortativité

L'assortativité est la tendance d'un noeud à s'associer à un noeud similaire ; dans notre cas, cette similarité s'expliquera en termes de degrés d'un noeud. L'assortativité se calcule via le coefficient de Newman [18]

$$r = \frac{M^{-1}\sum_i j_i k_i - (M^{-1}\sum_i \frac{1}{2}(j_i + k_i))^2}{M^{-1}\sum_i \frac{1}{2}(j_i^2 + k_i^2) - (M^{-1}\sum_i \frac{1}{2}(j_i + k_i))^2}$$

où j_i, k_i sont les degrés des vertices à la fin de la ième arête, avec $i = 1...M$. Elle est étudiée sur les réseaux sociaux et sur les graphiques du World Wide Web. Nous allons donc essayer d'établir un lien via l'assortativité entre nos réseaux locaux et les réseaux sociaux ou les graphiques du World Wide Web. Les réseaux sociaux possèdent une assortativité positive, et donc ils ont tendance à avoir des noeuds ayant un haut degré qui se connectent entre eux. Tandis que les graphiques du World Wide Web ont tendance à avoir une assortativité assez mixte, et, donc, ils ont tendance à avoir des noeuds avec des hauts degrés qui se connectent à d'autres du même style, mais aussi à des noeuds ayant des degrés plus faibles.

Le tout est maintenant de savoir à quelle assortativité correspondent les réseaux locaux. Les auteurs de cet article [21] ont obtenu une valeur moyenne d'assortativité $\overline{r} = -0.055 \pm 0.04$. Il est important de remarquer que cette valeur est fortement constante alors qu'elle a été mesurée sur une centaine de villes et elle ressemble énormément à celle obtenue sur les graphiques

du World Wide Web qui est $\overline{r} = -0.065$ et aussi à celle des graphiques de l'interaction des protéines qui est de $\overline{r} = -0.156$. Cette assortativité peut s'expliquer par différents facteurs, notamment par le fait qu'elle est polycentrique, c'est-à-dire que les réseaux locaux ont plusieurs centres. Cela est dû au fait que chaque endroit dans les réseaux locaux a un certain but : il y a, par exemple, des quartiers d'affaire, de restauration, ou encore de transport. De plus, les réseaux locaux ont une structure hiérarchique, car ils possèdent un haut coefficient de clustering et la "heavy-tailed degree distribution".

On peut voir d'ailleurs apparaître deux modèles principaux dans les réseaux locaux : un modèle lié aux transports qui correspond à des noeuds de degré élevé et un modèle correspondant à des noeuds de degré faible liés entre eux représentant les utilisateurs allant dans des endroits où se nourrir, puis allant dans des endroits proches où s'amuser, par exemple.

1.3 Prédiction des arêtes

Maintenant que nous avons établi les propriétés des réseaux locaux et les liens avec d'autres réseaux, il semble intéressant de se servir de ces propriétés et de ces liens afin d'établir des méthodes pour prévoir la création de ses arêtes.

Dans ce chapitre, nous allons nous servir de cinq méthodes pour prédire les arêtes manquantes dans le graphe et ainsi établir le mouvement de population qui lui est associé. Le problème consiste donc à associer un certain score r_{ij} à une paire de noeuds (i, j).

Ce score représentera la possibilité qu'une arête se crée en partant du noeud i à destination du noeud j avec i et j appartenant à l'ensemble des noeuds (V). Il existe de nombreuses manières d'établir ce score et beaucoup de paramètres sur lesquels se focaliser. Nous allons principalement nous concentrer sur le nombre de voisins communs pour ce travail et faire une analyse complète en ne gardant que 85%,60% et 35% des données et en voyant comment notre algorithme complète les déplacements manquants afin d'évaluer s'il est capable de prévoir les déplacements les plus probables de la population.

Nous nous arrêterons à 5 méthodes : quatre basées sur les voisins et une sur le découpage de graphe.

1.3.1 Méthode basée sur la fermeture triadique

Dans cette partie, nous allons nous concentrer sur les méthodes basées sur les autres réseaux. Nous allons donc nous servir des propriétés précédemment développées afin d'établir de bons algorithmes pour la prédiction des arêtes. Nous allons nous servir principalement de deux propriétés : la fermeture triadique et la mesure de centralité des noeuds. Pour cela nous nous servirons de différents articles : [21],[4],[8],[26],[14]

Un moyen de déterminer si deux noeuds se connecteront est le nombre de liens qu'ils partagent. Nous allons donc définir le nombre de voisins qu'ils partagent par $|\Gamma_i \cap \Gamma_j|$ où i et j représentent des noeuds et Γ l'ensemble des voisins. Et donc plus ils ont de voisins en commun, plus ils ont de chances de se connecter comme nous le montre la figure 11.

Cependant, il existe un coefficient appelé le coefficient de Jaccard qui mesure la similarité et qui est meilleur que le nombre de voisins communs, car il tient compte de la taille du voisinage des deux noeuds. Cette indice a été proposé il y a déjà plus de 100 ans bien avant l'étude de prédiction des liens. Celui-ci se définit comme suit :

$$\frac{|\Gamma_i \cap \Gamma_j|}{|\Gamma_i \cup \Gamma_j|}$$

Ensuite, nous utiliserons l'index d'allocation de ressources. Cet index est motivé par les ressources d'allocation dynamique dans les réseaux complexes. Si l'on considère deux noeuds qui ne sont pas directement connectés. En considérant la popularité de certains endroits, l'on peut imaginer qu'on envoie des personnes vers un autre lieu qui lui-même envoie les gens vers tous ses voisins dont les noeuds qui n'étaient pas directement connectés au premier . Par exemple : si l'on prend des gens visitant Paris, certaines personnes vont aller de la Tour Eiffel aux Champs Elysées en vue d'aller à l'Arc de Triomphe. Dans ce cas la Tour Eiffel et l'Arc de Triomphe ne sont pas directement reliés mais les Champs Elysées jouent le rôle de transmetteurs. La particularité de cette métrique est qu'elle est symétrique. De plus, comme la méthode suivante, elle pénalise les voisins ayant un haut degré.

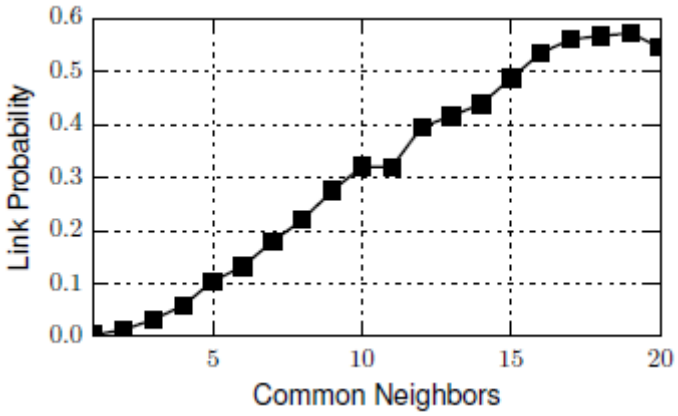

FIGURE 11 – Probabilité de la formation d'un lien dans la période de temps suivante en fonction du nombre de voisins partagés entre deux noeuds pour Chicago. Source : [21]

$$\sum_{z \in \Gamma_i \cap \Gamma_j} \frac{1}{|\Gamma_z|}$$

Enfin, nous utiliserons l'indicateur AdamicAdar qui dépend du nombre de voisins en commun, mais qui pénalise les voisins qui ont un haut degré. Cet indicateur est connu, car il est très efficace pour les réseaux sociaux[1].
Cet indicateur est défini comme suit :

$$\sum_{z \in \Gamma_i \cap \Gamma_j} \frac{1}{\log |\Gamma_z|}$$

1.3.2 Méthode basée sur la centralité des noeuds

En plus d'utiliser les différents indicateurs nécessaires pour l'utilisation de la propriété de la fermeture triadique, nous allons également essayer de nous servir de l'importance des différents noeuds. Pour cela, nous allons utiliser le degré de centralité comme mesure de centralité qui est incorporée dans le produit des degrés. Le score d'attachement préférentiel se définit donc comme suit :

$$|\Gamma_i|.|\Gamma_j|$$

.

1.3.3 Méthode de machine learning basée sur le calcul des vecteurs propres du Laplacien

Dans cette méthode, on va calculer le spectre du graphe Laplacien normalisé (en particulier les vecteurs propres associés aux plus petites valeurs propres).[13] Celui-ci est interprété en terme de nombre minimum de coupures nécessaires pour diviser le graphe en composantes de taille comparable. Cette méthode est donc très différente de la précédente car elle ne se base plus sur le nombre de voisins communs mais sur le graphe en lui-même. En effet il va subdiviser le graphe en plusieurs sous-graphiques afin de prédire les liens entre les noeuds.

Le calcul du graphe Laplacien est contruit comme $L = D - A$. Cependant, on calcule le Laplacien normalisé de cette façon $L = D^{-\frac{1}{2}}(D - A)D^{-\frac{1}{2}}$.[5]

1.4 Evaluation des méthodes de prédiction d'arêtes

Afin d'évaluer la précision des prédictions de nos algorithmes, nous allons devoir définir trois métriques la Receiver Operating Characteristic(ROC), Area under the receiver operating characteristic curve(AUC et la précision moyenne).[10][24]

1.4.1 Receiver Operating Characteristic and Area under the receiver operating characteristic curve

ROC établit une fonction de taux de vrais positifs en fonction du taux de faux positifs. ROC est donc le mieux sur le coin supérieur droit avec un taux de faux positifs de 0 et un taux de vrais positifs de 1. On notera que le maximum de vraisemblance est représenté comme la tangente en un point sur ce graphique.

AUC donne donc un rang pour toutes les arêtes qui n'ont pas été observées. Il peut être interprété comme la probabilité de choisir une arête manquante aléatoire est plus grande que de choisir une arête aléatoire inexistante. Dans les algorithmes implentés qui est notre cas, on calcule en général le score de chaque arête non observée au lieu de donner une liste ordonnée qui est plus complexe. Ensuite à chaque fois que l'algorithme choisit une arête manquante ou inexistante , on compare son score et si après n comparaisons indépendantes il y a n' fois un lien manquant ayant un plus grand score et n" fois qu'ils ont le même score, la valeur AUC vaut :

$$\frac{n' + 0.5n''}{n}$$

Si tous les scores sont générés selon une distribution identique et indépendante, la valeur de AUC devrait être aux alentours de 0.5. De ce fait plus la valeur excède 0.5 meilleur est l'algorithme. Si l'algorithme est inférieur à 0.5, cela signifie qu'il est moins bon qu'une distribution aléatoire des arêtes.

Si l'on souhaite exprimer AUC en fonction du ROC, AUC représente l'aire sous la courbe ROC ; elle correspond donc à l'intégrale de celle-ci.

1.4.2 Précision moyenne

Afin de comprendre cette mesure, nous devons dans un premier temps définir la précision et Recall.[9]

La précision(P) est définie comme le nombre de vrai positifs sur le nombre de vrais positifs et de faux positifs

$$P = \frac{T_p}{T_p + F_p}$$

Le recall(R) est défini comme le nombre de vrais positifs sur le nombre de vrais positifs et de faux négatifs

$$P = \frac{T_p}{T_p + F_n}$$

Ces quantités sont aussi en relation avec le score F1 qui est défini comme la moyenne harmonique de la précision et du recall

$$F1 = \frac{2PxR}{P + R}$$

La précision et le recall sont une mesure très efficace de prédiction quand les classes sont vraiment déséquilibrées. La précision est une mesure de la pertinence des résultats, et recall mesure combien de résultats vrais sont retenus.

Une AUC élevée représente une grande précision et un grand recall, car une haute précision est associée à un faible taux de faux positifs et un haut recall est associé à un faible taux de faux négatifs. Un grand score pour les deux montre que l'algorithme donne des résultats précis et une majorité de résultats positifs.

La précision moyenne résume donc un graphique à un moyenne pondérée de la précision à chaque étape avec l'augmentation du recall comme précédente étape utilisée comme poids.

$$AP = \sum_n (R_n - R_{n-1}) P_n$$

où P_n et R_n correspondent à la précision et au recall à la n^ime étape. La paire (R_k, P_k) se réfère à ce que l'on appelle un "operating point".

1.5 Conclusions

Ce chapitre nous a permis de retenir comment se comporte la répartion des arêtes et des noeuds et à quoi devraient correspondre nos données.

Il nous a également montré l'importance que possèdent certaines propriétés dans les algorithmes de prédiction d'arêtes.

Enfin il nous a permis de définir les métriques permettant d'évaluer ces derniers.

Avec tous ces résultats en tête, nous pouvons analyser nos données.

2 Analyse des données Foursquare

Afin de prédire les déplacements de population, nous nous sommes basé sur les données Fousquare utilisé dans cette article [27]. Avant de prédire les arêtes du graphe et donc les déplacements de population, il est important d'analyser les données afin de comprendre ce que décrivent les données.

Pour nous aider à construire ces graphique, nous nous sommes principalement servis du language python. Nous avons utiliser les librairies suivantes : Networkx[2],panda, collection, matplotlib, gephi.

Afin que notre analyse soit la plus complète possible nous nous sommes basés sur [20],[27],[21],[19] en vue de vérifier nos analyse et leur correspondance par rapport à d'autres résultats.

2.1 Analyse globale des données

Les données utilisées concernent les check-ins via l'application Foursquare collectées sur plus de 10 mois dans les villes de New York(du 10 Avril 2012 au 16 février 2013) et dans la ville de Tokyo (du 3 avril 2012 au 16 février 2013). Elle contient 227.428 check-ins dans la ville de New York et 573.703 check-ins dans la ville Tokyo. Chaque Check-in est associé à une date, à des coordonnées GPS(latitude et longitude), à au type d'endroit auquel il est associé. Ces données ont été utilisés à la base pour étudier les régularités spatio-temporelles des utilisateurs.
Ces données concernent 1083 utilisateurs étant allés dans 38333 endroits différents au sein de la ville de New York et 2293 utilisateurs étant allés dans 61858 endroits différents au sein de la ville de Tokyo.
Nous allons définir les endroits comme les noeuds de notre graphe et les arêtes comme les déplacements d'un endroit vers un autre.

Dans un premier temps, nous allons étudier les types de lieu auxquels Foursquare est associé afin d'étudier les types de lieu vers lesquels vont le plus les habitants New-Yorkais.

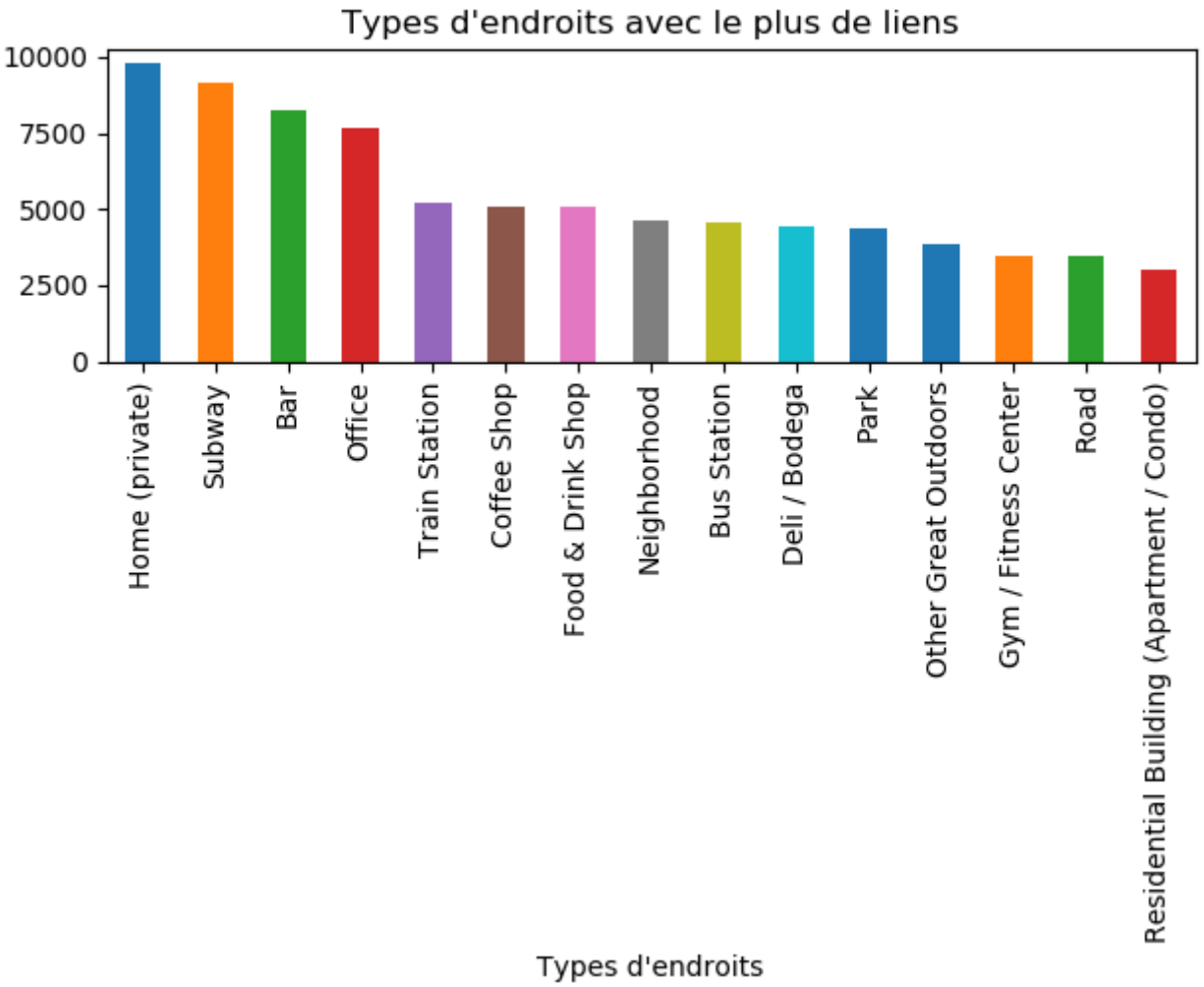

FIGURE 12 – Nombres de liens en fonction du type d'endroits à New York

2.1.1 Type d'endroits ayant le plus de liens

1. New York

Dans ce graphique (figure 12), on peut voir que la plupart des endroits enregistrés sont en premier lieu les maisons. En effet,en général l'on part de chez nous pour aller quelque part ou l'on revient chez nous pour passer la nuit. Chaque endroit visité depuis ou vers chez nous produit un lien si cela n'a pas encore été fait.

Ensuite l'on peut voir que les métros sont le plus utilisés. Dans les grandes villes, la plupart des gens n'ont pas de voiture ou se déplacent en métro, car c'est le moyen de transport le plus rapide.

En troisième lieu, ce sont les bars qui sont les lieux de rencontre de nombreuses personnes et en quatrième les bureaux qui sont aussi l'endroit où de nombreuses personne

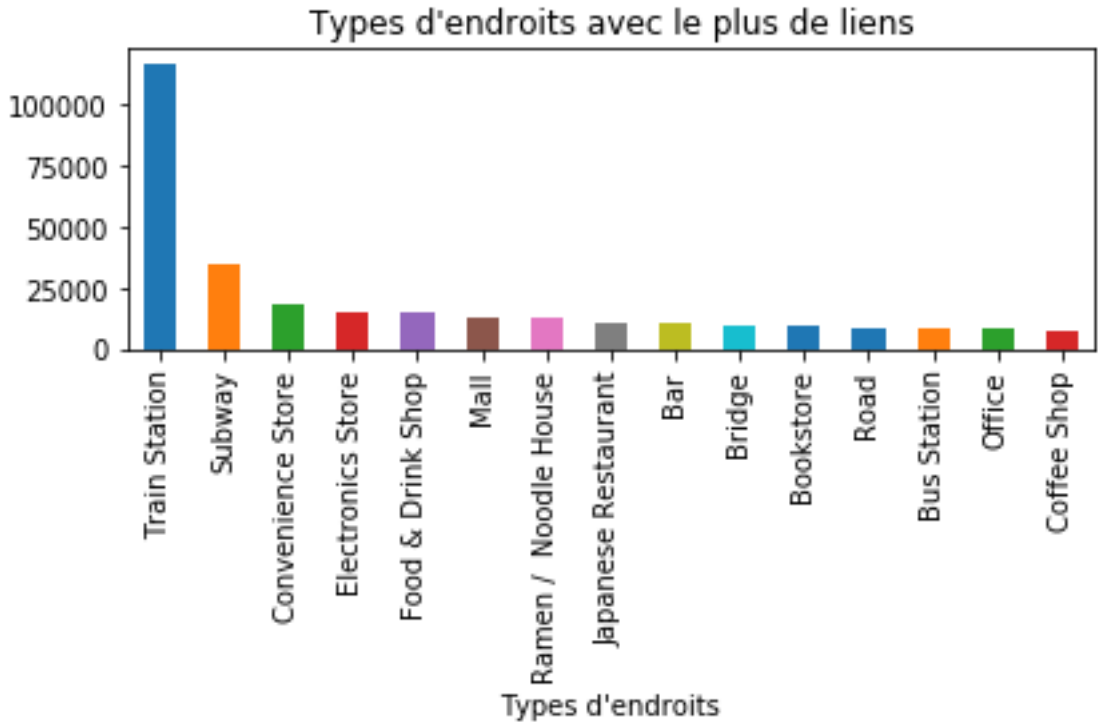

FIGURE 13 – Nombres de liens en fonction du type d'endroits à Tokyo

vont tous les jours de la semaine. La suite correspond à des raisons similaires, mais avec des endroits ayant moins de succès comme les stations de trains pour les déplacements plus longs, les coffee shops comme lieux de détente. On peut donc classifier les endroits restants en plusieurs catégories.

(a) Les déplacements correspondant au métros, stations de train, de bus ou les routes.

(b) Les lieux de vie comme les maisons et les appartements

(c) Les lieux de travail comme les bureaux

(d) Les lieux de détentes comme les bars, les coffee shop, les bodega, les parcs, les centres de fitness,...

(e) Les lieux où acheter de la nourriture et des boissons.

On notera qu'il n'y a ni la présence de restaurants ni celle de magasins qui sont pourtant des lieux souvent visités. Cela s'explique par le fait que Foursquare catégorise de nombreux types de restaurants et de magasins différents. Il sera donc intéressant de regrouper certains types de lieux pour la suite.

2. Tokyo

On peut voir que l'analyse n'est pas la même pour Tokyo(figure 13). En effet, on peut voir que la plupart des lieux correspondent à des stations de trains, puis à des métros. On peut donc voir l'importance des transports en commun dans cette ville. En effet, cela s'explique par le fait que le système de transport des principales agglomérations japonaises paraît répondre aux exigences de la mobilité urbaine contemporaine [17].

On peut aussi ajouter qu'il est étonnant de voir que les maisons ne sont pas incluses dans les 15 endroits ayant le plus d'entrées dans cette ville. Une des explications est sans doute la mentalité très différente entre la ville de New York et Tokyo.

Cela se montre clairement dans la suite de l'analyse. En effet, on peut voir que les magasins d'électronique sont en quatrième position car c'est un secteur important dans les pays asiatiques tels que le Japon, l'Inde ou la Chine. Le graphique nous montre l'importance de ceux-ci au Japon.

Vient ensuite les épiceries et en troisième position, les magasins vendant de la nourriture et des boissons et les centres commerciaux. On notera que les centres commerciaux et les épiceries n'étaient pas présents dans les quinze types d'endroits les plus visités à New York. Cela s'explique par la présence de ce que l'on appelle des Kombinis aux Japon qui correspondent à des "Convenience Store" américains. Cela correspond à des petits supermarchés. Il y a à peu près l'équivalent d'un kombini pour 3300 habitants au Japon. Ils sont de plus ouvert 24h/24 et 7j/7 favorisant ainsi l'apparition de ces noeuds dans notre réseau.[15]

On peut voir aussi que les habitudes alimentaires des Japonais sont très différentes de celles des New Yorkais. En effet, les restaurants n'étaient pas présents à New York car ils étaient d'une grande diversité. On peut voir ici que les Japonais possèdent des habitudes alimentaires plus précises en mangeant davantage dans des restaurants à nouilles ou ramen ou dans des restaurants japonais. Cela peut s'expliquer notamment par le fait que la nourriture européenne et américaine représentent une nourriture de luxe dans ce pays.

Enfin on peut voir que les lieux de détente se situent très loin dans les habitudes des Tokyotes. Encore une fois, cela s'explique en partie comme précédemment par leur mentalité que l'on associe en général à des travailleurs acharnés.

Cependant, les bureaux sont classés très loin dans les habitudes des Tokyotes. Peut-être qu'une des explications peut être le fait qu'il existe une moins grande quantité de bureaux à Tokyo qu'à New York ou l'importance accordée au travail est si grande qu'ils ne prennent pas la peine de l'enregistrer dans l'application. Effectivement, en 2005, une loi sur la santé et la sécurité au travail impose aux employeurs de limiter les heures supplémentaires au travail à 100h/mois.[11]

2.1.2 Types d'endroits avec le plus de liens sortants

1. New York

Afin de définir les liens sortants et les liens entrants de façon plus précise, nous imposons une contraite de temps de 3h sur les déplacements. Nous pouvons ainsi les données parasites. Ainsi les déplacements de plus de 3h ne correspondent pas à des liens entre deux endroits.

Dans ce graphique(figure 14), l'on peut voir que les lieux d'où l'on provient viennent en général du métro qui est un lieu de passage de nombreuses personnes. En général, les gens viennent du métro en vue d'aller quelque part. Il est donc normal que cela corresponde à à un lieu sortant.
Vient ensuite les maisons puis les bars.

On peut remarquer que les lieux correspondants aux transports correspondent davantage à des lieux sortants. En effet étant des lieux de passage,il est normal que cela corresponde davantage à des lieux sortants étant donné que l'on y va à en vue d'aller ailleurs.

2. Tokyo

On peut établir pour ce graphique (figure 15) le même raisonnement que sur le graphique de New York même si cela est accentué vu le nombre d'entrées pour les stations de train et les métros.
On peut en effet voir que les 2/3 des entrées pour les stations de trains et les métros sont des liens sortants.

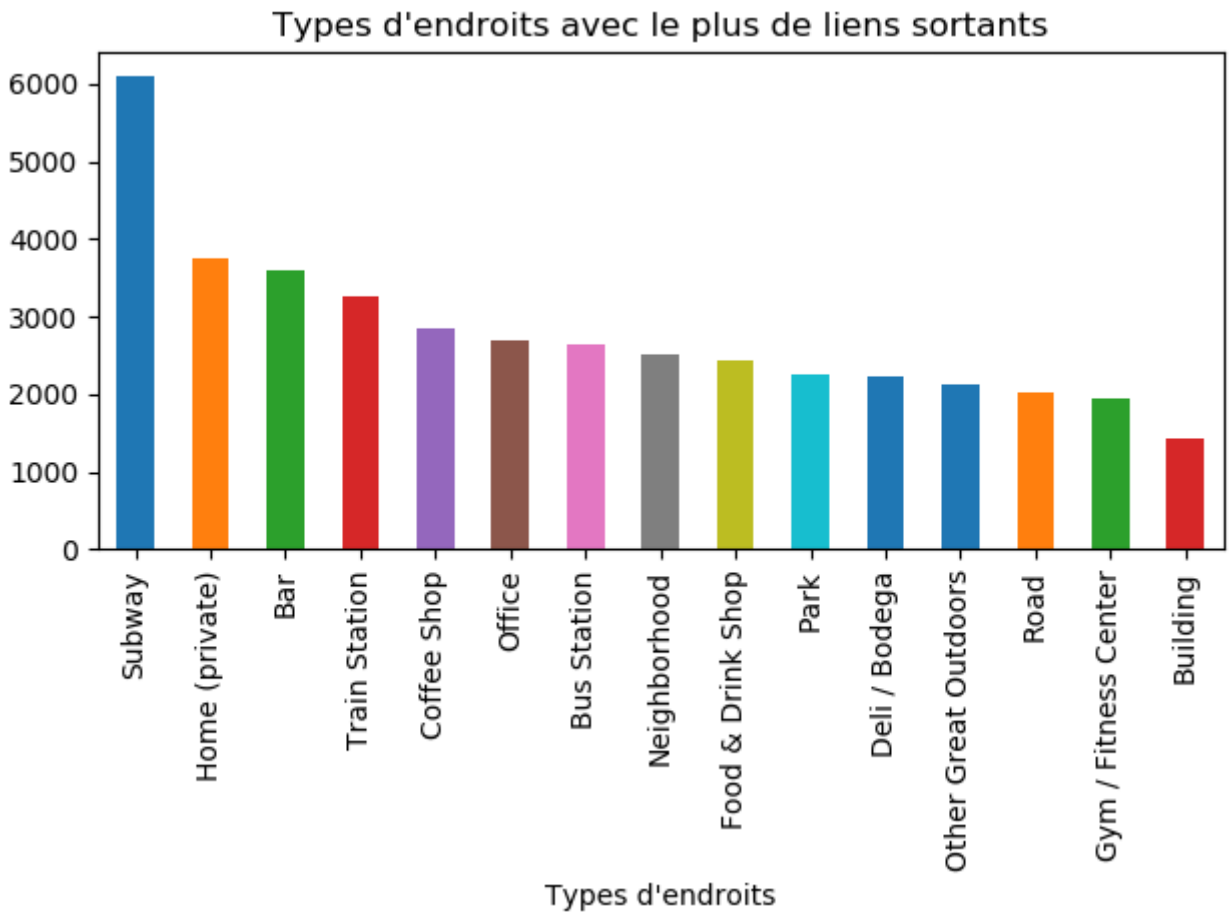

FIGURE 14 – Nombres de liens sortants en fonction du type d'endroits à New York

On peut voir que les bureaux ne sont plus présents étant donné que cela correspond davantage à un lieu où on reste longtemps.

2.1.3 Types d'endroits avec le plus de liens entrants

1. New York

Dans ce graphique(figure 16), on peut voir que les maisons et les bureaux sont davantage des lieux entrants, ce qui est logique, car cela correspond davantage à des lieux où l'on a prévu de rester. A contrario, les lieux de transport ont nettement chuté, ce qui confirme l'analyse précédente.

2. Tokyo

On peut établir le même raisonnement que sur le graphique de New York.(figure 17)

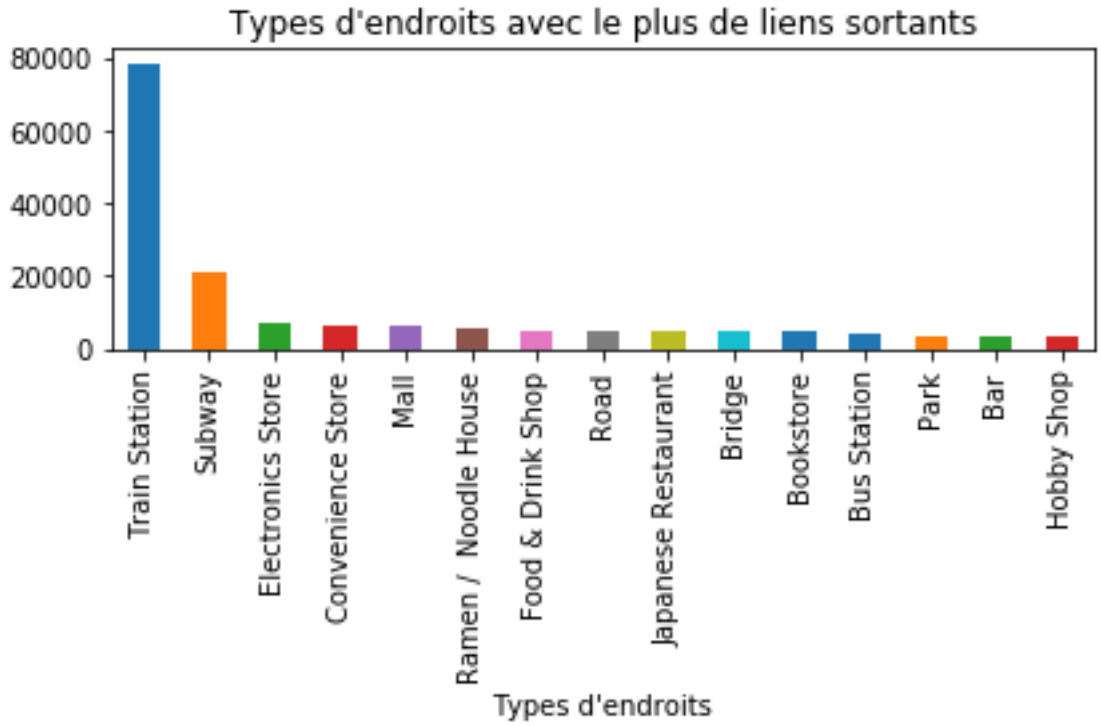

FIGURE 15 – Nombres de liens sortants en fonction du type d'endroits à Tokyo

Les liens sortants pour le métro et les stations de trains sont fortement réduits même s'ils restent prédominants à cause du nombre d'entrées dans ceux-ci.
On peut noter que les bureaux correspondent énormément à des liens entrants pour la raison citée précédemment.
On peut cependant noter une différence en ce qui concerne les bars qui semblent davantage correspondre à des liens entrants à Tokyo.

2.1.4 Distribution du type de noeuds en fonction de leur degré

Nous allons maintenant analyser la distribution des types de noeuds. Pour cela nous allons regrouper les données en six catégories :

1. Les lieux qui concernent exclusivement la nourriture et les boissons
2. Les lieux qui concernent l'enseignement
3. Les lieux qui concernent l'art et l'amusement(bar,café inclus)
4. Les lieux professionnels et les maisons
5. Les magasins

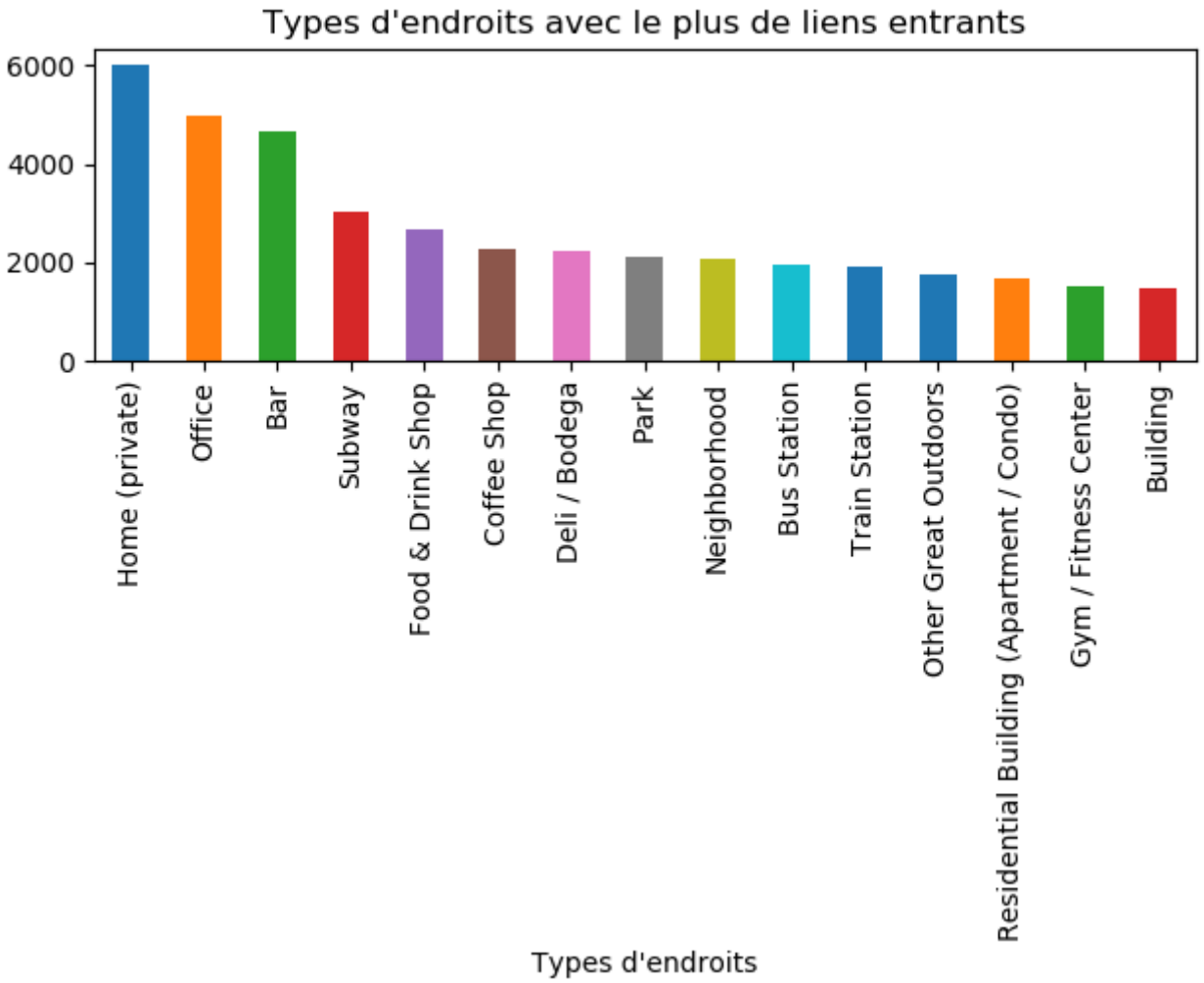

FIGURE 16 – Nombres de liens entrants en fonction du type d'endroits à New York

6. Les lieux associés aux transports

1. New York

Le choix d'associer le travail et les lieux d'habitations réside dans l'imprécision de l'enregistrement des données de Foursquare. Par exemple, l'entrée Building dans Foursquare est parfois associée à des grands immeubles d'appartements ou d'hôtels et parfois à de grands bureaux d'entreprises.

Dans ce graphique(figure 18), on peut voir qu'il y a énormément de lieux ayant un degré d'arêtes peu élevé qui concerne la nourriture. Cela s'explique par le fait qu'il existe de nombreux restaurants comme on l'a vu dans la partie théorique sur les réseaux locaux. Il y a ensuite une nette décroissance en fonction du poids.

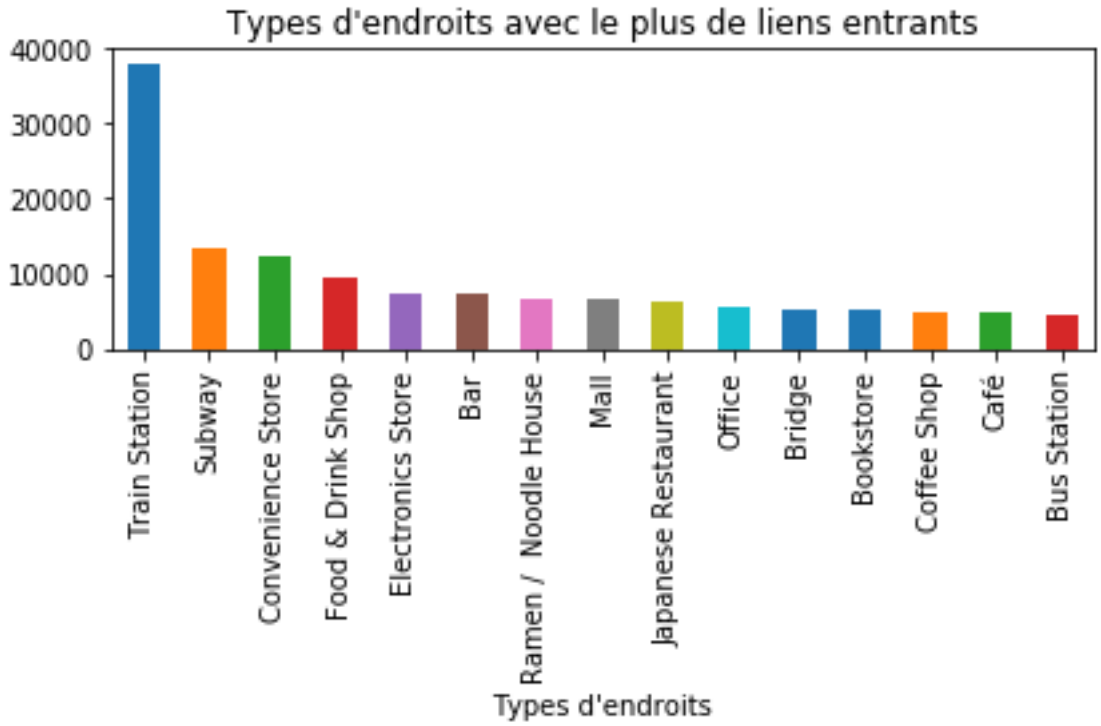

FIGURE 17 – Nombres de liens sortants en fonction du type d'endroits à Tokyo

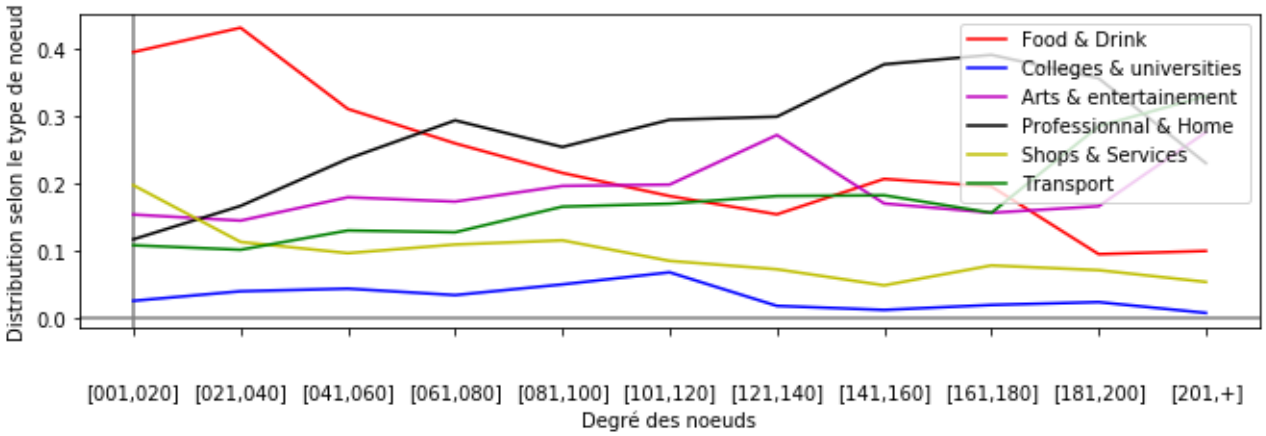

FIGURE 18 – Distribution du type de noeuds en fonction de leur degré au sein de la ville de New York

En ce qui concerne les lieux d'habitation et les lieux où l'on travaille, on peut expliquer la légère densification pour les noeuds de faible degré par la quantité de maisons. Cependant, comme les données ne concernent que 1083 personnes, il est normal que la quantité n'est pas énorme comparée à d'autres endroits. Les quantités plus grandes ayant un degré élevé s'expliquent par les quantités données par Office et Building qui correspondent parfois à des bureaux de grandes entreprises.

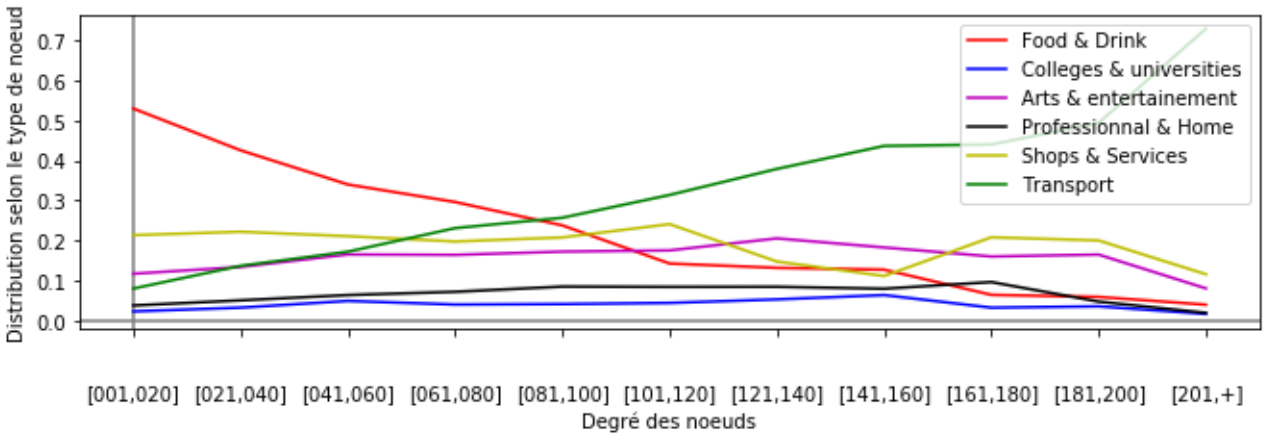

FIGURE 19 – Distribution du type de noeud en fonction de leur degré au sein de la ville de Tokyo

Les magasins diminuent avec le nombres de degrés ; cela s'explique par le fait qu'il existe de nombreux petits magasins de vêtements. Il y a cependant un léger pourcentage, car il existe de grandes galeries(mall) et de grands magasins.

On peut remarquer aussi que les transports augmentent avec le degré des noeuds cela s'explique par le fait que les métros, gares et arrêt de bus sont sur le trajet de nombreuses personnes en ville ; et donc le poids de ces noeuds devient importants.

2. Tokyo

Dans cette exemple (figure 19), la distribution pour le nombre de petit endroits où l'on peut manger, tels que les restaurants est énorme pour les lieux ayant un poids faible : on peut voir que plus de cinquante pour cent des données pour des noeudx de poids inférieur à 20 sont des restaurants.

De même, on peut noter pour notre analyse que le nombre de noeuds de degré plus élevé que 200 correspond à septante-cinq pour cent de nos données. Notre analyse pour la ville de New york est d'autant plus pertinente pour la ville de Tokyo.
On peut noter que les lieux pour s'amuser, concernant l'art ou de culte sont nettement moins présents dans la vie des Tokyotes que pour les Américains. Cependant, les magasins le sont davantage.

On notera aussi la faible présence des lieux d'habitations et des lieux de travail dans les

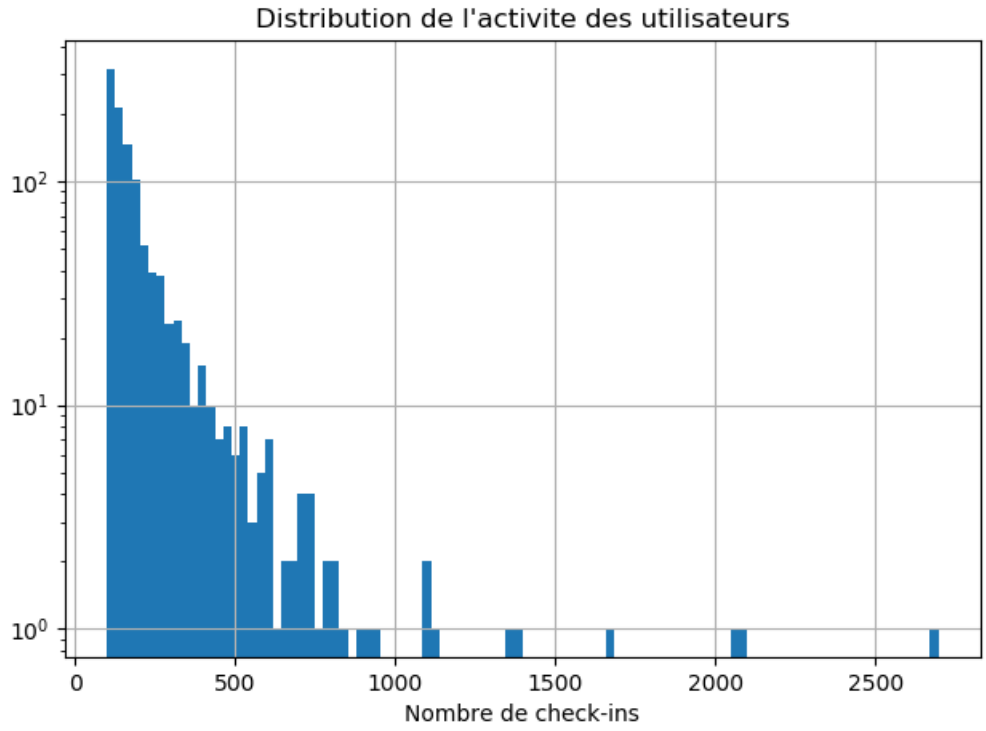

FIGURE 20 – Densité de répartitions du nombre de check-ins par utilisateur

données correspondant à la ville de Tokyo comparée à celle de New-York.

2.1.5 Activités des utilisateurs

1. New York

   Dans ce graphique(figure 20), l'on peut voir qu'il y a de nombreux utilisateurs n'ayant fait que quelques entrées dans l'application et que très peu ont contribué énormément aux données.

   On notera que la distribution de l'activité des utilisateurs suit une loi exponentielle décroissante et qu'un utilisateur a contribué à 3000 entrées dans la base de données.

2. Tokyo

   On peut voir que ces données sont du même acabit que celles de New-York et que le nombre de check-ins par utilisateur suit également une loi exponentielle décroissante comme pour les données de New-York.(figure 21)

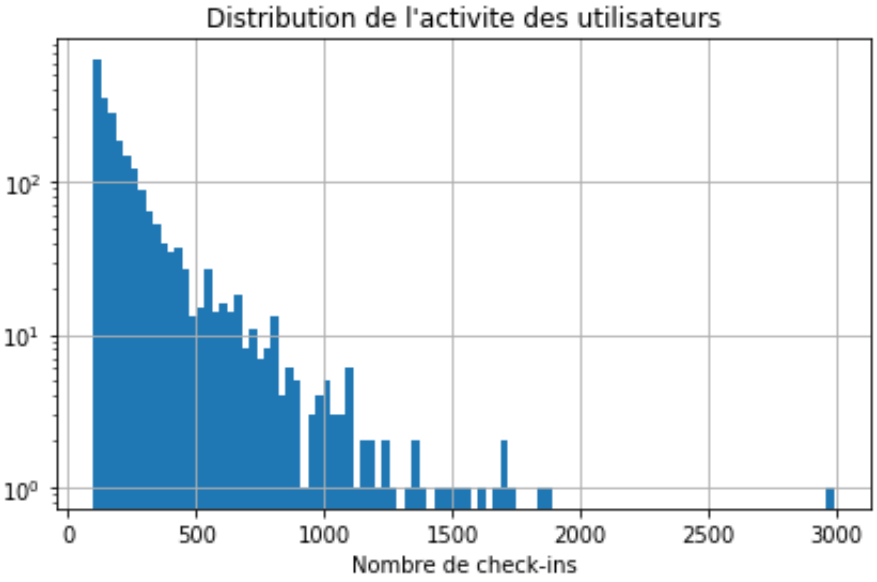

FIGURE 21 – Densité de répartition du nombre de check-ins par utilisateur

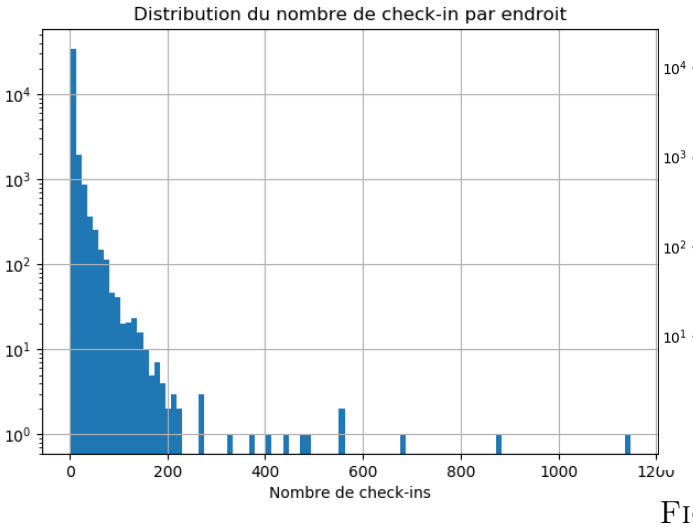

FIGURE 22 – Distribution du nombre d'entrées par endroit à New-York

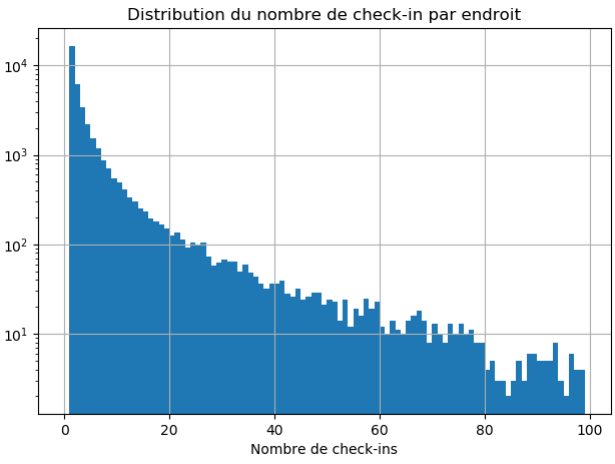

FIGURE 23 – Distribution du nombre d'entrées par endroit pour les 100 premières valeurs à New-York

2.1.6 Distribution du nombre d'entrées par endroits

1. New York

Dans le premier graphique(figure 22), l'on peut voir qu'il y a de nombreux endroits qui n'ont été visités qu'une seule fois et que quelques-uns ont été visités énormément de fois. On peut noter qu'un lieu a été visité 1140 fois.

Dans le second graphique(figure 23), on peut noter une répartition exponentielle dé-

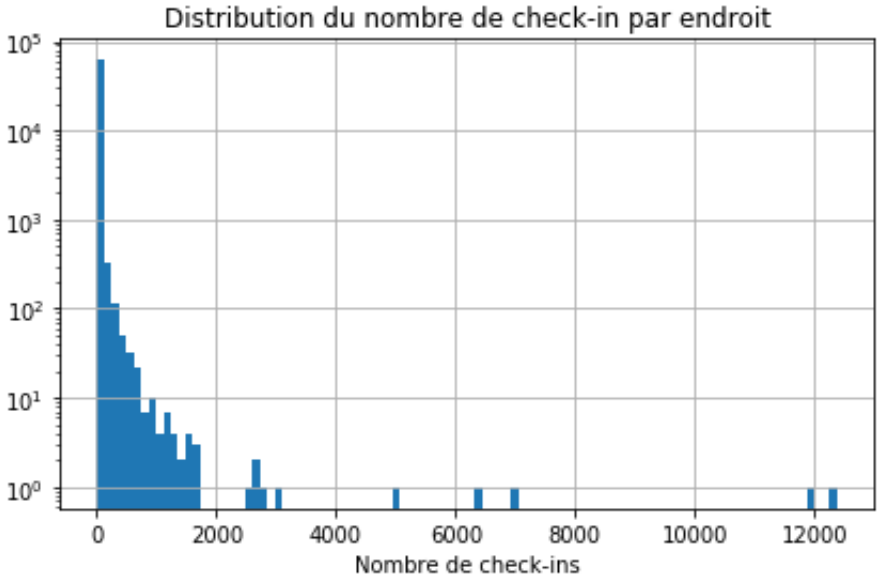

FIGURE 24 – Distribution du nombre d'entrées par endroit à Tokyo

croissante assez nette pour les 100 premières valeurs et le fait que plus de 10 000 lieux n'apparaissent qu'une fois dans nos données, ce qui corresponds à 1/4 de nos données.

2. Tokyo

   On peut voir que le graphique(figure 24) est du même acabi que le précédent est suit une exponentielle décroissante. Le fait qu'elle soit plus étendue est du notamment à la quantité de données supérieure à celle de New York.

2.2 Analyse de la correspondance aux propriétés théoriques des réseaux locaux

La quantité de données étant nettement inférieure au cas présenté dans le cas théorique comprenant plus de 10^8 données, il devient important aussi d'analyser si elles respectent les propriétés des réseaux locaux évoqués plus haut afin que les algorithmes de prédictions fonctionnent au mieux.

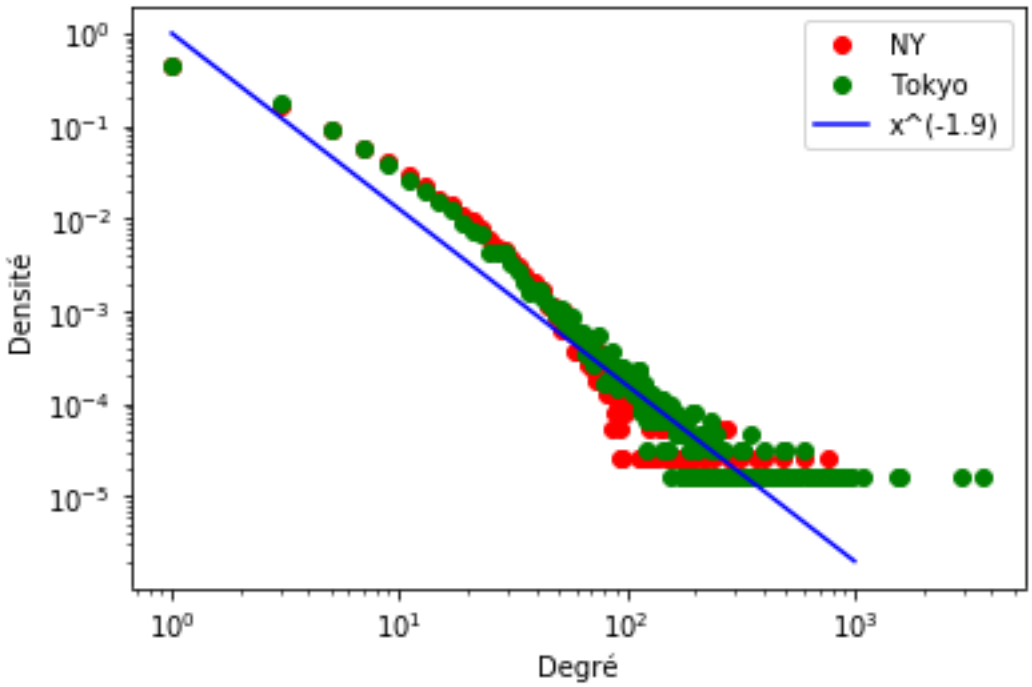

FIGURE 25 – Densité des noeuds en fonction de leur degré

2.2.1 Distribution des noeuds en fonction de leur degré

Dans cette figure(figure 25), l'on peut voir ce que l'on a vu précédemment, comme la heavy-tailed répartition propre aux réseaux locaux. On peut en effet voir que l'on a de nombreux noeuds ayant peu de degré et peu de noeuds ayant un degré élevé. On peut voir également que vu le manque de données, nous avons une limite 2.10^{-5}. Ce qui explique les lignes droites obtenues à la fin du graphique. On peut également remarquer que la fonction se rapprochant le plus du graphique est elle-même très proche des cas présentés en théorie, laquelle était de $k^{-1.84}$.

2.2.2 Distribution des arêtes en fonction de leur poids

1. New York

   On peut voir que la répartition des arêtes en fonction de leur poids suit une loi exponentielle décroissante.(figure 26)
   Elle suit une exponentielle décroissante $x^{-2.75}$, ce qui est inférieur à la déviation analysée dans le cas théorique. Cependant, si l'on regarde le graphique dans le cas théorique, New-York semble plus bas que cette exponentielle et comme nous possédons moins de

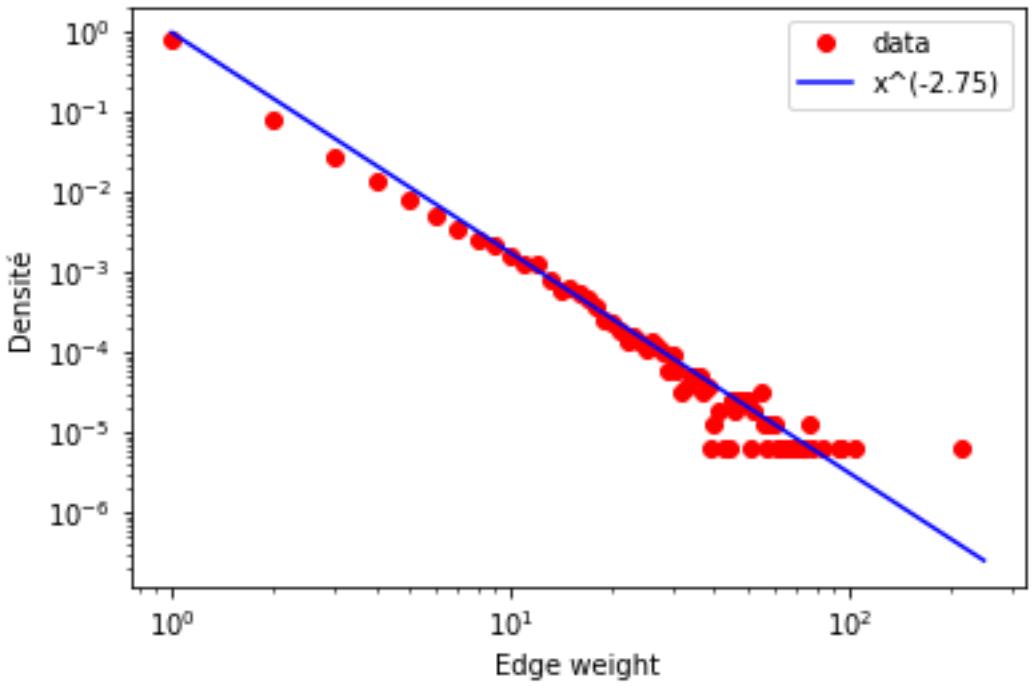

FIGURE 26 – Répartition des arêtes en fonction de leur poids pour la ville de New York.

données, la déviation est légèrement supérieure.
De ce fait on peut considérer que nos données suivent bien la théorie évaluée plus haut. Comme pour les arêtes on notera que nous avons une limite en 8.10^{-6} à cause de la quantité de nos données.

2. Tokyo

Pour la ville de Tokyo, les données se rapprochent davantage du cas théorique avec une exponentielle décroissante de type $x^{-2.6}$.(figure 27)

2.2.3 Nombre de noeuds en fonction du nombre d'arêtes

Dans ce graphique(figure 28), l'on peut voir que les données de New-York et de Tokyo suivent la densification du réseau vue dans la partie théorie. En effet, le nombre d'arêtes augmente superlinéairement avec le nombre de noeuds. Donc, si on prend n(t) comme le nombre de noeuds observés au temps t et e(t) comme le nombre d'arêtes à ce même temps t, on a

$$e(t) \propto 0.02n(t)^{1.49}$$

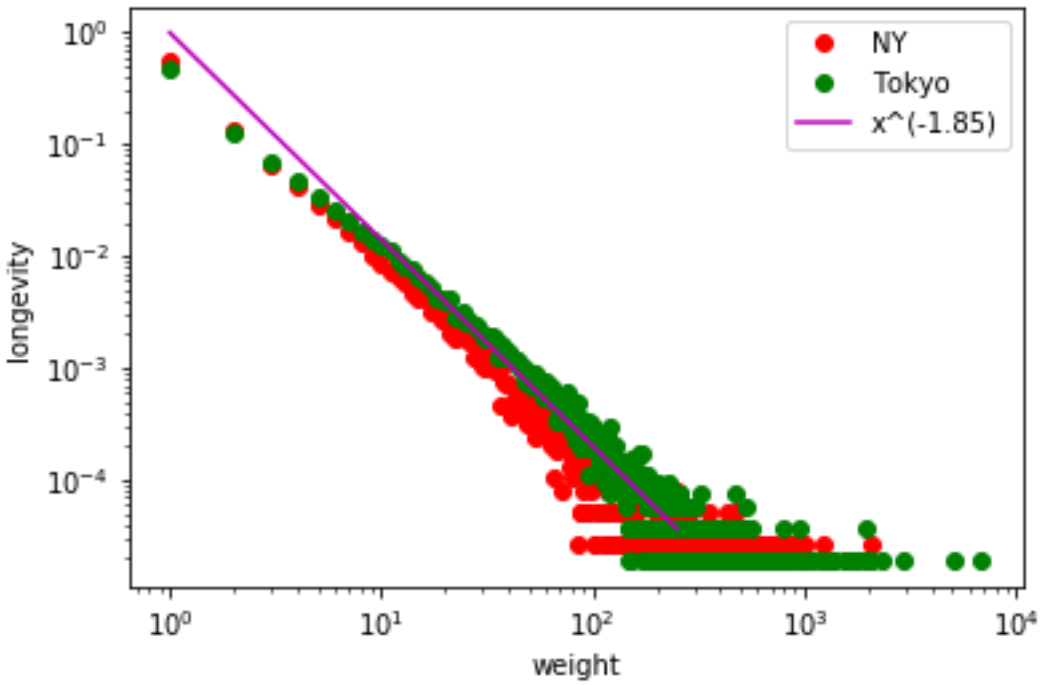

FIGURE 27 – Répartition des arêtes en fonction de leur poids pour la ville de New York.

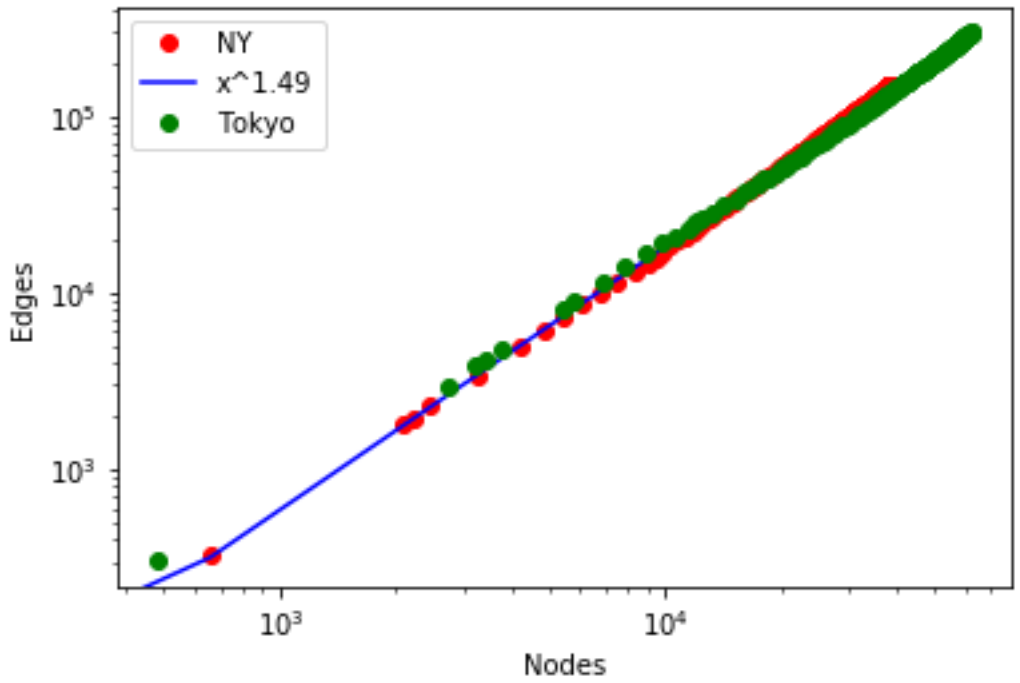

FIGURE 28 – Nombres de noeuds en fonction du nombre d'arêtes

Cela montre donc que le nombre de degrés va augmenter au fur et à mesure du temps, car $\alpha > 1$. Cependant, cette constation est contraire au cas théorique présenté où l'exponentielle

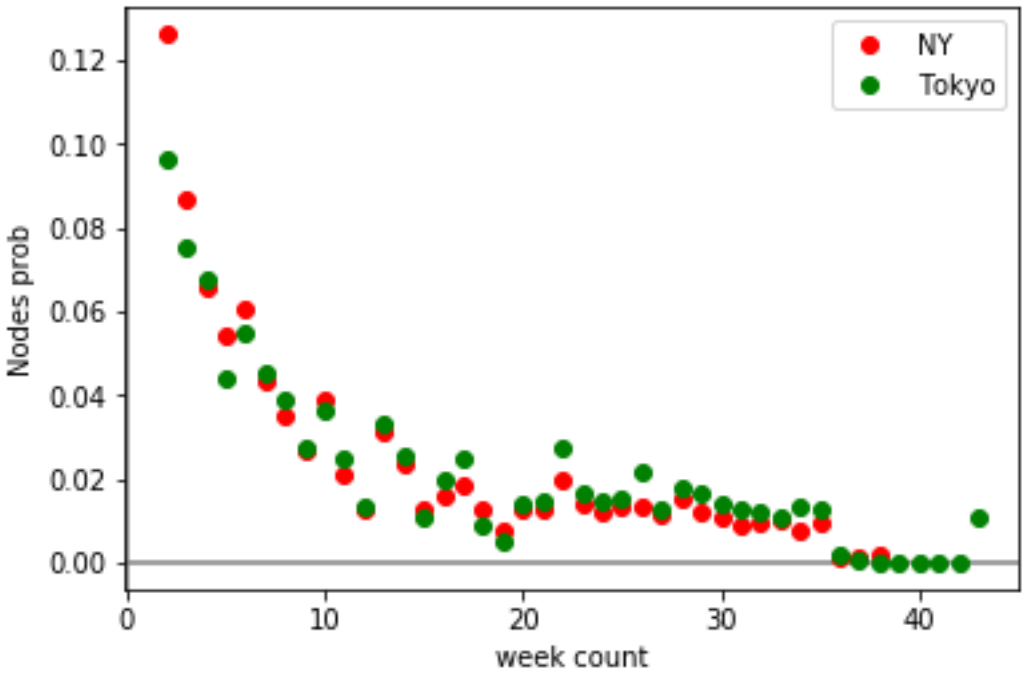

FIGURE 29 – Probabilité de découvrir un nouvel endroit sur le total d'endroits trouvés en fonction du nombre de semaines pour New York et Tokyo

est égale à 1.49, ce qui est bien supérieur au cas théorique. Cependant afin de le faire correspondre aux données j'ai dû multiplier la fonction par 0.02. Cela s'explique par le fait que les données correspondent à peu de personnes et donc à la faible quantité de données par rapport à la théorie.

2.2.4 Probabilité de découvrir un nouvel endroit

Bien que la décroissance est moins nette, cela correspond aux cas théoriques(figure 29). Cette décroissance s'explique encore une fois par le manque de données. Avec le temps, les utilisateurs découvrent de moins en moins d'endroits ; cela n'est pas dû au fait que le nombre de nouveaux endroits diminue dans notre cas, mais plutôt au fait que le nombre d'utilisateurs étant réduit, les habitudes des utilisateurs doivent être prises en compte. On peut voir que le nombre de nouveaux endroits continue d'apparaître de façon superlinéaire.

2.2.5 Probabilité d'une nouvelle arête entre deux périodes de temps

On peut voir dans ce graphique(figure 30) que le nombre de nouvelles arêtes est beaucoup plus élevé que ceux présentés dans le cas théorique. Dans le cas de New-York il se situe entre

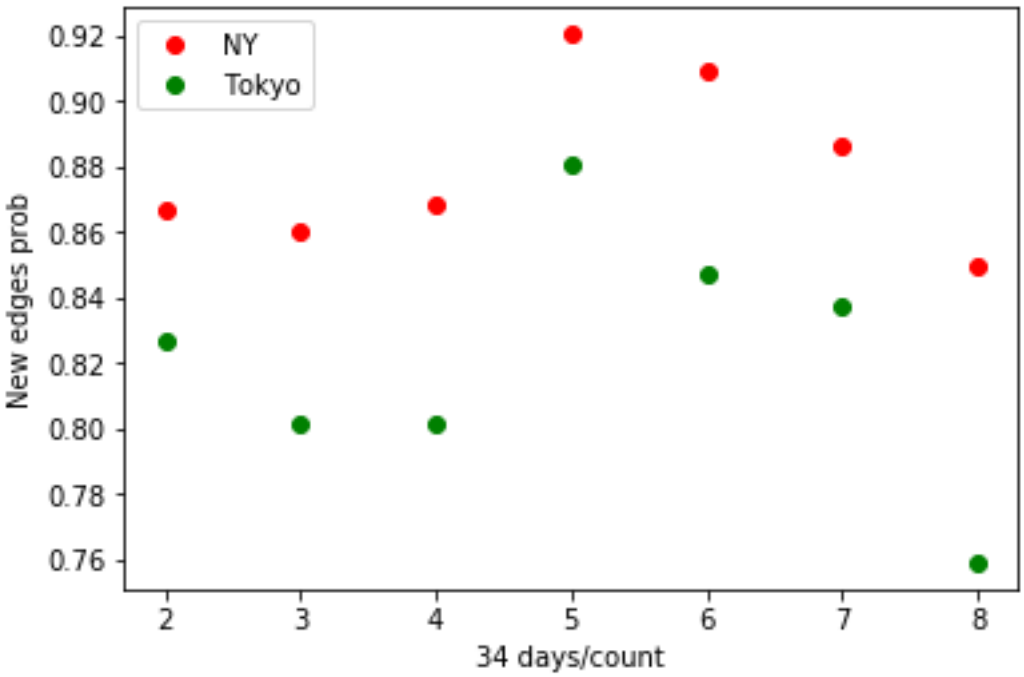

FIGURE 30 – Probabilité de découvrir une nouvelle arête entre deux endroits entre deux périodes de temps pour New York et Tokyo

0.85 et 0.92, alors qu'il devrait se situer entre 0.7 et 0.8. On peut observer que les données de Tokyo se rapprochent plus de la théorie avec des valeurs entre 0.755 et 0.88. Cela s'explique encore une fois par le manque de données présents dans notre base de données. La propriété small-world s'applique de façon moindre, étant donné que le nombre de données par rapport à la taille de l'endroit est faible. La probabilité d'une nouvelle arête est donc augmentée.

2.2.6 Probabilité d'un nouveau noeud entre deux périodes de temps

Dans ce graphique(figure 31), on peut observer que nous sommes loin des données théoriques. En effet, celles-ci estime la probabilité d'un nouveau noeud autour de 5 pour cent. Or dans mon cas cette probabilité est en moyenne de 50 pour cent. Cela est dû à la quantité de données qui est nettement insuffisante par rapport au cas théorique. La propriété small-world n'a que très peu d'impact dans notre cas, vu la zone évaluée et la quantité de données à notre disposition.

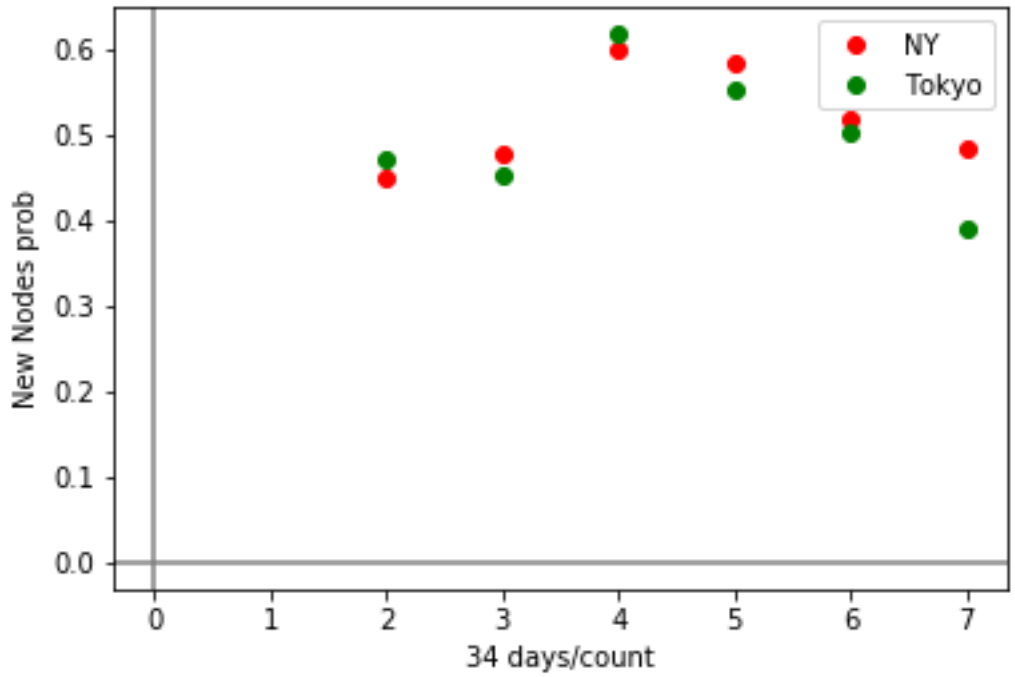

FIGURE 31 – Probabilité de decouvrir un nouveau lieu entre deux périodes de temps pour New York et Tokyo

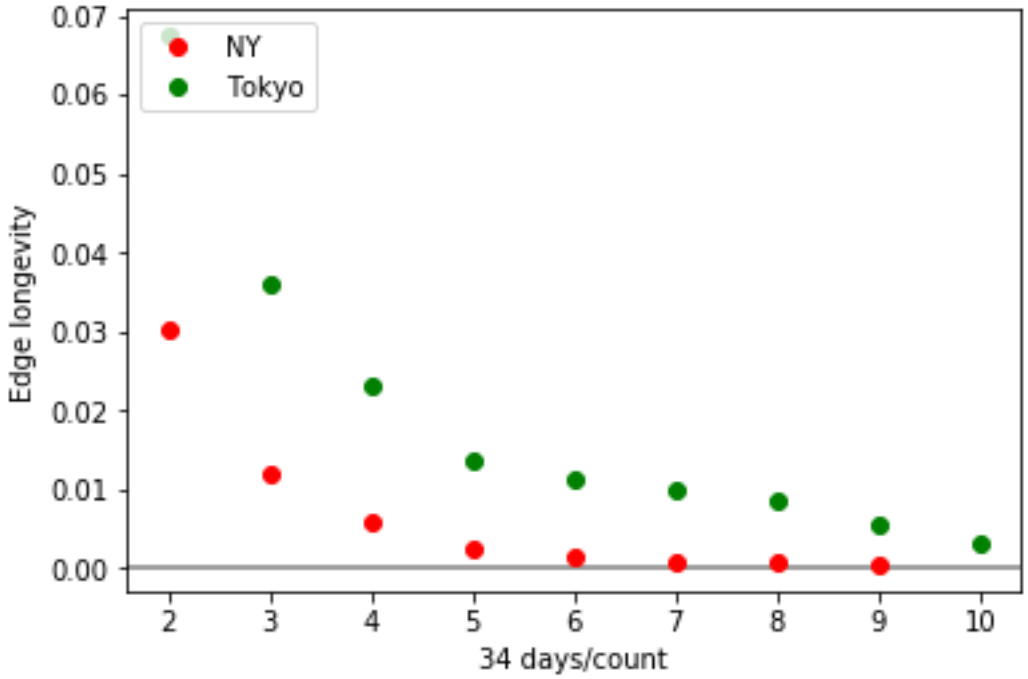

FIGURE 32 – Persistance des arêtes au fur et à mesure du temps

2.2.7 Persistance des arêtes au fur et à mesure du temps

On peut observer que la persistance des arêtes est loin du cas théorique(figure 32). En effet elle se situe entre 7 % et 0.0005 %.Or elle devrait se situer aux alentours de 30% au début

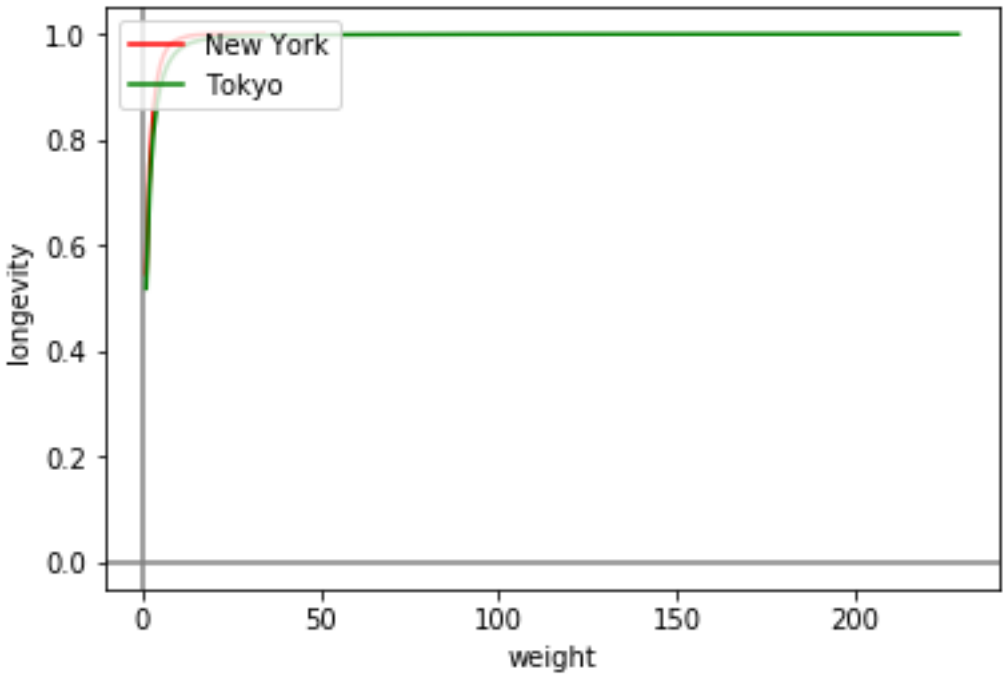

FIGURE 33 – Persistance des arêtes en fonction du poids de celle-ci sur deux périodes de temps successives

et 23% à la fin.
Cela s'explique par la quantité insuffisante de données. Cela va énormément influencer nos algorithmes de prédiction des arêtes.

2.2.8 Persistance des arêtes en fonction du poids de celles-ci sur deux périodes de temps successives

On peut voir que 54 % des arêtes présentes sur deux périodes de temps successives possèdent un poids de deux(figure 33). Cela s'explique par le grand nombre d'arêtes de faible poids. 90 % des arêtes présentes sur deux périodes de temps succéssives possèdent un poids inférieur ou égal à 5.

2.2.9 Persistance des arêtes en fonction du poids de celles-ci sur la totalité des périodes de temps

On peut voir dans ce graphique(figure 34) que la persistance des noeuds sur la totalité des périodes est nettement moins croissante, mais suit une courbe similaire à la précédente qui correspondait à deux périodes de temps successives.

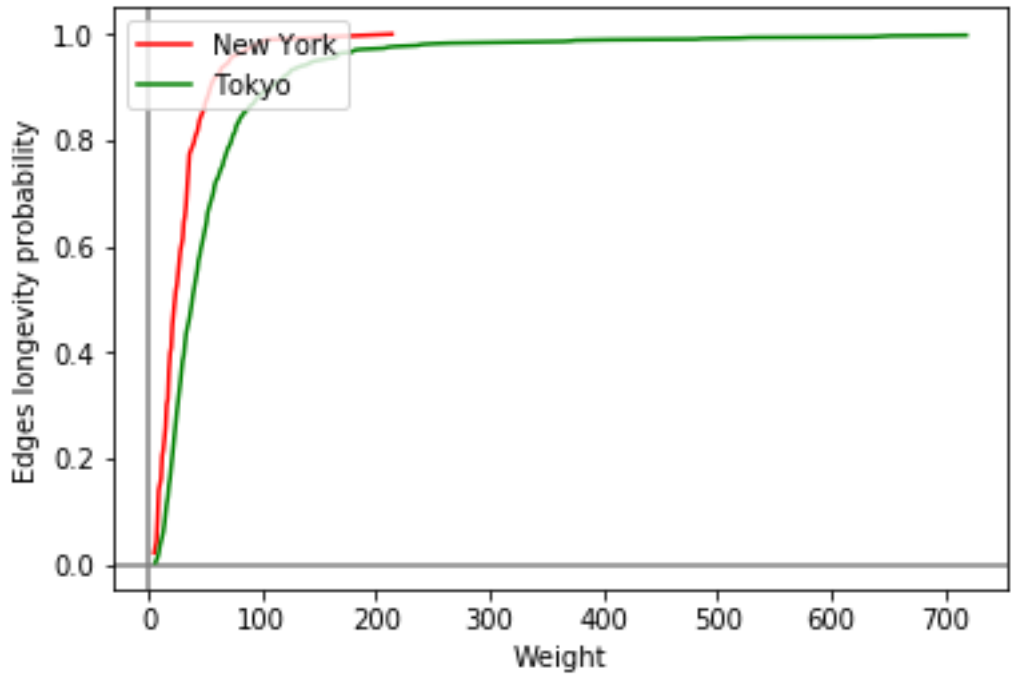

FIGURE 34 – Persistance des arêtes en fonction du poids de celles-ci sur la totalité des périodes de temps

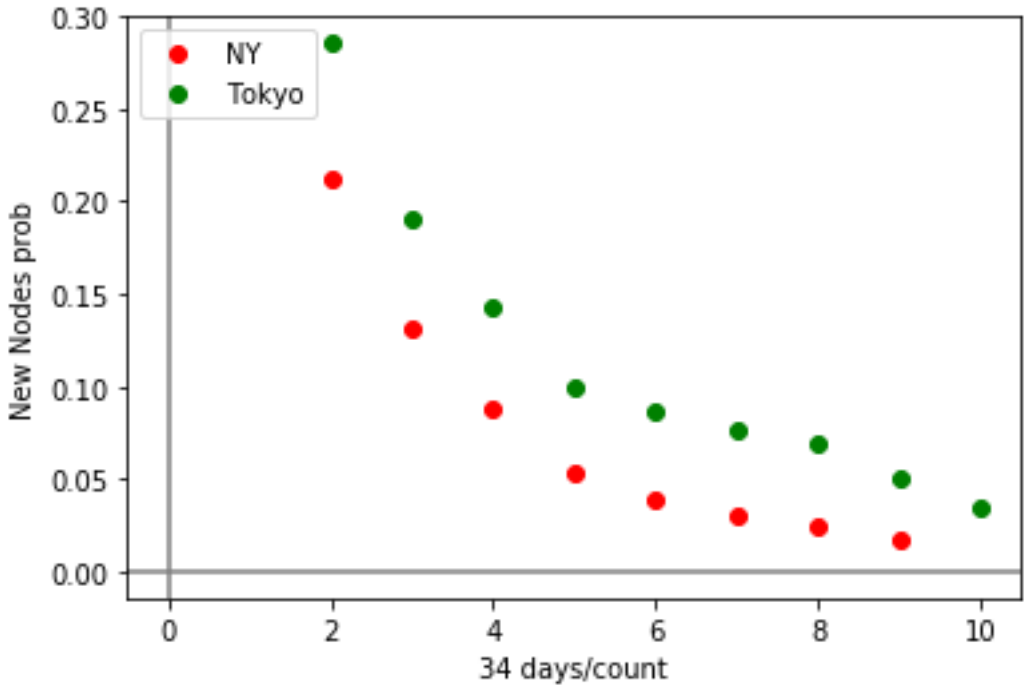

FIGURE 35 – Persistance des arêtes au fur et à mesure du temps

2.2.10 Persistance des noeuds au fur et à mesure du temps

On peut observer que la persistance des noeuds est loin derrière la théorie également à cause de l'insuffisance des données(figure 35). La persistance des noeuds au début se situe aux

alentours de 21% pour New York et 29% pour Tokyo et après 306 jours aux alentours de 5% pour Tokyo et 1% pour New York. La longévité des noeuds est donc très loin des 90% pour les 18 mois de la théorie, vu que notre nombre de données est restreint par rapport à la taille de la région étudiée.

2.2.11 Conclusions

2.3 Adaptation et seconde analyse de nos données

La quantité de données étant réduites pour la taille de la région étudiée, il devient important de les remodeler afin que nos algorithmes de prédiction d'arêtes puissent être aussi efficaces. En effet, étant donné le grand nombre d'apparitions de nouveaux noeuds, les graphiques ont une tendance nettement plus imprévisible que les cas présentés en théorie. Il devient donc important de réduire le nombre de noeuds afin que ceux-ci deviennent efficaces. Cependant, au vu des graphes, l'on peut dire que la répartition des noeuds et des arêtes correspond bien à un sous-ensemble représentant correctement les données complètes de Foursquare.

1. Le type de lieu

   Soit nous considérons que nos données sont un type de lieu comme nous l'avons étudié au début et nous cherchons à prédire de quel type de noeud vers quel type de noeud les utilisateurs se dirigent.
   Par exemple, les liens pourraient nous prédire qu'après un journée de travail la plupart des utilisateurs se dirigent vers les bars ou leur maison ou qu'une fois l'heure du repas, ils se dirigent en général vers les lieux de restauration.
   Cependant si l'on souhaite se baser sur des algorithmes utilisant la distance comme mesure de proximité, cela ne pourra pas fonctionner.

2. Nettoyage des données

   Soit nous retirons les noeuds et les arêtes pour lesquels nous manquons de données.
   Cela pourrait nous donner de bons algorithmes de prédictions d'arêtes, mais réduirait encore nos données.
   Cependant cela nous permettrait de conserver la totalité des caractéristiques des noeuds gardés.

3. Couple latitude longitude correspondant à une zone définie

   Soit nous utilisons le couple latitude et longitude des points comme couple désignant un lieu en réduisant un noeud à une zone latitude,longitude définie. Par exemple : Nous pouvons définir tous les endroits se situant entre la latitude 87.54 et 87.55 et entre la longitude 57.35 et 57.36 comme un seul noeud. Ce qui permettrait de garder la notion de distance entre deux noeuds, car nous pouvons calculer cette distance à partir de ces données.
   Cependant par cette méthode nous perdons le type de lieu précis étant donné que plusieurs types de lieux se trouvent dans une zone même si la plupart des grandes villes ont des quartiers résidentiels, de bars, de restaurants, de bureaux ...

En vue d'obtenir des résultats corrects en fonction de la quantité de données, nous avons décidé de définir une zone comprise entre deux latitudes et deux longitudes comme un noeud correspondant à tous les lieux qui se situent dans cette zone. Afin de réduire le nombre de noeuds, nous avons décidé de définir chaque zone comprise entre X et Y pour la latitude et XX et YY pour la longitude où $|X - Y| = 0.01$ et $|XX - YY| = 0.01$. L'analyse de ces données correspondra moins au cas théorique car les noeuds ont changé et ne correspondent plus totalement à l'analyse faite dans l'article [21].

Notre nouveau graphe comprend donc 1668 zones et donc noeuds pour New York et 36334 arêtes. En ce qui concerne les données pour Tokyo, il y a 1194 noeuds pour 47407 arêtes

2.3.1 Distribution des noeuds en fonction de leur degré

Dans cette figure36, l'on peut voir ce que l'on a vu précédemment à savoir que comme la heavy-tailed répartition propre aux réseaux locaux reste présente pour la région de New York. Cependant en raffinant nos données l'on peut voir que celle de Tokyo apparaît beaucoup moins. Une fois les noeuds de degré supérieur à 20, l'on se rapproche davantage d'une ditribution heavy-tailed pour Tokyo. On peut voir également qu'elle diminue de façon décroissante selon une fonction se rapprochant de $x^{-1.3}$s

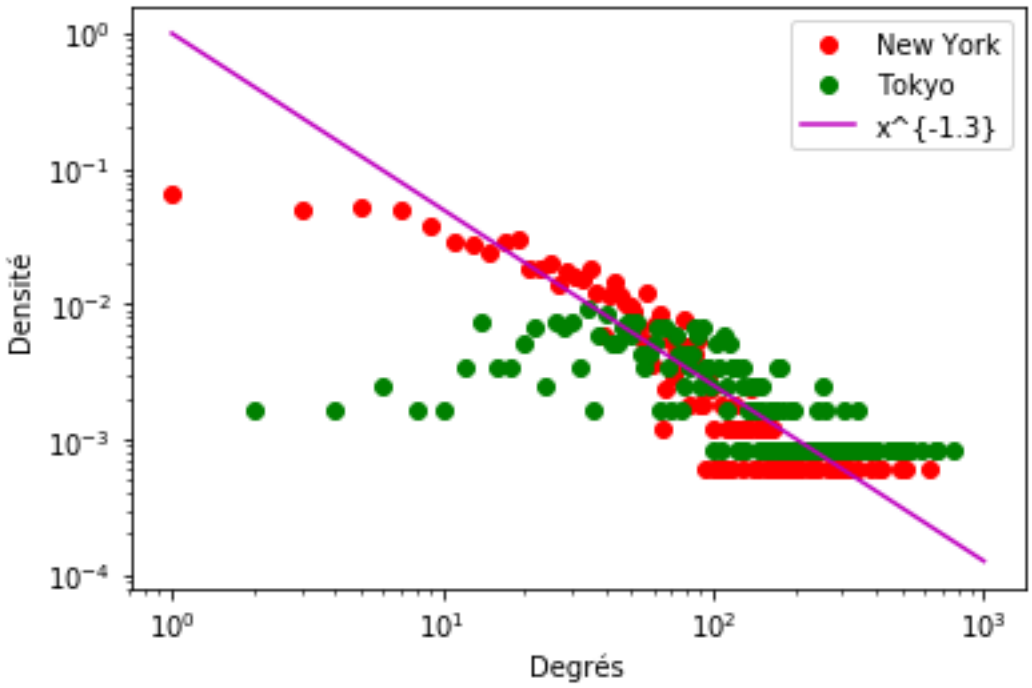

FIGURE 36 – Densité des noeuds en fonction de leur degré

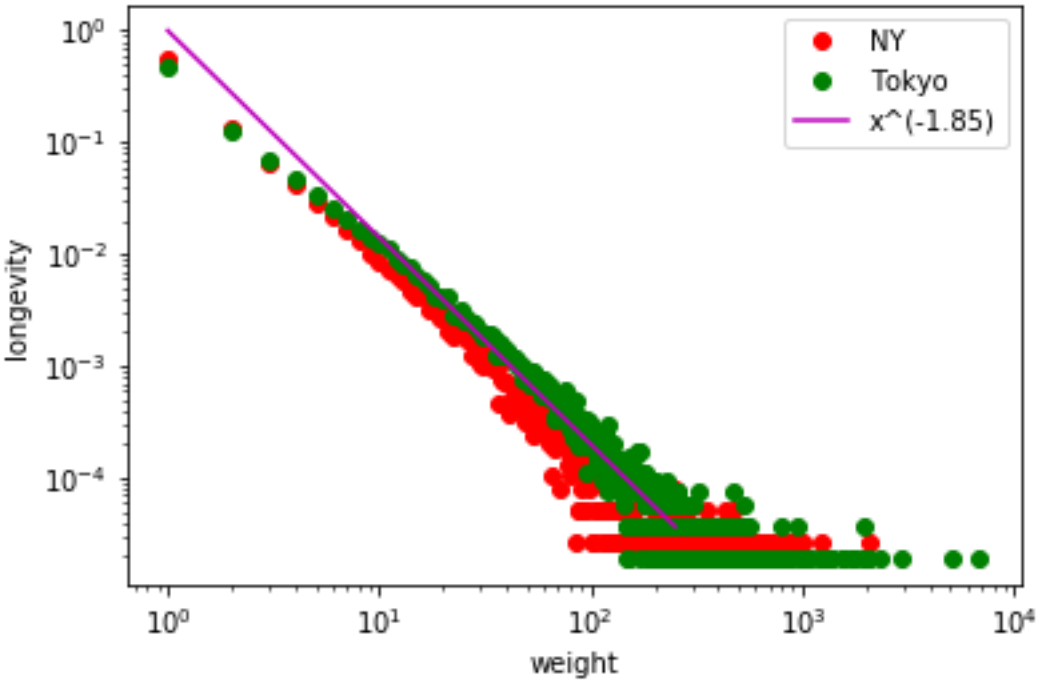

FIGURE 37 – Répartition des arêtes en fonction de leur poids pour la ville de New York.

2.3.2 Distribution des arêtes en fonction de leur poids

On peut voir que la répartition des arêtes en fonction de leur poids suit une loi exponentielle décroissante(figure 37).

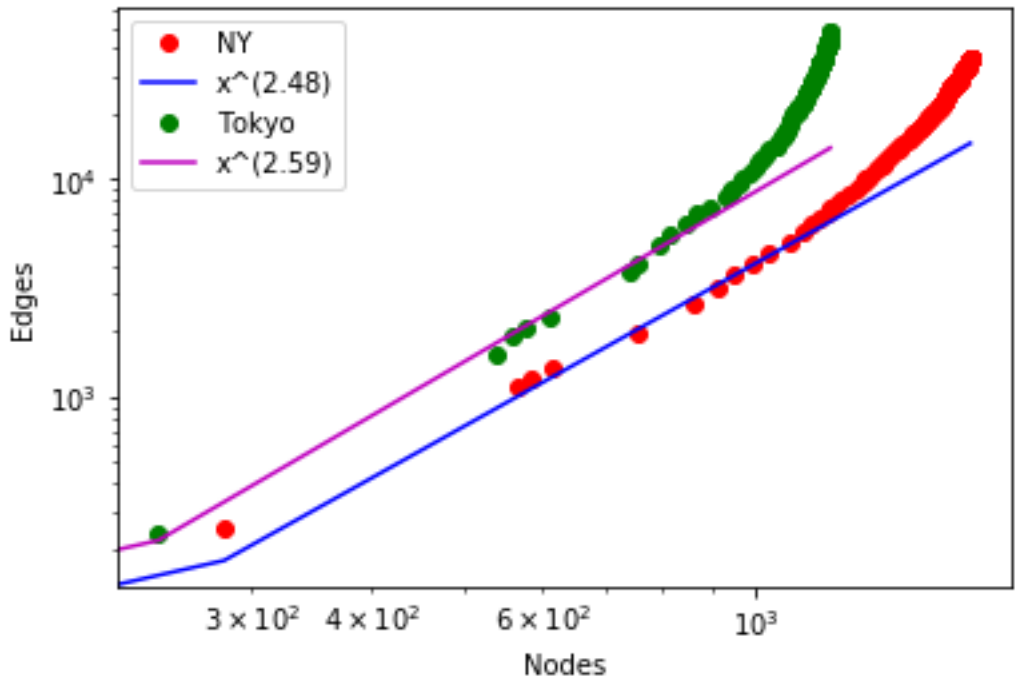

FIGURE 38 – Nombres de noeuds en fonction du nombre d'arêtes

Elle suit une exponentielle décroissante $x^{-1.85}$, ce qui est supérieur à la déviation analysée dans la création de notre graphe précédent.

2.3.3 Nombres de noeuds en fonction du nombres d'arêtes

Dans ce graphique(figure 38), l'on peut voir que les données de New-York et de Tokyo suivent la densification du réseau vue dans la partie théorie. En effet, le nombre d'arêtes augmente superlinéairement avec le nombre de noeuds. Donc, si on prend n(t) comme le nombre de noeuds observés au temps t et e(t) comme le nombre d'arêtes à ce même temps t, on a pour New York :

$$e(t) \propto 0.00015 n(t)^{2.48}$$

Et pour Tokyo :

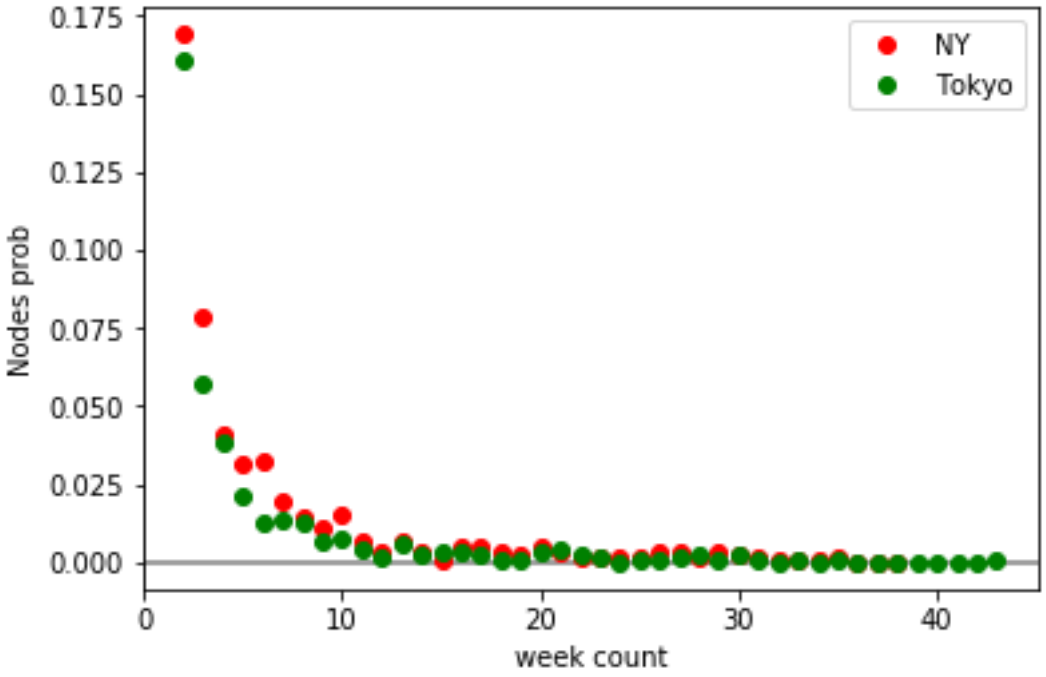

FIGURE 39 – Probabilité de découvrir un nouvel endroit sur le total d'endroits trouvés en fonction du nombre de semaines pour New York et Tokyo

$$e(t) \propto 0.00015 n(t)^{2.59}$$

Cela montre donc que le nombre de degrés va augmenter au fur et à mesure du temps car $\alpha > 1$. On peut voir que l'augmentation est nettement supérieure que précédemment. Cependant afin de faire correspondre la fonction exponentielle aux données, j'ai dû multiplier la fonction par 0.00015.

2.3.4 Probabilité de découvrir un nouvel endroit

La décroissance est plus nette que précédemment. On peut également dans cette figure 39 voir qu'après sept semaines plus de 98% des zones ont été découvertes. Le fait que l'on soit dans une zone délimitée fait qu'à partir d'un certain moment la population est allée dans toutes les zones définies. On peut voir que la nombre de nouveaux endroits continue de suivre une exponentielle décroissante.

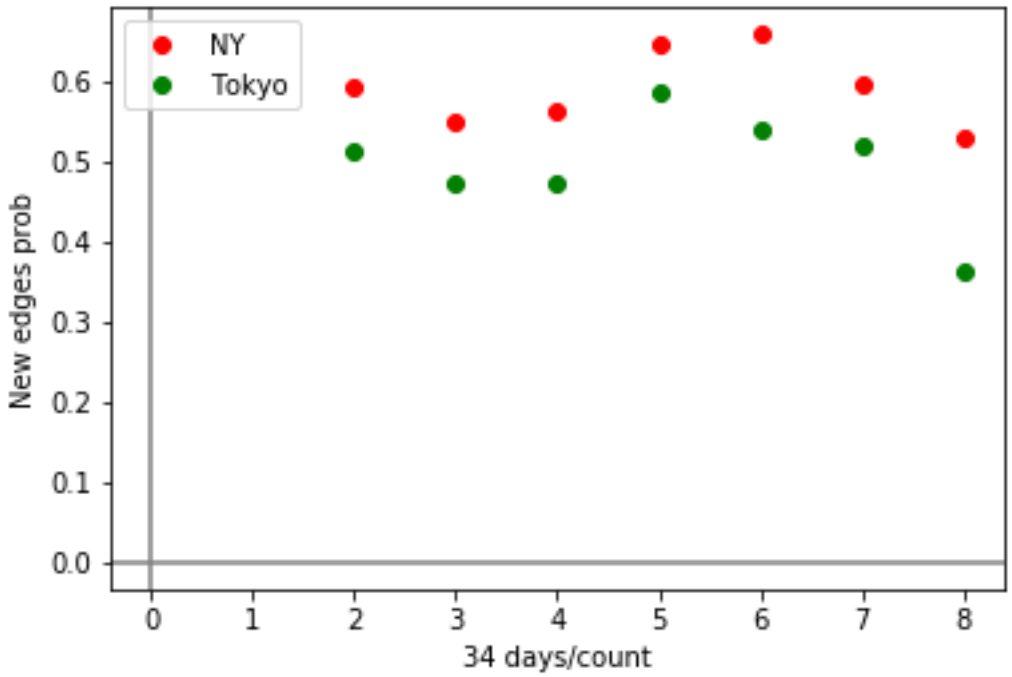

FIGURE 40 – Probabilité de découvrir une nouvelle arête entre deux endroits entre deux périodes de temps pour New York et Tokyo

2.3.5 Probabilité d'une nouvelle arête entre deux périodes de temps

On peut voir dans ce graphique(figure 40) que le nombre de nouvelles arêtes au fur et à mesure du temps a nettement diminué. En effet, la probabilité d'une nouvelle arête pour Tokyo se situe entre 0.4 et 0.6 et pour NY entre 0.5 et 0.7. Ce qui est nettement inférieur à notre graphe précédent. Etant donné que l'on a réduit la quantité de noeuds, la création d'arêtes est elle aussi réduite. La probabilité d'une nouvelle arête est donc diminuée.

2.3.6 Probabilité d'un nouveau noeud entre deux périodes de temps

Dans ce graphique(figure 41) l'on peut voir que la probabilité de découvrir un nouveau lieu entre deux périodes de temps colle parfaitement à la théorie pour la ville de Tokyo et est légèrement plus grande pour la ville de New York. Cela nous montre que pour la ville de New York malgré la délimitation en zone l'apparition d'une nouvelle arête reste élevée. Après 170 jours il y a près de 18% d'endroits qui n'ont pas été visités dans la période de temps précédente.
Au vu du graphique, on peut voir que la probabilité d'un nouveau noeud lors de la période de temps suivante est vraiment faible comparée à notre graphe précédent.

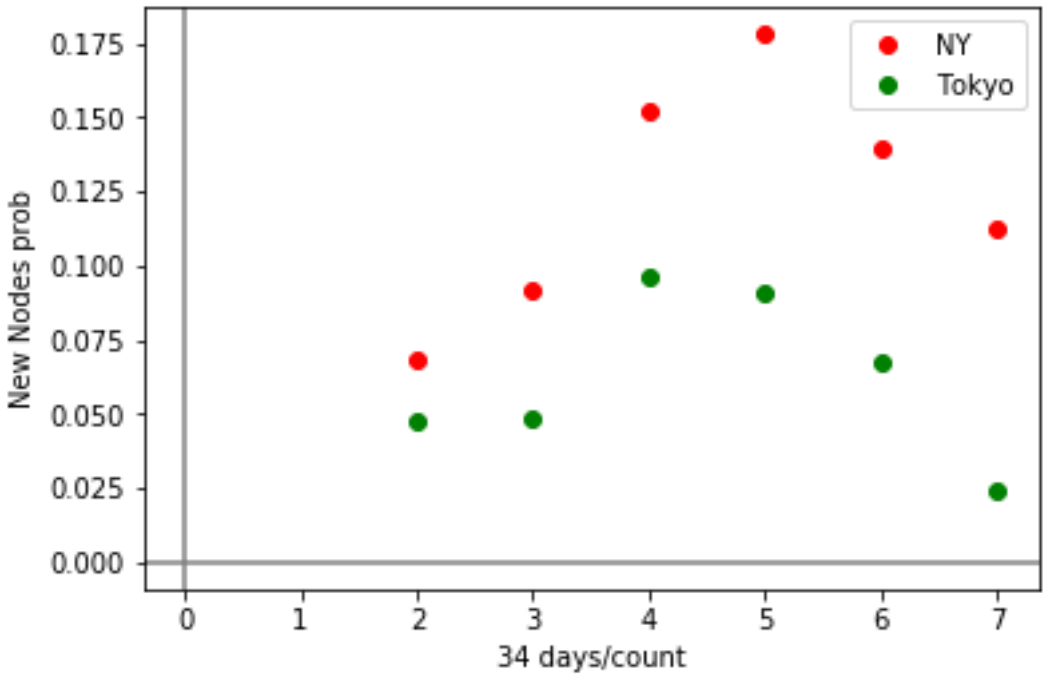

FIGURE 41 – Probabilité de découvrir un nouveau lieu entre deux périodes de temps pour New York et Tokyo

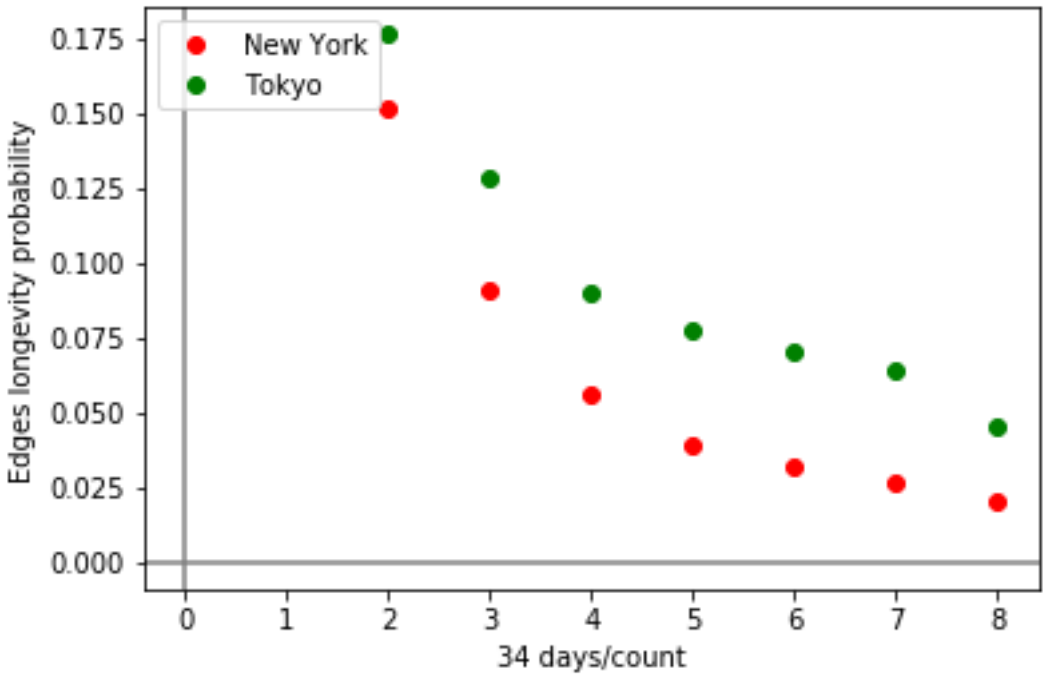

FIGURE 42 – Persistance des arêtes au fur et à mesure du temps

2.3.7 Persistance des arêtes au fur et à mesure du temps

Dans ce graphique(figure 42), on peut voir que la longévité des arêtes est nettement supérieure que précédemment. On peut voir qu'après 272 jours il y a 5% des arêtes présentes

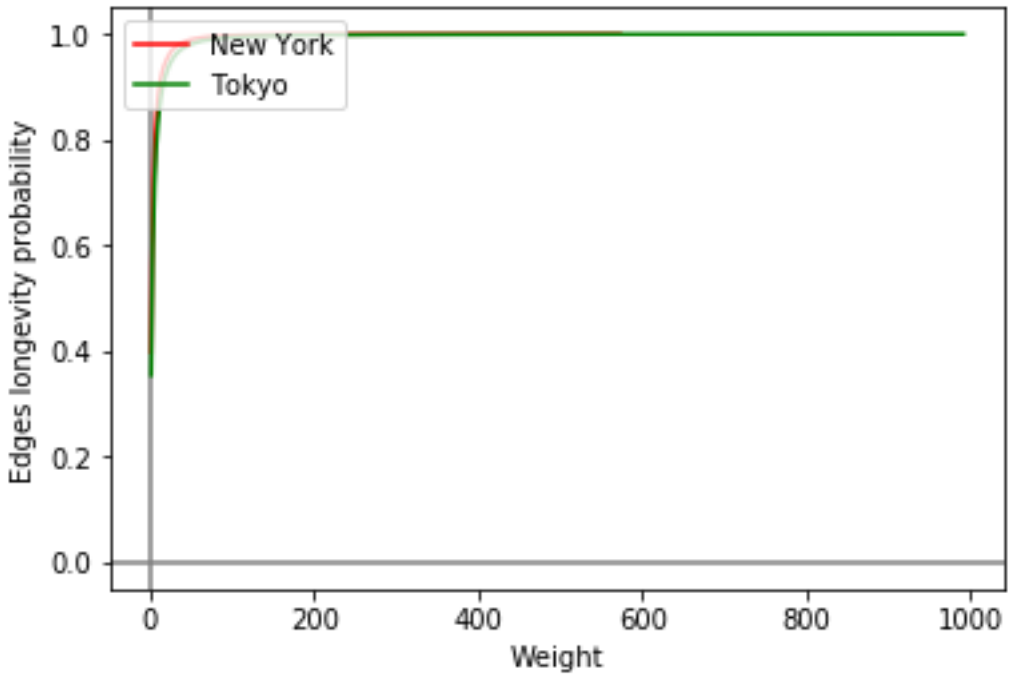

FIGURE 43 – Persistance des arêtes en fonction du poids de celles-ci entre deux périodes de temps successives

à chaque période de temps pour Tokyo et 2.5% des arêtes pour New York. Bien que l'on ait réduit notre nombre de noeuds, la longévité des arêtes reste assez faible.

2.3.8 Persistance des arêtes en fonction du poids de celles-ci sur deux périodes de temps successives

On peut voir que le graphique(figure 43) suit une courbe semblable à celle que l'on avait au graphique précédent, bien que celle-ci soit moins croissante.

2.3.9 Persistance des arêtes en fonction du poids de celle-ci sur la totalité des périodes de temps

On peut voir dans ce graphique(figure 44) que la persistance des noeuds sur la totalité des périodes est nettement moins croissante, mais suit une courbe similaire à la précédente qui correspondait à deux périodes de temps successives.
Si l'on compare ce graphique à celui du graphe précédent, il est moins croissant.

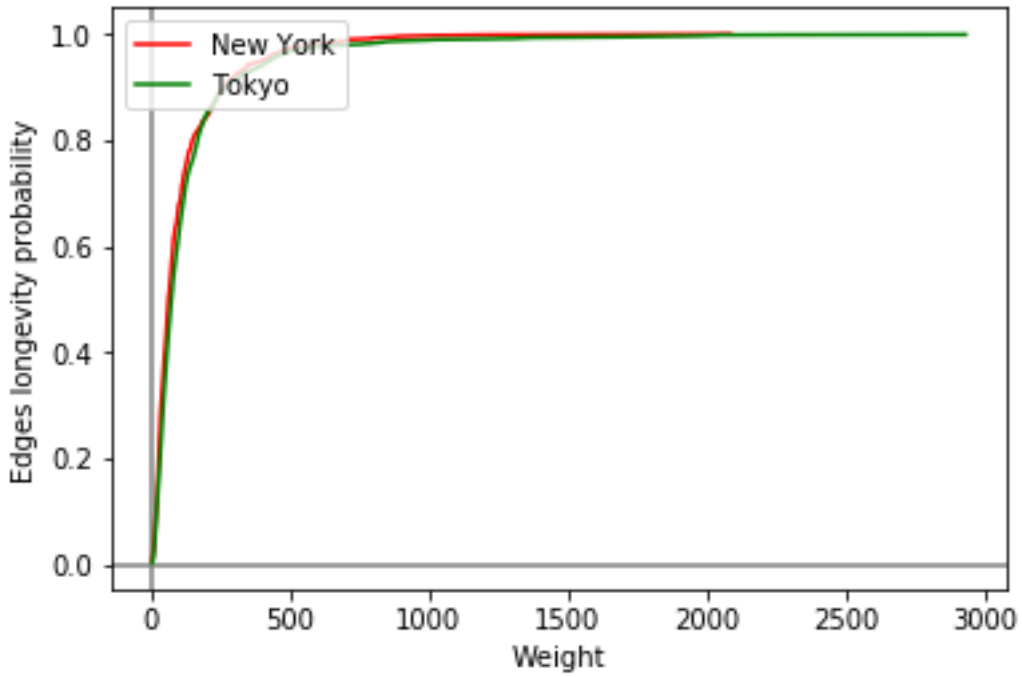

FIGURE 44 – Persistance des arêtes en fonction du poids de celles-ci sur la totalité du temps

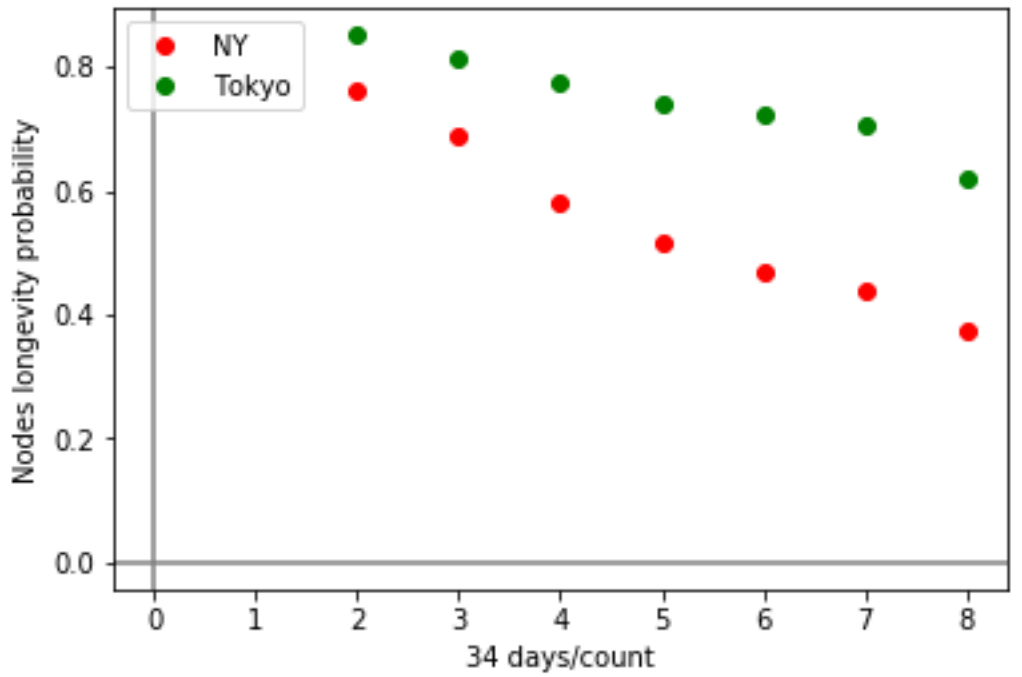

FIGURE 45 – Persistance des arêtes au fur est à mesure du temps

2.3.10 Persistance des noeuds au fur et à mesure du temps

On peut observer que la persistance des noeuds est de loin supérieure à ce que nous avons constaté précédemment dans la figure 45. Pour des périodes de 34 jours, l'on peut noter

que pour New York près de 40 % des noeuds de base sont restés présents lors des périodes de temps suivantes. En ce qui concerne la ville de Tokyo, cette probabilité est de l'ordre de 60 % et ce sur 272 jours. Grâce au fait que l'on a réduit la nombre de noeuds, la longévité des noeuds a nettement augmenté.

2.4 Conclusions

Dans un premier temps nous avons effectué une analyse globale de nos données.
Nous avons donc analysé les différents types d'endroits sur nos deux bases de données que sont les données Foursquare de Tokyo et New York. Nous avons vu une différence de mentalité dans ces pays expliquant les raisons de nos résultats. Nous avons aussi analysé le type d'endroits en fonction du poids des arêtes afin de comprendre les habitudes de nos utilisateurs. Nous avons aussi pu voir que peu d'utilisateurs ont contribué beaucoup à nos données, mais que beaucoup y ont peu contribué.Nous avons aussi vu que l'on avait un résultat semblable pour le nombre d'entrées par lieux.

Ensuite, nous avons pu voir que nos données, bien que respectant quelques propriétés, ne pouvaient totalement les satisfaire, car elles sont trop faibles par rapport à la taille de la région étudiée.

Afin d'avoir des résultats efficaces, nous avons décidé d'aménager nos données afin que les noeuds de notre graphique correspondent à une zone comprise entre un couple de latitude et un couple de longitude.

Avec cette modification de nos données, nos algorithmes devraient être en mesure de prédire les arêtes au mieux.

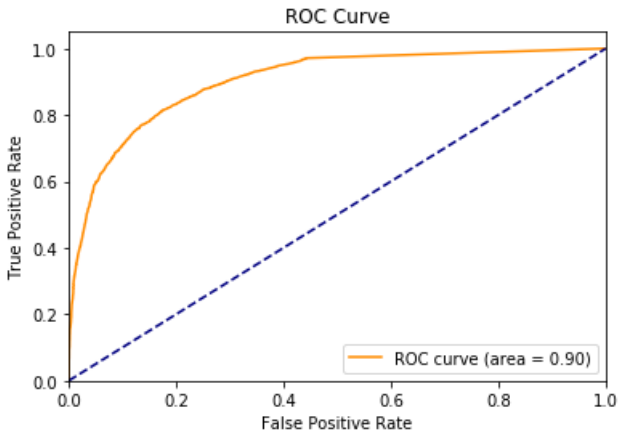

FIGURE 46 – Courbe ROC de New York pour 85 % des données avec l'algorithme utilisant le coefficient de Jaccard pour la prédiction des arêtes

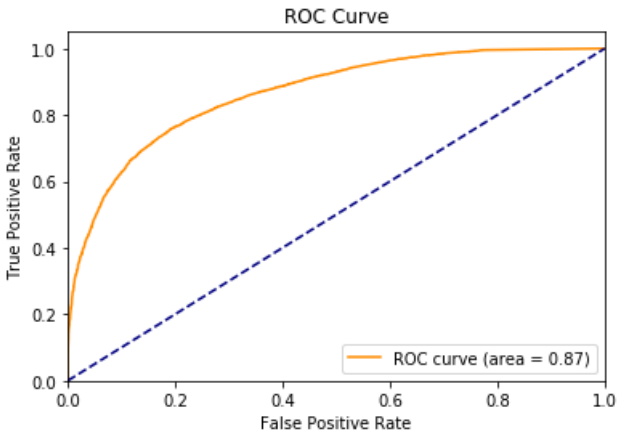

FIGURE 47 – Courbe ROC de Tokyo pour 85 % des données avec l'algorithme utilisant le coefficient de Jaccard pour la prédiction des arêtes

3 Prédiction d'arêtes

Dans cette section, nous allons nous servir des algorithmes exposés dans la partie théorique en vue de prédire les arêtes de notre graphe. Il existe de nombreux autres algorithmes basés sur des méthodes de machines learning ou des méthodes globales non étudiées dans ce travail.

Nous allons appliquer ces méthodes sur nos données Foursquare de Tokyo et de New York que nous avons modifiées comme présenté dans la partie précédente.

Nous allons faire une analyse complète en vue de prédire les arêtes en ne gardant que 85%,60% et 35% de ces arêtes et en voyant comme notre algorithme complète les déplacements manquants afin d'évaluer s'il est capable de prévoir les déplacements de population les plus probables.

Pour établir ces algorithmes, nous nous sommes servis des librairies supplémentaires que sont scikit-learn [22],tenserflow, numpy et gae (Graph auto-Encoders)[12]

3.1 A l'aide du coefficient de Jaccard

1. ROC curve et AUC score avec 85% des données

Score AUC pour New York= 0.9016608059539181

Score de précision moyenne pour New York = 0.8979399252714186

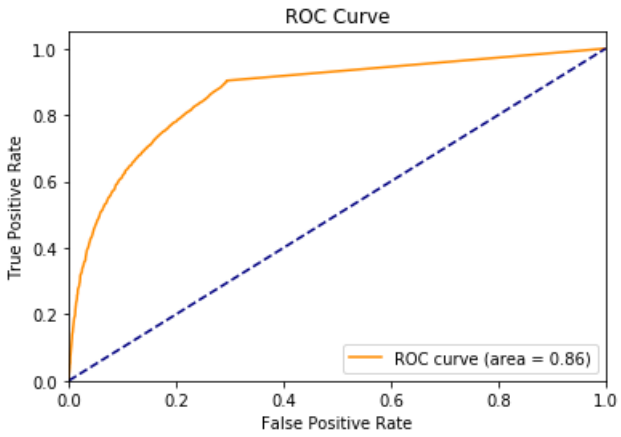

FIGURE 48 – Courbe ROC de New York pour 60 % des données avec l'algorithme utilisant le coefficient de Jaccard pour la prédiction des arêtes

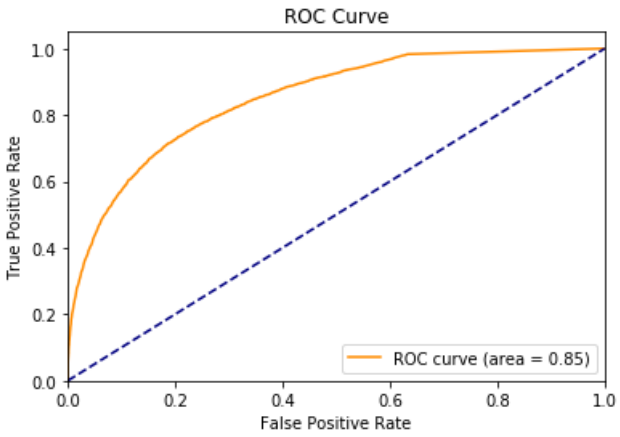

FIGURE 49 – Courbe ROC de Tokyo pour 60 % des données avec l'algorithme utilisant le coefficient de Jaccard pour la prédiction des arêtes

Score AUC pour Tokyo= 0.8650725147048163

Score de précision moyenne pour Tokyo = 0.8704388423353553

2. ROC curve et AUC score avec 60% des données

Score AUC pour New York= 0.8639584360420463

Score de précision moyenne pour New York = 0.8485863907673963

Score AUC pour Tokyo= 0.8495086657627766

Score de précision moyenne pour Tokyo = 0.8485794348267268

3. ROC curve et AUC score avec 35% des données

Score AUC pour New York= 0.7640499917020834

Score de précision moyenne pour New York = 0.7194414882790211

Score AUC pour Tokyo= 0.7889208940866229

Score de précision moyenne pour Tokyo = 0.7375698642127473

4. Analyse des résultats obtenus

Dans ces deux premiers graphiques(figure 46 et 47) et résultats, on peut voir que l'algorithme de prédiction d'arêtes utilisant le coefficient de Jaccard a un score à AUC se situant entre 0.86 et 0.9 et a une précision moyenne entre 0.87 et 0. 89. Ce qui correspond

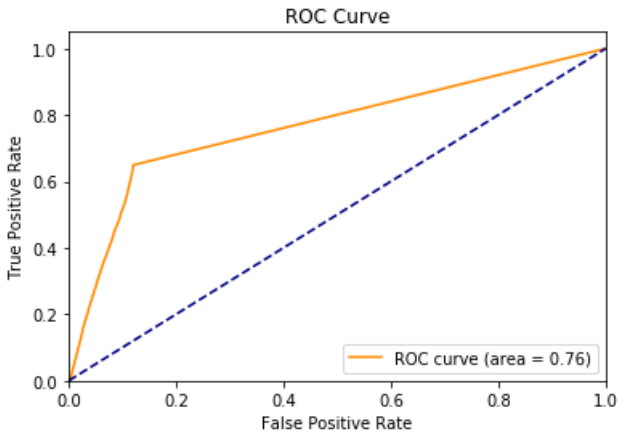

FIGURE 50 – Courbe ROC de New York pour 35 % des données avec l'algorithme utilisant le coefficient de Jaccard pour la prédiction des arêtes

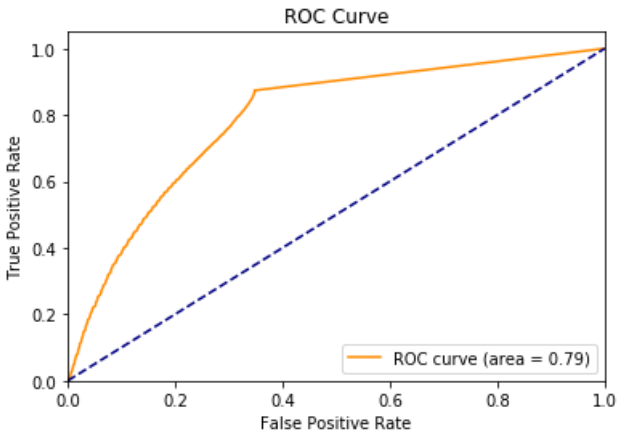

FIGURE 51 – Courbe ROC de Tokyo pour 35 % des données avec l'algorithme utilisant le coefficient de Jaccard pour la prédiction des arêtes

à de très hauts score. Cependant l'algorithme s'est basé sur 85 % des résultats, ce qui est beaucoup.

Lorsque l'algorithme se base sur moins de données(figure 48 et 49), on peut voir qu'il a perdu 0.02 à 0.04 au score AUC et 0.03 à 0.05 pour le score de précision moyenne en se basant sur 25 % de données en moins.

Enfin lorsque l'algorithme se base sur 35% des données(figure 50 et 51), on peut voir que le score AUC a perdu entre 0.06 et 0.1 et le score de précision moyenne a perdu entre 0.11 et 0.13 par rapport au cas précédent.

On peut voir que l'algorithme est nettement influencé par l'insuffisance des données.

De plus on peut remarquer que lorque l'algorithme se base sur 85% des données la courbe est logarithmique ; cependant lorsque le nombre de données devient insuffisant, on peut voir que le maximum de vraisemblance se rapproche de 1 à un moment précis. Lorsque l'algorithme utilise 60% des données, le maximum de vraisemblance se rapproche nettement de 1 en 0.3 et lorsqu'il utilise 35% des données, ce point se situe en 0.15 pour le taux de faux positifs.

3.2 A l'aide de l'index d'allocation de ressources

1. ROC curve et AUC score avec 85% des données

Score AUC pour New York= 0.9380672735243015

Score de précision moyenne pour New York = 0.9390843621410436

Score AUC pour Tokyo= 0.9334822545814299

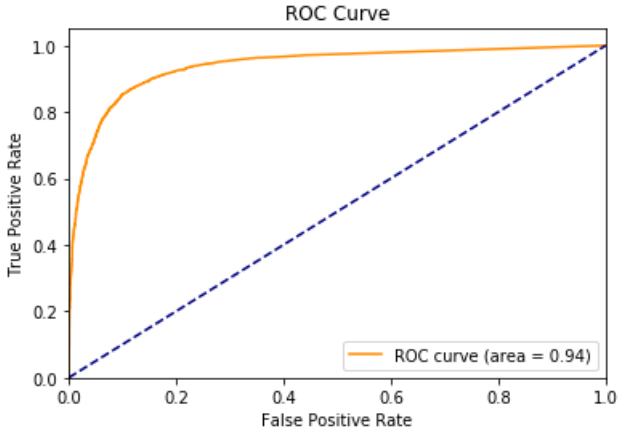

FIGURE 52 – Courbe ROC de New York pour 85 % des données avec l'algorithme utilisant le coefficient de Jaccard pour la prédiction des arêtes

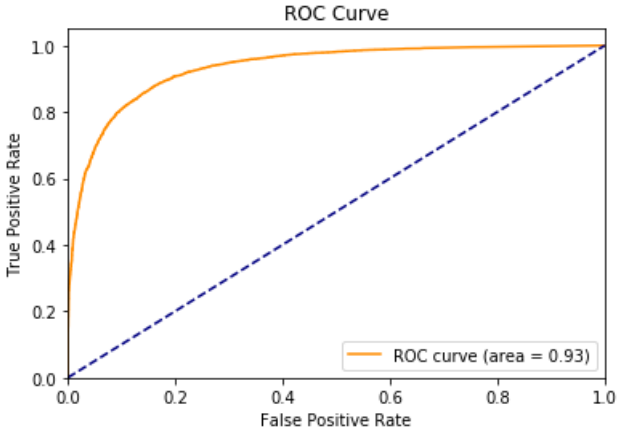

FIGURE 53 – Courbe ROC de Tokyo pour 85 % des données avec l'algorithme utilisant le coefficient de Jaccard pour la prédiction des arêtes

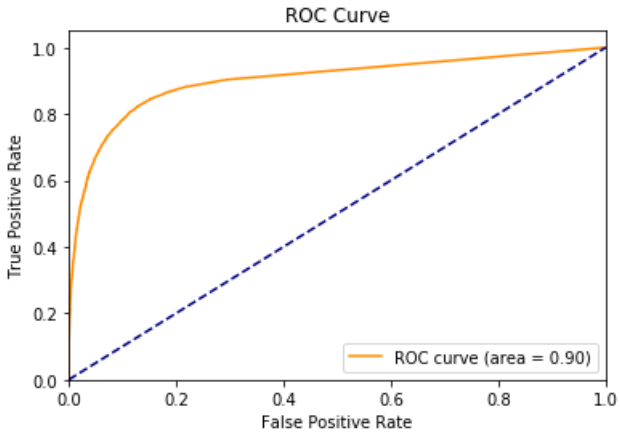

FIGURE 54 – Courbe ROC de New York pour 60 % des données avec l'algorithme utilisant le coefficient de Jaccard pour la prédiction des arêtes

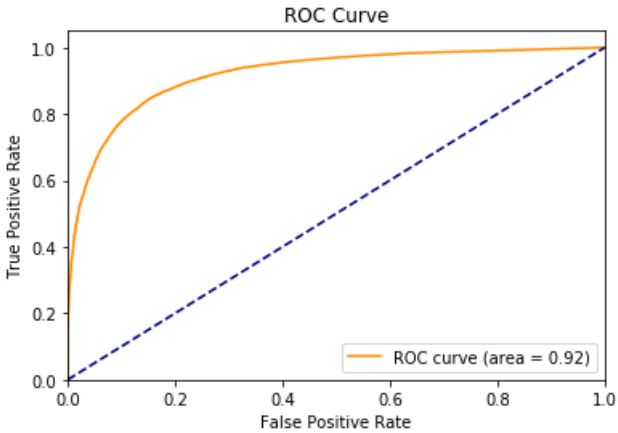

FIGURE 55 – Courbe ROC de Tokyo pour 60 % des données avec l'algorithme utilisant le coefficient de Jaccard pour la prédiction des arêtes

Score de précision moyenne pour Tokyo = 0.9345058665016697

2. ROC curve et AUC score avec 60% des données

Score AUC pour New York= 0.9010044894680832

Score de précision moyenne pour New York = 0.9054929174422887

Score AUC pour Tokyo= 0.9201721483636893

Score de précision moyenne pour Tokyo = 0.9240382183078969

3. ROC curve et AUC score avec 35% des données

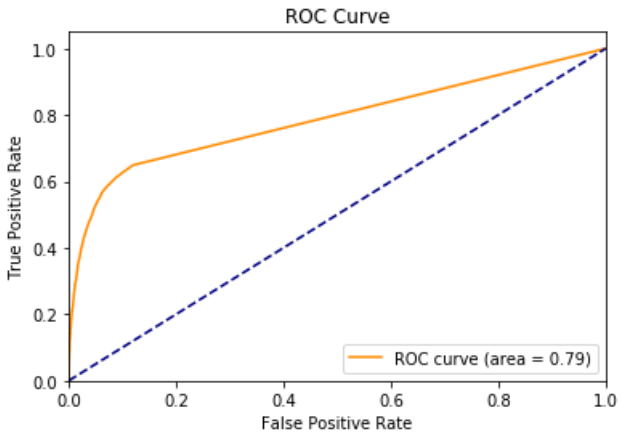

FIGURE 56 – Courbe ROC de New York pour 35 % des données avec l'algorithme utilisant le coefficient de Jaccard pour la prédiction des arêtes

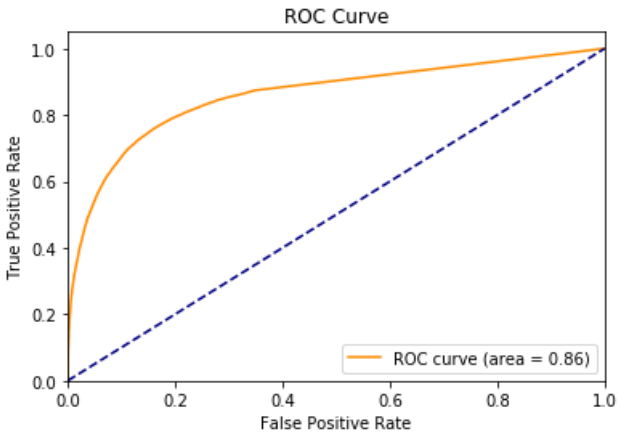

FIGURE 57 – Courbe ROC de Tokyo pour 35 % des données avec l'algorithme utilisant le coefficient de Jaccard pour la prédiction des arêtes

Score AUC pour New York= 0.7863618017195008

Score de précision moyenne pour New York = 0.7899203707883597

Score AUC pour Tokyo= 0.8595110077903809

Score de précision moyenne pour Tokyo = 0.8656442775890387

4. Analyse des résultats obtenus

Dans ces deux premiers graphiques(figure 52 et 53) et résultats, on peut voir que l'algorithme de prédiction d'arêtes utilisant l'index d'allocation de ressources a un score à AUC se situant aux alentours de 0.93 et a une précision moyenne proche de0.93. Ce qui correspond à de très hauts scores, meilleurs que celui obtenu par l'algorithme du coefficient de Jaccard. Lorsque l'algorithme se base sur moins de données (figure 54 et 55), on peut voir qu'il a perdu 0.01 à 0.03 au score AUC et 0.01 à 0.03 pour le score de précision moyenne en se basant sur 25 % de données en moins.

Enfin lorsque l'algorithme se base sur 35% des données(figure 56 et 57), on peut voir que le score AUC a perdu entre 0.07 et 0.12 et le score de précision moyenne a perdu entre 0.06 et 0.12.

On peut voir que l'algorithme est nettement influencé par l'insuffisance des données.

Cependant on obtient dans tous les cas de meilleurs résultats que le coefficient de Jaccard.

On peut également noter que l'algorithme est nettement moins influencé pour la ville de Tokyo : cela s'explique par le fait que le nombre de données de base est supérieur et

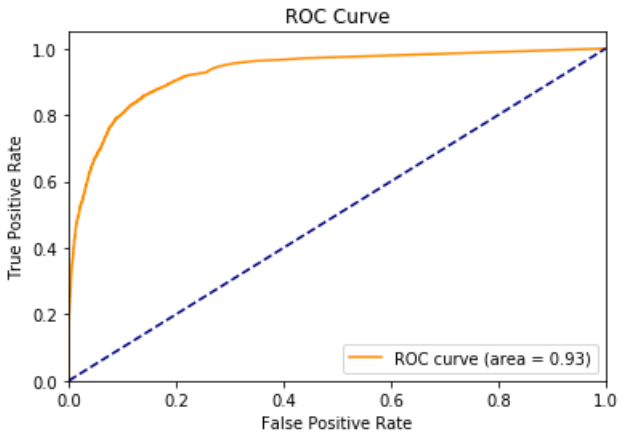

FIGURE 58 – Courbe ROC de New York pour 85 % des données avec l'algorithme utilisant le coefficient de Jaccard pour la prédiction des arêtes

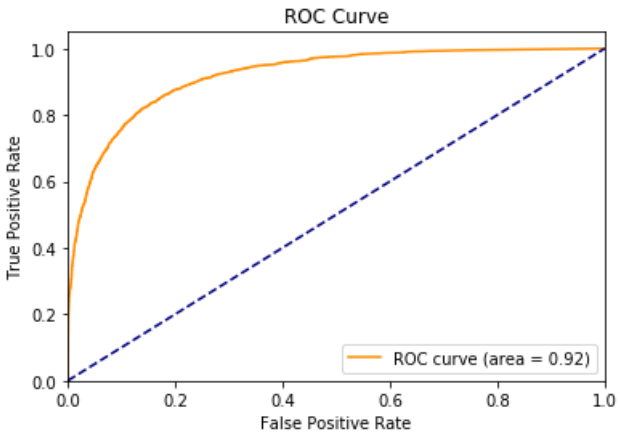

FIGURE 59 – Courbe ROC de Tokyo pour 85 % des données avec l'algorithme utilisant le coefficient de Jaccard pour la prédiction des arêtes

permet donc une meilleur prédiction pour l'algorithme.

3.3 A l'aide de l'indicateur AdamicAdar

1. ROC curve et AUC score avec 85% des données

   Score AUC pour New York= 0.9291451713172801
   Score de précision moyenne pour New York= 0.9300993647397974
   Score AUC pour Tokyo= 0.9194311340628848
   Score de précision moyenne pour Tokyo = 0.9207091773539684

2. ROC curve et AUC score avec 60% des données

   Score AUC pour New York= 0.8973242333232037
   Score de précision moyenne pour New York = 0.9017193078442631
   Score AUC pour Tokyo= 0.91178841439862471
   Score de précision moyenne pour Tokyo = 0.9153577544601862

3. ROC curve et AUC score avec 35% des données

   Score AUC pour New York= 0.786714485099952
   Score de précision moyenne pour New York = 0.7924258960134231
   Score AUC pour Tokyo= 0.8590082095223877

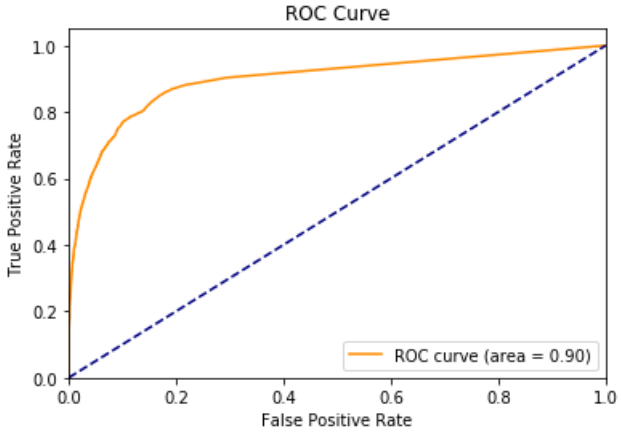

FIGURE 60 – Courbe ROC de New York pour 60 % des données avec l'algorithme utilisant le coefficient de Jaccard pour la prédiction des arêtes

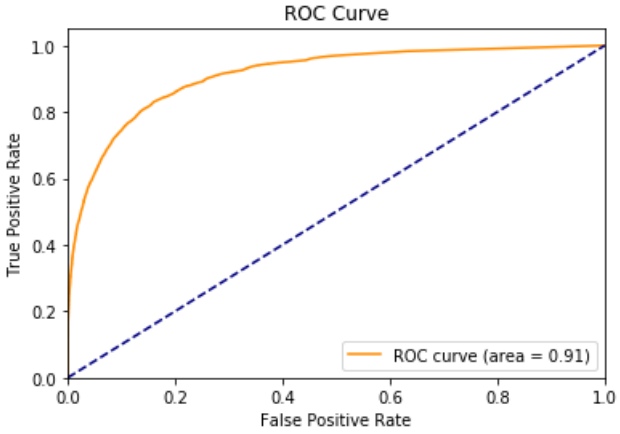

FIGURE 61 – Courbe ROC de Tokyo pour 60 % des données avec l'algorithme utilisant le coefficient de Jaccard pour la prédiction des arêtes

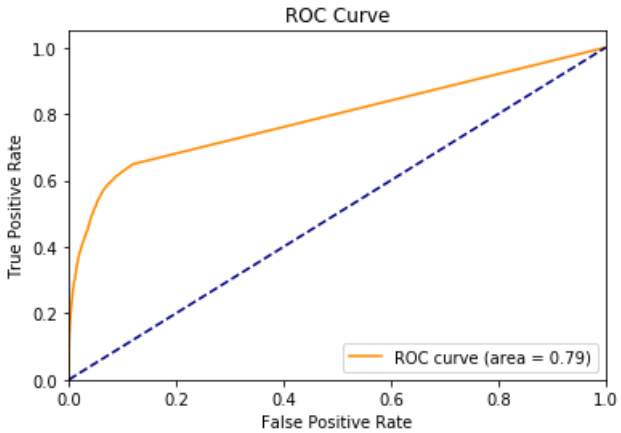

FIGURE 62 – Courbe ROC de New York pour 35 % des données avec l'algorithme utilisant le coefficient de Jaccard pour la prédiction des arêtes

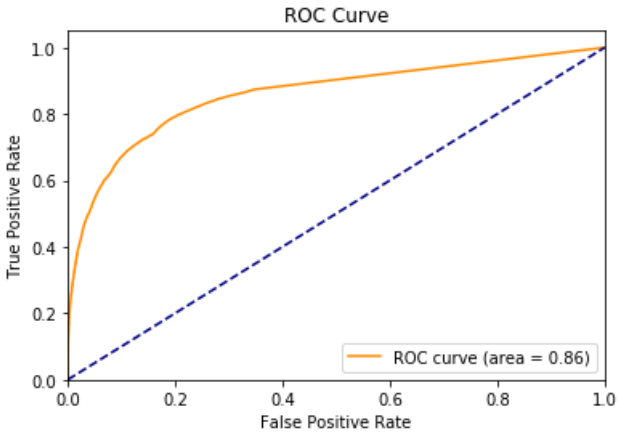

FIGURE 63 – Courbe ROC de Tokyo pour 35 % des données avec l'algorithme utilisant le coefficient de Jaccard pour la prédiction des arêtes

Score de précision moyenne pour Tokyo = 0.8665563141594074

4. Analyse des résultats obtenus

Dans ces deux premiers graphiques(figure 58 et 59) et résultats, on peut voir que l'algorithme de prédiction d'arêtes utilisant l'index d'allocation de ressources a un score AUC se situant entre 0.91 et 0.92 et a une précision moyenne se situant entre 0.92 et 0.93. Ce qui correspond à de très hauts scores, meilleurs que celui obtenu par l'algorithme du coefficient de Jaccard, mais moins bien que celui de l'index d'allocation. Lorsque l'algorithme se base sur moins de données(figure 60 et 61), on peut voir qu'il a perdu 0.008 à

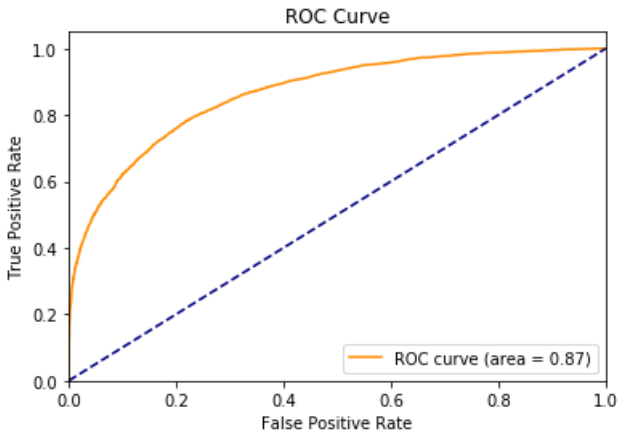

FIGURE 64 – Courbe ROC de New York pour 85 % des données avec l'algorithme utilisant le coefficient de Jaccard pour la prédiction des arêtes

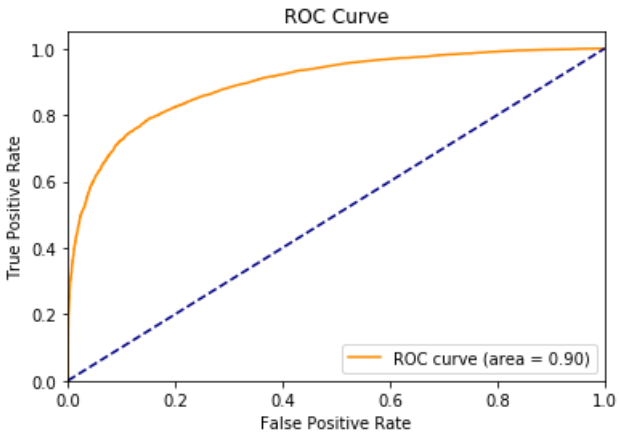

FIGURE 65 – Courbe ROC de Tokyo pour 85 % des données avec l'algorithme utilisant le coefficient de Jaccard pour la prédiction des arêtes

0.03 au score AUC et 0.005 à 0.03 pour le score de précision moyenne en se basant sur 25 % de données en moins.

Enfin lorsque l'algorithme se base sur 35% des données (figure 62 et 63), on peut voir que le score AUC a perdu entre 0.06 et 0.11 et le score de précision moyenne a perdu entre 0.05 et 0.11.

On peut voir que l'algorithme est influencé par l'insuffisance des données.

Cependant on obtient dans tout les cas de meilleurs résultats qu'avec le coefficient de Jaccard et des résultats proches de l'index d'allocation de ressources.

3.4 A l'aide du score d'attachement préférentiel

1. ROC curve et AUC score avec 85% des données

   Score AUC pour New York= 0.8656160389613567

   Score de précision moyenne pour New York = 0.8747092921078394

   Score AUC pour Tokyo= 0.896013991975332

   Score de précision moyenne pour Tokyo = 0.9054347921079361

2. ROC curve et AUC score avec 60% des données

   Score AUC pour New York= 0.8626181383606623

   Score de précision moyenne pour New York = 0.8713262755791722

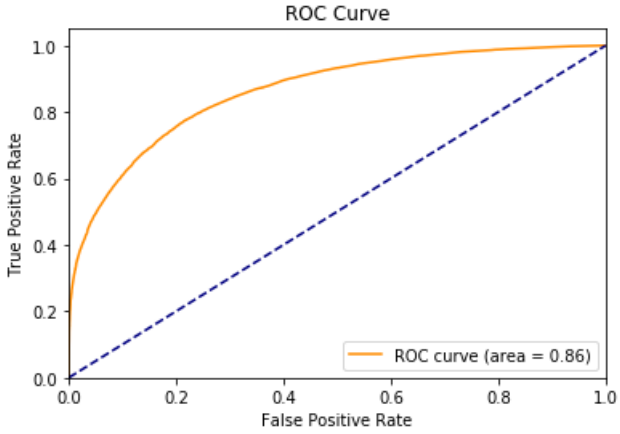

FIGURE 66 – Courbe ROC de New York pour 60 % des données avec l'algorithme utilisant le coefficient de Jaccard pour la prédiction des arêtes

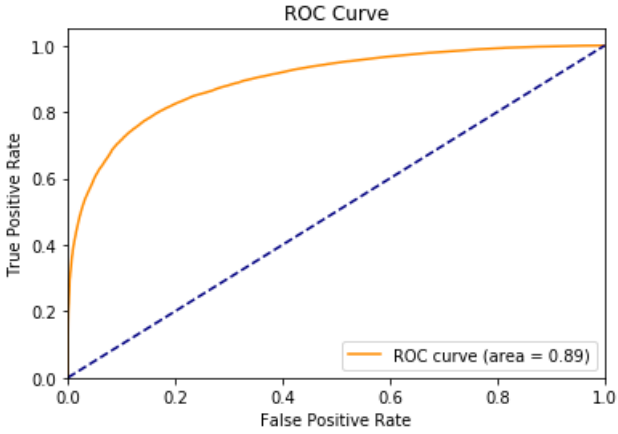

FIGURE 67 – Courbe ROC de Tokyo pour 60 % des données avec l'algorithme utilisant le coefficient de Jaccard pour la prédiction des arêtes

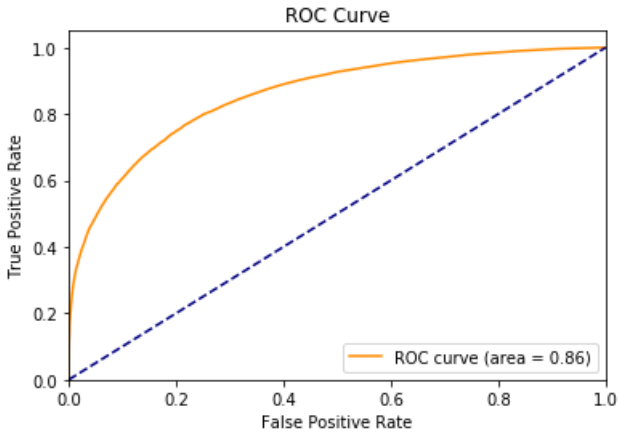

FIGURE 68 – Courbe ROC de New York pour 35 % des données avec l'algorithme utilisant le coefficient de Jaccard pour la prédiction des arêtes

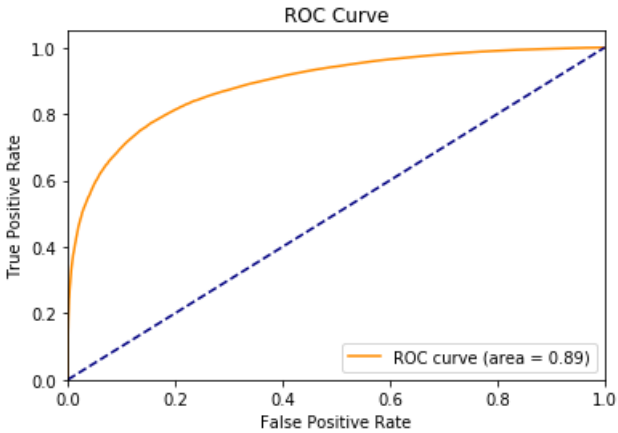

FIGURE 69 – Courbe ROC de Tokyo pour 35 % des données avec l'algorithme utilisant le coefficient de Jaccard pour la prédiction des arêtes

Score AUC pour Tokyo= 0.89489097862472

Score de précision moyenne pour Tokyo = 0.9049125700156188

3. ROC curve et AUC score avec 35% des données

Score AUC pour New York= 0.8582349745243039

Score de précision moyenne pour New York =0.8674288006115816

Score AUC pour Tokyo= 0.8891663126674508

Score de précision moyenne pour Tokyo = 0.8996848133060692

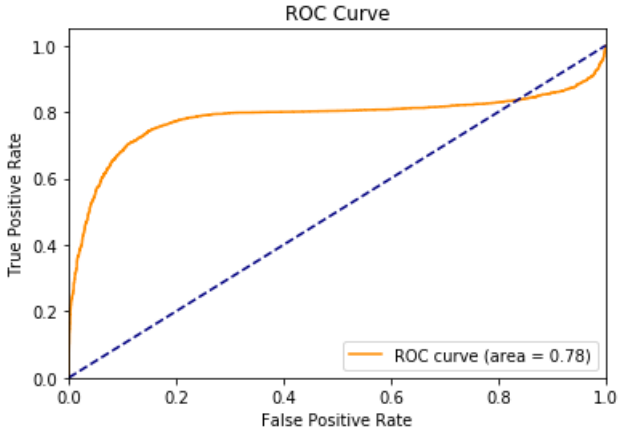

FIGURE 70 – Courbe ROC de New York pour 85 % des données avec l'algorithme utilisant le coefficient de Jaccard pour la prédiction des arêtes

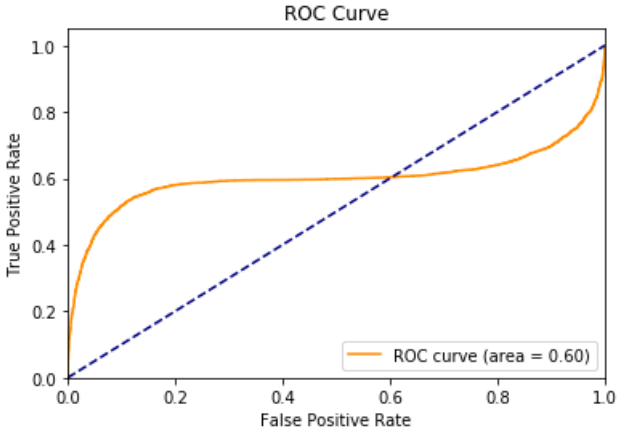

FIGURE 71 – Courbe ROC de Tokyo pour 85 % des données avec l'algorithme utilisant le coefficient de Jaccard pour la prédiction des arêtes

4. Analyse des résultats obtenus

Dans ces deux premiers graphiques(figure 64 et 65) et résultats, on peut voir que l'algorithme de prédiction d'arêtes utilisant l'index d'allocation de ressources a un score AUC se situant entre 0.86 et 0.89 et a une précision moyenne se situant entre 0.87 et 0.90. On peut remarquer que les scores AUC sont légèrement inférieurs à celui de Jaccard, cependant le score de précision moyen est légèrement meilleur que ce dernier.
Lorsque l'algorithme se base sur moins de données (figure 66 et 67), on peut voir qu'il n'a quasiment rien perdu quant à son score AUC et à sa précision moyenne.
Enfin lorsque l'algorithme se base sur 35% des données(figure 68 et 69), on peut voir que le score AUC et de précision a perdu une valeur proche de 0.01.
On peut voir que cet algorithme basé sur la centralité est beaucoup plus stable que le précédent lorsque le nombre de données sur lesquelles se base l'algorithme est réduit.
Cet algorithme est donc intéressant par le fait qu'il permet d'obtenir des résultats efficaces malgré l'absence de données sur lesquelles se base l'algorithme.

3.5 A l'aide du spectre du graphe Laplacien normalisé

1. ROC curve et AUC score avec 85% des données

Score AUC pour New York= 0.7848572716691724
Score de précision moyenne pour New York = 0.8468621948538406

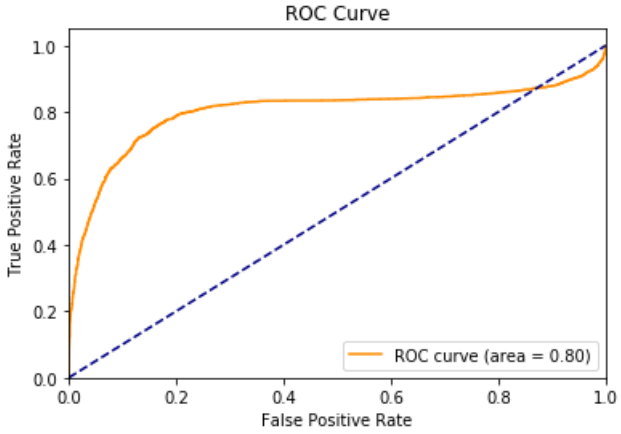

FIGURE 72 – Courbe ROC de New York pour 60 % des données avec l'algorithme utilisant le coefficient de Jaccard pour la prédiction des arêtes

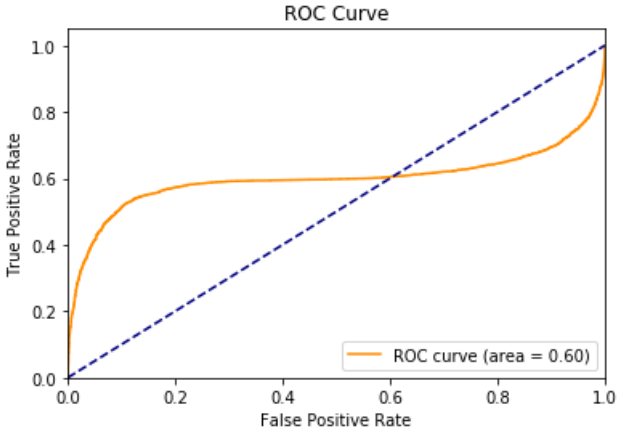

FIGURE 73 – Courbe ROC de Tokyo pour 60 % des données avec l'algorithme utilisant le coefficient de Jaccard pour la prédiction des arêtes

Score AUC pour Tokyo= 0.6105452601501985

Score de précision moyenne pour Tokyo = 0.743091556174591

2. ROC curve et AUC score avec 60% des données

Score AUC pour New York= 0.8078636572301132

Score de précision moyenne pour New York = 0.8564709611419911

Score AUC pour Tokyo= 0.6081328091151332

Score de précision moyenne pour Tokyo = 0.7358046675823764

3. ROC curve et AUC score avec 35% des données

Score AUC pour New York=0.8046417309033693

Score de précision moyenne pour New York = 0.8489541183710129

Score AUC pour Tokyo= 0.6209893305059797

Score de précision moyenne pour Tokyo = 0.7294711295802585

4. Analyse des résultats obtenus

On peut voir que cet algorithme est nettement moins efficace que les précédent cependant il est lui aussi moins influencé par le manque de données sur lesquelles se baser.
(figure 70 à 75) Cependant, le plus surprenant avec cet algorithme est que l'on peut voir

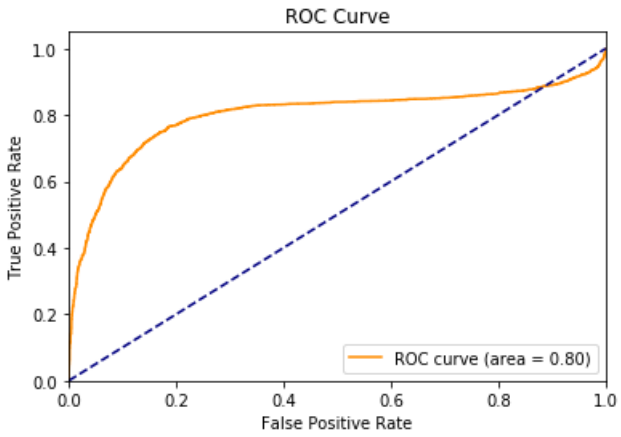

FIGURE 74 – Courbe ROC de New York pour 35 % des données avec l'algorithme utilisant le coefficient de Jaccard pour la prédiction des arêtes

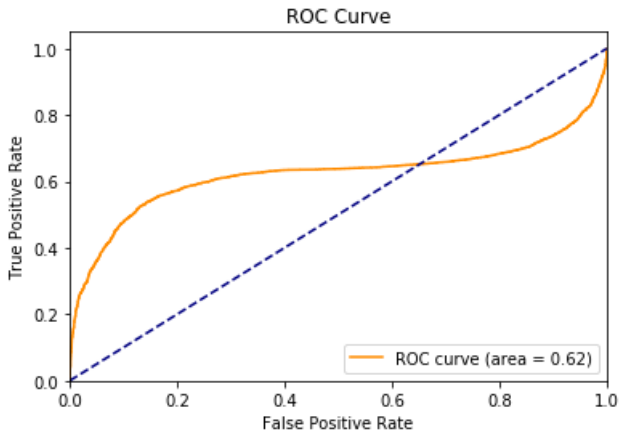

FIGURE 75 – Courbe ROC de Tokyo pour 35 % des données avec l'algorithme utilisant le coefficient de Jaccard pour la prédiction des arêtes

que sur la courbe ROC lorsque le taux de False positive rate, est faible l'algorithme est nettement plus efficace que la plupart, mais diminue très vite pour finalement passer en dessous de la courbe x=y. On notera également que l'algorithme n'est pas très efficace pour la ville de Tokyo.

Cependant on peut noter que cette méthode, bien que coûteuse, ne nécessite pas de connaître le voisinage de chaque noeud et est une méthode basée sur le machine learning se basant sur le graphe entier et établissant des groupements.

Cet algorithme faisant des groupements,on peut supposer qu'il fonctionne moins bien pour Tokyo, car la densité des noeuds en fonction de leur degré ne suit pas exactement une heavy-tailed distribution comme nous le montre la figure 37. Il est donc plus difficile de faire des groupements, car le Laplacien normalisé en sera nettement influencé.[13]

3.6 Conclusions

Après analyse de nos différents graphes, on peut clairement voir que l'index d'allocation de ressources est légèrement supérieur à celui d' AdamicAdar qui est lui même très nettement supérieur à celui du coefficient de Jaccard.

Concernant le score d'attachement préférentiel, l'on peut voir que lorsqu'il doit prédire peu de données à l'aide d'un grand nombre de données sur lesquelles il peut se baser il n'est pas le meilleur des algorithmes. Cependant lorsque celui-ci doit prédire avec moins de données, il s'avère plus efficace que les précédents.

Enfin l'algorithme utilisant le spectre du graphe Laplacien normalisé est moins efficace que les précédents, mais peut s'avérer tout de même efficace lorsque le nombre de données sur lesquelles il se base est réduit.

De plus nous avons pu remarquer que les quatre premiers algorithmes nous donnent des meilleurs résultats pour la ville de Tokyo lorsqu'il y a un moins grand pourcentage de données, car le nombre de données associées à cette ville est supérieur à celle de New York.

Enfin nous avons pu voir que le dernier de nos algorithmes fournissait des résultats moins probants pour Tokyo, car les données pour cette ville une fois remodelées ne correspondaient plus totalement à une heavy-tailed ditribution.

En conclusion, nous avons vu que nous pouvons prédire les déplacements de population à l'aide de ces différents algorithmes et montrer la différence d'efficacité de ces derniers.

4 Conclusions

Dans ce travail nous avons étudié les propriétés du réseaux locaux. Nous avons pu voir quelques propriétés des réseaux locaux telles que la heavy-tailed distribution qui correspond au fait que dans les réseaux locaux il y a beaucoup de noeuds ayant un faible degré et peu ayant un degré très grand. On a pu également voir ce qu'était la fermeture triadique et qu'elle est due notamment à la propriété small-world. Nous avons également vu l'assortativité, c'est-à-dire la tendance d'un noeud à s'associer a un noeud similaire.

Nous avons ensuite pu voir différentes méthodes de prédiction d'arêtes que nous avons utilisées par la suite et les différentes métriques afin de les évaluer.

Nous avons ensuite pu analyser les différentes habitudes des utilisateurs afin de voir vers quel noeud ils se dirigeaient et quels étaient les types des noeuds faibles. Nous avons pu voir aussi la distribution des check-ins en fonction des utilisateurs et des endroits. A l'aide de cela, nous avons pu comprendre ce que représentaient nos données et nous en avons tiré certaines conclusions.

Nous avons pu aussi comprendre en analysant plus amplement le graphe que nos données étaient trop peu nombreuses pour obtenir de bons résultats avec nos algorithmes et qu'il était important de les remanier afin de pouvoir utiliser ceux-ci.

Finalement, nous avons appliqué nos cinq algorithmes à nos données remaniées et après avoir analysé les résultats, nous avons pu voir que l'index d'allocation de ressources est le plus apte à prédire au mieux les déplacements de population. On a également vu qu'avec peu de données le score d'attachement préférentiel qui se base sur une mesure de centralité était le plus efficace des algorithmes. Enfin nous avons vu que le spectre du graphe Laplacien normalisé nous donnait des résultats assez constants même lorsque l'on manquait de données. Bien que ce ne soient pas les meilleurs résultats, il est peut-être intéressant de l'utiliser dans certains cas.

Il est difficile de comparer nos résultats à d'autres articles. Cependant si l'on compare nos résultats à [21],il faut tenir compte du fait qu'il entraine ses algorithmes sur 3 mois pour prédire les 3 mois suivants. De plus, l'entrainement que j'ai utilisé pour mes algorithmes est différent,et la quantité de données à prédire et sur lesquelles se baser diffère également. Cependant, les algorithmes que nous avons utilisés fournissent des résultats aux tests assez élevés même en ne se basant que sur 35% des données.

La AUC et la précision moyenne de chaque algorithme nous montrent l'efficacité de chaque algorithme. Lorsque l'indicateur est supérieur à 0.75, nous pouvons dire que cet algorithme est efficace concernant la prédiction d'arête et est apte à prédire les déplacements de population d'une zone à l'autre avec une précision supérieure à 0.75.

Notre étude s'est limitée à une zone définie qui correspond à une limite liée à nos données. La deuxième limite de ce travail est le nombre d'algorithmes de prédiction de l'ordre de cinq. Il existe de nombreux algorithmes de prédiction et une perspective de ce travail est de le continuer en se servant d'autres algorithmes. On peut citer les algorithmes

1. se servant de la distance entre les zones calculées en fonction de la latitude et de la longitude.

2. de machines learning autre que celui utilisé.

3. se basant sur les types de lieu en vue de les associer.

4. utilisant des indices de similarité globale comme le Katz index.

5. de deep learning si la base de données est plus conséquente.

Références

[1] Lada A. Adamic and Eytan Adar. Friends and neighbors on the web. Social networks, 2003.

[2] Daniel A. Schult Aric A. Hagberg and Pieter J. Swart. "exploring network structure, dynamics, and function using networkx". pages 11–15, 08 2008.

[3] Rémi Bachelet. Réseaux sociaux. `http://rb.ec-lille.fr/l/Socio_orgas/cours-socio_reseaux_sociaux.pdf`.

[4] Juan Fuxman Bass, Alos Diallo, Justin Nelson, Juan M Soto, Chad L Myers, and Albertha J M Walhout. Using networks to measure similarity between genes : association index selection. Nat Methods, March 2014.

[5] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. 390 :1373–1396, 3 2003.

[6] Eunjoon Cho, Seth A. Myers, and Jure Leskovec. Friendship and mobility : User movement in location-based social networks. In Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '11, pages 1082–1090, New York, NY, USA, 2011. ACM.

[7] Dennis Crowley, Jeff Glueck, and Steven Rosenblatt. Foursquare : A propos de nous. `https://fr.foursquare.com/about`.

[8] J. Kleinberg D. Liben-Nowell. The link prediction problem for social networks. 2004.

[9] Peter Flach and Meelis Kull. Precision-recall-gain curves : Pr analysis done right. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, Advances in Neural Information Processing Systems 28, pages 838–846. Curran Associates, Inc., 2015.

[10] K Hajian-Tilaki. Receiver operating characteristic (roc) curve analysis for medical diagnostic test evaluation. 4.2 :627–635, 2013.

[11] Tseng Yu-Hwei Jobin Paul. « le suicide comme karōshi ou l'overdose de travail. les suicides liés au travail au japon, à taiwan et en chine ». 31 :45–88, 2014.

[12] Thomas N Kipf and Max Welling. Variational graph auto-encoders. NIPS Workshop on Bayesian Deep Learning, 2016.

[13] Andrew V. Knyazev. Toward the optimal preconditioned eigensolver : Locally optimal block preconditioned conjugate gradient method. volume 23.2, page 517–541. SIAM Journal on Scientific Computing and Society for Industrial and Applied Mathematic, 2015.

[14] Linyuan Lü and Tao Zhou. Link prediction in complex networks : A survey. 390, 10 2010.

[15] Dablanc Laetitia. « le territoire urbain des konbini et des takkyubin au japon ». 78.4 :68–70, 2009.

[16] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time : Densification laws, shrinking diameters and possible explanations. In Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining, KDD '05, pages 177–187, New York, NY, USA, 2005. ACM.

[17] Aveline Natasha. La ville et le rail au Japon : l'expansion des groupes ferroviaires privés à Tokyo et Osaka. Paris, 2003.

[18] M. E. Newman. Assortative Mixing in Networks. Physical Review Letters, 89(20) :208701, October 2002.

[19] Anastasios Noulas, Salvatore Scellato, Neal Lathia, and Cecilia Mascolo. A random walk around the city : New venue recommendation in location-based social networks. 09 2012.

[20] Anastasios Noulas, Salvatore Scellato, Cecilia Mascolo, and Massimiliano Pontil. An empirical study of geographic user activity patterns in foursquare. 01 2011.

[21] Anastasios Noulas, Blake Shaw, Renaud Lambiotte, and Cecilia Mascolo. Topological properties and temporal dynamics of place networks in urban environments. CoRR, abs/1502.07979, 2015.

[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn : Machine learning in Python. Journal of Machine Learning Research, 12 :2825–2830, 2011.

[23] Adam Sadilek, Henry Kautz, and Jeffrey P. Bigham. Finding your friends and following them to where you are. In Proceedings of the Fifth ACM International Conference on Web Search and Data Mining, WSDM '12, pages 723–732, New York, NY, USA, 2012. ACM.

[24] Salvatore Scellato, Anastasios Noulas, and Cecilia Mascolo. Exploiting place features in link prediction on location-based social networks. In Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '11, pages 1046–1054, New York, NY, USA, 2011. ACM.

[25] Guide social média. Wellconnect : Foursquare comment-ca marche ? `http://guidesocialmedia.com/2012/05/foursquare-comment-ca-marche/`.

[26] Y.-C. Zhang T. Zhou, L. Lu. Predicting missing links via local information. Eur. Phys. J. B, 71, 2009.

[27] Dingqi Yang, Daqing Zhang, Vincent. W. Zheng, and Zhiyong Yu. Modeling user activity preference by leveraging user spatial temporal characteristics in lbsns. IEEE Transactions on Systems, Man, and Cybernetics : Systems, 45(1) :129–142, 2015.

UNIVERSITE DE NAMUR

Faculté des Sciences

ANALYSES ET PREDICTIONS DE DEPLACEMENTS DE POPULATION A NEW YORK ET TOKYO BASE SUR LES PROPRIETES DES RESEAUX

ANNEXE

Mémoire présenté pour l'obtention

du grade académique de master en didactique des mathématiques

Charles DELMOTTE

Août 2018

Annexes

Link prédiction méthode

Programme principal

```
import networkx as nx
import link_prediction_scores as lp



RANDOM_SEED = 0
FED= [0.25, 0.5, 0.75]


G = nx.Graph()
G1 = nx.Graph()


convDict={}# liste de la position de ces personnes
pers = []#liste de toutes les personnes


G.clear()
with open("dataset_TSMC2014_NYC.txt", "r") as file:#ouverture du fichier
   for line in file:#parcourir le fichier
        vecline = line.rstrip().split("         ")
        #print(vecline)


        if  (vecline[4][0:5],vecline[5][0:6]) not in G.nodes():# Si non-présent  dans la liste des noeuds
           #nodes1.append(vecline[1])
           G.add_node((vecline[4][0:5],vecline[5][0:6]))#ajout du noeud
        if vecline[0] not in pers:# Si la personne a déjà contribut
           pers.append(vecline[0])#liste des personnes rencontree(une fois chaque)
           convDict[vecline[0]]=(vecline[4][0:5],vecline[5][0:6])# position de ces personnes
        else :
           if ((convDict[vecline[0]],(vecline[4][0:5],vecline[5][0:6])) not in G.edges()):# Si le
déplacement est déjà dans les arêtes
```

```
            G.add_edge(convDict[vecline[0]],(vecline[4][0:5],vecline[5][0:6]),weight=1)  # ajout d'une arête de poids 1
            convDict[vecline[0]]=(vecline[4][0:5],vecline[5][0:6]) # enregistrement de la nouvelle position
        else:
            G[convDict[vecline[0]]][(vecline[4][0:5],vecline[5][0:6])]['weight'] +=1 #incrémentation du poids
            convDict[vecline[0]]=(vecline[4][0:5],vecline[5][0:6]) # enregistrement de la nouvelle position
G=nx.Graph(G)
G.remove_edges_from(G.selfloop_edges())     #nettoyage du graphique pour éviter certains problèmes de compilation
#parite de code similaire à celui au-dessus
convDict={}
pers = []
G1.clear()
with open("dataset_TSMC2014_TKY.txt", "r") as file:#ouverture du fichier
   for line in file:#parcourir le fichier
        vecline = line.rstrip().split("\t")
        if  (vecline[4][0:5],vecline[5][0:6]) not in G1.nodes():# si on est au debut ou si on change de pays
            #nodes1.append(vecline[1])#liste des regions rencontree(une fois chaque)
            G1.add_node((vecline[4][0:5],vecline[5][0:6]))
        if vecline[0] not in pers:# si on est au debut ou si on change de pays
            pers.append(vecline[0])#liste des regions rencontree(une fois chaque)
            convDict[vecline[0]]=(vecline[4][0:5],vecline[5][0:6])
        else :
            if ((convDict[vecline[0]],(vecline[4][0:5],vecline[5][0:6])) not in G1.edges()):
                G1.add_edge(convDict[vecline[0]],(vecline[4][0:5],vecline[5][0:6]),weight=1)
                convDict[vecline[0]]=(vecline[4][0:5],vecline[5][0:6])
            else:
                G1[convDict[vecline[0]]][(vecline[4][0:5],vecline[5][0:6])]['weight'] +=1
                convDict[vecline[0]]=(vecline[4][0:5],vecline[5][0:6])
```

```
G1=nx.Graph(G1)
G1.remove_edges_from(G1.selfloop_edges())




### ---------- Run Link Prediction Tests ---------- ###



nx_results = {}# résultats de nos différentes méthodes de prédiction d'arête




   # Iterate over fractions of edges to hide
for frac_hidden in FED:
   val_frac = 0.1
   test_frac = frac_hidden - val_frac

   adj = nx.adjacency_matrix(G)

   experiment_name = 'G'
   print("Current experiment: ", experiment_name)

        # Run all link prediction methods on current graph, store results
   nx_results[experiment_name] = lp.calculate_all_scores(adj, \
                                        test_frac=test_frac, val_frac=val_frac, \
                                        random_state=RANDOM_SEED, verbose=1)
   adj = nx.adjacency_matrix(G1)

   experiment_name = 'G1'
   print("Current experiment: ", experiment_name)
```

```
        # Run all link prediction methods on current graph, store results
    nx_results[experiment_name] = lp.calculate_all_scores(adj, \
                                            test_frac=test_frac, val_frac=val_frac, \
                                            random_state=RANDOM_SEED, verbose=1)
```

Algorithme de prédiction d'arêtes

```
from __future__ import division
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import roc_auc_score, average_precision_score, roc_curve
from sklearn.manifold import spectral_embedding,locally_linear_embedding
from sklearn import linear_model
import time
import tensorflow as tf
from gae.preprocessing import mask_test_edges

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Input: positive test/val edges, negative test/val edges, edge score matrix
# Output: ROC AUC score, ROC Curve (FPR, TPR, Thresholds), AP score
def get_roc_score(edges_pos, edges_neg, score_matrix, apply_sigmoid=False):

    # Edge case
    if len(edges_pos) == 0 or len(edges_neg) == 0:
        return (None, None, None)

    # Enregistre les prédictions  positive de nos arêtes
    preds_pos = []
    pos = []
    for edge in edges_pos:
        if apply_sigmoid == True:
            preds_pos.append(sigmoid(score_matrix[edge[0], edge[1]]))
        else:
            preds_pos.append(score_matrix[edge[0], edge[1]])
```

```
    pos.append(1) # actual value (1 pour valeur positive)

# Enregistre les prédiction négative
preds_neg = []
neg = []
for edge in edges_neg:
    if apply_sigmoid == True:
        preds_neg.append(sigmoid(score_matrix[edge[0], edge[1]]))
    else:
        preds_neg.append(score_matrix[edge[0], edge[1]])
    neg.append(0) # actual value (0 pour valeur negative)

# Calcul du score et affichag de la courbe ROC
preds_all = np.hstack([preds_pos, preds_neg])
labels_all = np.hstack([np.ones(len(preds_pos)), np.zeros(len(preds_neg))])
roc_score = roc_auc_score(labels_all, preds_all)
fpr, tpr, thresholds = roc_curve(labels_all, preds_all)
ap_score = average_precision_score(labels_all, preds_all)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', label='ROC curve (area = %0.2f)' % roc_score)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()

# return roc_score, roc_curve_tuple, ap_score
return roc_score, ap_score
```

```
# retoune une liste de tuples (node1,node2) pour les algorithme de prédiction de Networkx
def get_ebunch(train_test_split):
    adj_train, train_edges, train_edges_false, val_edges, val_edges_false, \
        test_edges, test_edges_false = train_test_split

    test_edges_list = test_edges.tolist() # convertir en une liste imbriquée
    test_edges_list = [tuple(node_pair) for node_pair in test_edges_list] # conversion en pair de noeud
    test_edges_false_list = test_edges_false.tolist()
    test_edges_false_list = [tuple(node_pair) for node_pair in test_edges_false_list]
    return (test_edges_list + test_edges_false_list)

# Input: NetworkX training graph, train_test_split (from mask_test_edges)
# Output: dictionary with ROC AUC, ROC Curve, AP, Runtime
def adamic_adar_scores(g_train, train_test_split):

    adj_train, train_edges, train_edges_false, val_edges, val_edges_false, \
        test_edges, test_edges_false = train_test_split # Unpack input

    start_time = time.time()

    aa_scores = {}

    # Calcul du score d'AdamicAdar
    aa_matrix = np.zeros(adj_train.shape)
    for u, v, p in nx.adamic_adar_index(g_train, ebunch=get_ebunch(train_test_split)): # (u, v) = couple de noeuds, p =  index d'Adamic-Adar
        aa_matrix[u][v] = p
        #aa_matrix[v][u] = p # afin d'être sur que la matrice est symétrique
    aa_matrix = aa_matrix / aa_matrix.max() # matrice normalisée
```

```
    runtime = time.time() - start_time #calcul du temps de l'algorithme
    aa_roc, aa_ap = get_roc_score(test_edges, test_edges_false, aa_matrix) #récupération des scores pour l'index AdamicAdar

    aa_scores['test_roc'] = aa_roc
    aa_scores['test_ap'] = aa_ap
    aa_scores['runtime'] = runtime
    return aa_scores

# Input: NetworkX training graph, train_test_split (from mask_test_edges)
# Output: dictionary with ROC AUC, ROC Curve, AP, Runtime
def ressource_allocation_scores(g_train, train_test_split):

    adj_train, train_edges, train_edges_false, val_edges, val_edges_false, \
        test_edges, test_edges_false = train_test_split # Unpack input

    start_time = time.time()

    ra_scores = {}

    # Calcul du score
    ra_matrix = np.zeros(adj_train.shape)
    for u, v, p in nx.resource_allocation_index(g_train, ebunch=get_ebunch(train_test_split)): # (u, v) = couple de noeuds, p = Adamic-Adar index
        ra_matrix[u][v] = p
        #ra_matrix[v][u] = p
    ra_matrix = ra_matrix / ra_matrix.max() # matrice normalisée

    runtime = time.time() - start_time
    ra_roc, ra_ap = get_roc_score(test_edges, test_edges_false, ra_matrix)

    ra_scores['test_roc'] = ra_roc
```

```
    ra_scores['test_ap'] = ra_ap
    ra_scores['runtime'] = runtime
    return ra_scores

# Input: NetworkX training graph, train_test_split (from mask_test_edges)
# Output: dictionary with ROC AUC, ROC Curve, AP, Runtime
def jaccard_coefficient_scores(g_train, train_test_split):
    if g_train.is_directed(): # Jaccard coef only works for undirected graphs
        g_train = g_train.to_undirected()

    adj_train, train_edges, train_edges_false, val_edges, val_edges_false, \
        test_edges, test_edges_false = train_test_split # Unpack input

    start_time = time.time()
    jc_scores = {}

    # Calcul des scores
    jc_matrix = np.zeros(adj_train.shape)
    for u, v, p in nx.jaccard_coefficient(g_train, ebunch=get_ebunch(train_test_split)): # (u, v) = couple de noeuds, p = Jaccard coefficient
        jc_matrix[u][v] = p
        #jc_matrix[v][u] = p
    jc_matrix = jc_matrix / jc_matrix.max()

    runtime = time.time() - start_time
    jc_roc, jc_ap = get_roc_score(test_edges, test_edges_false, jc_matrix)

    jc_scores['test_roc'] = jc_roc
    jc_scores['test_ap'] = jc_ap
    jc_scores['runtime'] = runtime
    return jc_scores
```

```
# Input: NetworkX training graph, train_test_split (from mask_test_edges)
# Output: dictionary with ROC AUC, ROC Curve, AP, Runtime
def preferential_attachment_scores(g_train, train_test_split):

    adj_train, train_edges, train_edges_false, val_edges, val_edges_false, \
        test_edges, test_edges_false = train_test_split # Unpack input

    start_time = time.time()
    pa_scores = {}

    # Calcules des scores
    pa_matrix = np.zeros(adj_train.shape)
    for u, v, p in nx.preferential_attachment(g_train, ebunch=get_ebunch(train_test_split)): # (u, v) = couple de noeuds, p = préférential attachement score
        pa_matrix[u][v] = p
        #pa_matrix[v][u] = p
    pa_matrix = pa_matrix / pa_matrix.max()

    runtime = time.time() - start_time
    pa_roc, pa_ap = get_roc_score(test_edges, test_edges_false, pa_matrix)

    pa_scores['test_roc'] = pa_roc
    pa_scores['test_ap'] = pa_ap
    pa_scores['runtime'] = runtime
    return pa_scores


# Input: train_test_split (from mask_test_edges)
# Output: dictionary with ROC AUC, ROC Curve, AP, Runtime
```

```
def spectral_clustering_scores(train_test_split, random_state=0):
    adj_train, train_edges, train_edges_false, val_edges, val_edges_false, \
        test_edges, test_edges_false = train_test_split # Unpack input

    start_time = time.time()
    sc_scores = {}

    # Dpectral clustering link prediction
    spectral_emb = spectral_embedding(adj_train, n_components=16, random_state=random_state)
    sc_score_matrix = np.dot(spectral_emb, spectral_emb.T)

    runtime = time.time() - start_time
    sc_test_roc, sc_test_ap = get_roc_score(test_edges, test_edges_false, sc_score_matrix,
apply_sigmoid=True)
    sc_val_roc, sc_val_ap = get_roc_score(val_edges, val_edges_false, sc_score_matrix,
apply_sigmoid=True)

    # enregristrement des scores
    sc_scores['test_roc'] = sc_test_roc
    sc_scores['test_ap'] = sc_test_ap

    sc_scores['val_roc'] = sc_val_roc
    sc_scores['val_ap'] = sc_val_ap

    sc_scores['runtime'] = runtime
    return sc_scores



# Input: adjacency matrix (in sparse format), features_matrix (normal format), test_frac, val_frac,
verbose
    # Verbose: 0 - N'affichera aucun des résultats, 1 - Affichage des résultats,
```

```
# Returns: Dictionary of results (ROC AUC, ROC Curve, AP, Runtime) pour chaque méthode de prédiction d'arête
def calculate_all_scores(adj_sparse, features_matrix=None, directed=False, \
    test_frac=.3, val_frac=.1, random_state=0, verbose=1, \
    train_test_split_file=None,
    tf_dtype=tf.float32):
  np.random.seed(random_state) # Guarantee consistent train/test split
  tf.set_random_seed(random_state) # Consistent GAE training

  # Préparation du dictionnaires des résultats
  lp_scores = {}

  ### ---------- PREPROCESSING ---------- ###
  train_test_split = None

  train_test_split = mask_test_edges(adj_sparse, test_frac=test_frac, val_frac=val_frac)

  adj_train, train_edges, train_edges_false, val_edges, val_edges_false, \
    test_edges, test_edges_false = train_test_split # Unpack tuple

  # g_train: nouveau graphe ne contenant pas les arêtes cachées

  g_train = nx.Graph(adj_train)

  ### ---------- LINK PREDICTION BASELINES ---------- ###
  # Adamic-Adar
  aa_scores = adamic_adar_scores(g_train, train_test_split)
  lp_scores['aa'] = aa_scores
  if verbose == 1:
    print ('')
    print ('Adamic-Adar Test ROC score: ', str(aa_scores['test_roc']))
```

```
    print ('Adamic-Adar Test AP score: ', str(aa_scores['test_ap']))


  # Jaccard Coefficient
  jc_scores = jaccard_coefficient_scores(g_train, train_test_split)
  lp_scores['jc'] = jc_scores
  if verbose == 1:
    print ('')
    print ('Jaccard Coefficient Test ROC score: ', str(jc_scores['test_roc']))
    print ('Jaccard Coefficient Test AP score: ', str(jc_scores['test_ap']))


  # Preferential Attachment
  pa_scores = preferential_attachment_scores(g_train, train_test_split)
  lp_scores['pa'] = pa_scores
  if verbose == 1:
    print ('')
    print ('Preferential Attachment Test ROC score: ', str(pa_scores['test_roc']))
    print ('Preferential Attachment Test AP score: ', str(pa_scores['test_ap']))


   # Ressource Allocation
  ra_scores = ressource_allocation_scores(g_train, train_test_split)
  lp_scores['ra'] = ra_scores
  if verbose == 1:
    print ('')
    print ('Ressource Allocation Test ROC score: ', str(ra_scores['test_roc']))
    print ('Ressource Allocation Test AP score: ', str(ra_scores['test_ap']))



### ---------- SPECTRAL CLUSTERING ---------- ###
  sc_scores = spectral_clustering_scores(train_test_split)
  lp_scores['sc'] = sc_scores
  if verbose == 1:
```

```
    print ('')
    print ('Spectral Clustering Validation ROC score: ', str(sc_scores['val_roc']))
    print ('Spectral Clustering Validation AP score: ', str(sc_scores['val_ap']))
    print ('Spectral Clustering Test ROC score: ', str(sc_scores['test_roc']))
    print ('Spectral Clustering Test AP score: ', str(sc_scores['test_ap']))


return lp_scores
```

Algorithme de fractionnement des arêtes en « test and train set » basé sur https://github.com/tkipf/gae/blob/master/gae/preprocessing.py.

Ce code étant soumis à une license. Toute copie même modifiée de ce code doit contenir cette license*

```
import numpy as np
import scipy.sparse as sp
import networkx as nx

# Conversion de la matrice en tuple
def sparse_to_tuple(sparse_mx):
    if not sp.isspmatrix_coo(sparse_mx):
        sparse_mx = sparse_mx.tocoo()
    coords = np.vstack((sparse_mx.row, sparse_mx.col)).transpose()
    values = sparse_mx.data
    shape = sparse_mx.shape
    return coords, values, shape

# Normalisation de l'adjacency matrix: A_norm
def preprocess_graph(adj):
    adj = sp.coo_matrix(adj)
    adj_ = adj + sp.eye(adj.shape[0])
    rowsum = np.array(adj_.sum(1))
    degree_mat_inv_sqrt = sp.diags(np.power(rowsum, -0.5).flatten())
    adj_normalized =
adj_.dot(degree_mat_inv_sqrt).transpose().dot(degree_mat_inv_sqrt).tocoo()
    return sparse_to_tuple(adj_normalized)

# Séparation en test et train
    # Adjacency matrix in sparse format
    # Returns: adj_train, train_edges, val_edges, val_edges_false,
        # test_edges, test_edges_false
def mask_test_edges(adj, test_frac=.1, val_frac=.05, prevent_disconnect=True,
verbose=False):
```

```
    if verbose == True:
        print ('preprocessing...')

    # Suppression des éléments diagonaux
    adj = adj - sp.dia_matrix((adj.diagonal()[np.newaxis, :], [0]), shape=adj.shape)
    adj.eliminate_zeros()
    # Vérification que la diagonal vaut 0
    assert np.diag(adj.todense()).sum() == 0

    g = nx.from_scipy_sparse_matrix(adj)
    orig_num_cc = nx.number_connected_components(g)

    adj_triu = sp.triu(adj) # Partie triangulaire supérieure de l' adjency matrix
    adj_tuple = sparse_to_tuple(adj_triu) # (coords, valeurs, forme), les arêtes non pas de direction
    edges = adj_tuple[0] # tout les noeuds sont cités une seule fois ( not 2 ways car l'algorithme concerne les graphiques non orienté)
    num_test = int(np.floor(edges.shape[0] * test_frac)) # Contrôle combien le test devrait être
    num_val = int(np.floor(edges.shape[0] * val_frac)) # contrôle combien le test de validation devrait être

    # Enregistre les arêtes en une liste ordonnée de tuple (node1,node2) où node1 < node2
    edge_tuples = [(min(edge[0], edge[1]), max(edge[0], edge[1])) for edge in edges]
    all_edge_tuples = set(edge_tuples)
    train_edges = set(edge_tuples) # Initialise le fait que les arêtes d'entrainement correspondent à toutes les arêtes
    test_edges = set()
    val_edges = set()

    if verbose == True:
```

```
    print ('generating test/val sets...')

  # Itère sur les arêtes mélangées et l'ajoute à l'ensemble train/val
  np.random.shuffle(edge_tuples)
  for edge in edge_tuples:
    node1 = edge[0]
    node2 = edge[1]

    # Si le fait d'enlever une arête déconnecte un noeud, l'algorithme revient en arrière et continue
    g.remove_edge(node1, node2)
    if prevent_disconnect == True:
      if nx.number_connected_components(g) > orig_num_cc:
        g.add_edge(node1, node2)
        continue

    # Remplis l'ensemble des arêtes de test : test_edges
    if len(test_edges) < num_test:
      test_edges.add(edge)
      train_edges.remove(edge)

    # remplis les val_edges
    elif len(val_edges) < num_val:
      val_edges.add(edge)
      train_edges.remove(edge)

    # Lorsque les deux ensemble son remplis on arête la boucle
    elif len(test_edges) == num_test and len(val_edges) == num_val:
      break
```

```
    if (len(val_edges) < num_val or len(test_edges) < num_test):
        print ("WARNING: not enough removable edges to perform full train-test split!")
        print ("Num. (test, val) edges requested: (", num_test, ", ", num_val, ")")
        print ("Num. (test, val) edges returned: (", len(test_edges), ", ", len(val_edges), ")")

    if prevent_disconnect == True:
        assert nx.number_connected_components(g) == orig_num_cc

    if verbose == True:
        print ('creating false test edges...')

    test_edges_false = set()
    while len(test_edges_false) < num_test:
        idx_i = np.random.randint(0, adj.shape[0])
        idx_j = np.random.randint(0, adj.shape[0])
        if idx_i == idx_j:
            continue

        false_edge = (min(idx_i, idx_j), max(idx_i, idx_j))

        # On s'assure que les fausse arêtes ne sont pas des arêtes déjà ajoutées ou présentes
        if false_edge in all_edge_tuples:
            continue
        if false_edge in test_edges_false:
            continue

        test_edges_false.add(false_edge)

    if verbose == True:
```

```
    print ('creating false val edges...')

  val_edges_false = set()
  while len(val_edges_false) < num_val:
    idx_i = np.random.randint(0, adj.shape[0])
    idx_j = np.random.randint(0, adj.shape[0])
    if idx_i == idx_j:
      continue

    false_edge = (min(idx_i, idx_j), max(idx_i, idx_j))

    # On s'assure que les fausse arêtes ne sont pas des arêtes déjà ajoutées ou présentes ni dans l'ensemble test_edges_false
    if false_edge in all_edge_tuples or \
      false_edge in test_edges_false or \
      false_edge in val_edges_false:
      continue

    val_edges_false.add(false_edge)

  if verbose == True:
    print ('creating false train edges...')

  train_edges_false = set()
  while len(train_edges_false) < len(train_edges):
    idx_i = np.random.randint(0, adj.shape[0])
    idx_j = np.random.randint(0, adj.shape[0])
    if idx_i == idx_j:
      continue
```

```
        false_edge = (min(idx_i, idx_j), max(idx_i, idx_j))

        # On s'assure que les fausse arêtes ne sont pas des arêtes déjà ajoutées ou présentes ni dans l'ensemble test_edges_false et val_edges_false
        if false_edge in all_edge_tuples or \
            false_edge in test_edges_false or \
            false_edge in val_edges_false or \
            false_edge in train_edges_false:
            continue

        train_edges_false.add(false_edge)

    if verbose == True:
        print ('final checks for disjointness...')

    # Vérification que les falses_edges sont actuellement faux

    assert test_edges_false.isdisjoint(all_edge_tuples)
    assert val_edges_false.isdisjoint(all_edge_tuples)
    assert train_edges_false.isdisjoint(all_edge_tuples)

    # assert: On s'assure que les ensembles de fausses arêtes sont disjoints
    assert test_edges_false.isdisjoint(val_edges_false)
    assert test_edges_false.isdisjoint(train_edges_false)
    assert val_edges_false.isdisjoint(train_edges_false)

    # assert:On s'assure que l'ensemble des arêtes positives est disjoint
    assert val_edges.isdisjoint(train_edges)
    assert test_edges.isdisjoint(train_edges)
    assert val_edges.isdisjoint(test_edges)
```

```
    if verbose == True:
        print ('creating adj_train...')

    # Reconstruction de l'adjency matrix avec les arêtes restantes
    adj_train = nx.adjacency_matrix(g)

    # Conversion de la liste d'arête en tableau numpy
    train_edges = np.array([list(edge_tuple) for edge_tuple in train_edges])
    train_edges_false = np.array([list(edge_tuple) for edge_tuple in train_edges_false])
    val_edges = np.array([list(edge_tuple) for edge_tuple in val_edges])
    val_edges_false = np.array([list(edge_tuple) for edge_tuple in val_edges_false])
    test_edges = np.array([list(edge_tuple) for edge_tuple in test_edges])
    test_edges_false = np.array([list(edge_tuple) for edge_tuple in test_edges_false])

    if verbose == True:
        print ('Done with train-test split!')
        print ('')

    return adj_train, train_edges, train_edges_false, \
        val_edges, val_edges_false, test_edges, test_edges_false
```